Design and Development of Component-based Adaptive Web Applications

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

M.Sc. Zoltán Fiala
geboren am 19. März 1977 in Budapest

Gutachter:

Prof. Dr.-Ing. Klaus Meißner (Technische Universität Dresden)
Prof. Dr. rer. nat. habil. Uwe Aßmann (Technische Universität Dresden)
Prof. Dr. ir. Geert-Jan Houben (Vrije Universiteit Brussel)

Tag der Verteidigung: 19. Februar 2007

Dresden im Februar 2007

Summary

The WWW is rapidly evolving to a ubiquitous information and application medium. Formerly a collection of static HTML pages, today's Web sites are complex *Web Information Systems* offering large amounts of content and functionality. Their growing audience is characterized by heterogeneous goals, preferences, and capabilities. Furthermore, people access the Web from a diversity of locations and devices. These trends necessitate *adaptive Web sites* that automatically adjust their content, navigation, and presentation to their usage context.

This need for adaptation implies additional requirements for the already complex development process of Web applications. Still, even though current Web design methods already address limited adaptation issues at design time, existing Web document formats do not provide a sufficient implementation base for structured adaptation engineering. The missing support for a clear separation of different application and adaptation concerns prevents the efficient reuse of configurable implementation artefacts for different platforms and contexts.

This dissertation addresses the aforementioned shortcomings by combining the benefits of model-based Web design methods with the advantages of component-based implementation techniques for efficiently engineering *adaptive* Web sites. The main goal is the intuitive composition of context-dependent Web applications from declarative, reusable, and adaptable components, aided by a systematic development process and appropriate tool support.

After a thorough review of related Web engineering approaches, a novel, *concern-oriented component model* is presented for adaptive Web applications. It is based on the notion of declarative *document components* that encapsulate separate application and adaptation aspects (e.g. content, structure, navigation, semantics, presentation) on different abstraction levels. Document components contain inherent adaptation rules, allowing to realize numerous hypermedia adaptation techniques. Composed to complex document structures, they can be automatically published to different output formats and client platforms, adapted to the current usage context. For the systematic development of component-based adaptive Web applications a *multi-stage, model-based authoring process* and a visual authoring tool are presented. The resulting engineering process supports different kinds of static and dynamic adaptation at both design and implementation level. Its practical applicability for the systematic development of dynamic multimedia Web Information Systems is demonstrated and thus constructively validated by a number of application prototypes. Finally, it is investigated how the lessons learned from authoring component-based adaptive Web applications can be applied to adapt already existing legacy Web-based systems. As a generalization of the proposed component-based approach, the *Generic Adaptation Component* is presented for decoupling and adding selected adaptation concerns to XML-based Web applications.

Acknowledgments

First of all, I would like to thank my advisor, Prof. Dr.-Ing. Klaus Meißner, for supervising and reviewing this dissertation. I thank him for his overall support throughout the last years and the opportunity to research and teach at the Multimedia Technology Group.

Second, I would like to thank Prof. Dr. ir. Geert-Jan Houben for the fruitful collaboration we had and for his readiness to review this thesis. The cooperation with him was a great source of inspiration for this work, leading to valuable discussions, new research ideas, and joint publications.

Third, I would like to thank Prof. Dr. rer. nat. habil. Uwe Aßmann for being a reviewer of the thesis. I benefited a lot from our discussions on component technology, as well as his valuable comments and remarks.

Special thanks go out to the whole Multimedia Technology Group. Most particularly, I would like to thank Michael Hinz for the pleasant and fruitful cooperation in the AMACONT project, Frank Wehner and Raimund Dachselt for countless research discussions and comments regarding this work, and Annett Mitschick for thoroughly reading and commenting parts of the thesis. Furthermore, I thank Ramona Behling and Udo Wähner for providing the administrative and technical background of my research work.

I also would like to thank all students who contributed to the implementation of the concepts described in this thesis. Especially, I would like to mention Matthias Niederhausen, Vincent Tietz, and Norbert Kopcsek.

I had a very fruitful collaboration with members of the Hera research group from the Technische Universiteit Eindhoven and the Vrije Universiteit Brussel: Flavius Frasincar, Sven Casteleyn, Kees van der Sluijs, and Peter Barna.

Finally, I thank József Váradi for polishing the English language used in this work as a native speaker.

Contents

List of Figures

List of Tables

List of Source Code Examples

Chapter 1

Introduction

> *"The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man."*[1]

1.1 Background and Motivation

Since the emergence of the World Wide Web (WWW), the size of its audience and the amount of information published on it have grown enormously. The survey "How much information?", carried out in the end of 2003 at the University of California at Berkeley, reports of an Internet community counting more than 600 million members, 167 Terabytes of published Web content, and prognosticates an annual doubling for the future [Lyman et al. 2003]. Another important trend is the Web's evolution from a simple presentation medium to a dynamic medium of interaction and communication. Whereas the first Web sites consisted of a small collection of HTML pages presenting mainly static content, today's sites act as complex *Web Information Systems (WIS)* that provide both up-to-date information and functionality. It is no exaggeration to claim that the World Wide Web has changed the way of accessing and processing information fundamentally. Meanwhile, it offers electronic services in a number of different application areas, among them the press, education, culture, entertainment, tourism, commerce, administration, etc.

This heterogeneity characterizes not only the application areas of the WWW, but also its growing community. Even though originally developed for the academic field, the Web has quickly found its way to the general public. Today's Web users significantly differ in age, education, preferences, interests, capabilities, cultural background, etc. Furthermore, they access the Web from a growing diversity of locations and client devices. Besides traditional desktop computers and Web browsers, the usage of mobile appliances (e.g. smart phones, PDAs, notebooks, set top boxes) is rapidly increasing, each provided with different presentation, communication, and interaction features.

These trends necessitate the appropriate selection, generation, and delivery of up-to-date information that is automatically adjusted to the user, the capabilities of his client device, and his entire *usage context*. The result is *adaptive Web sites*: sites that are customized (personalized) to better meet the specific requirements, preferences, and characteristics of their audience. Typically, this customization needs to concern different aspects of a Web application: the *data* it presents, the *navigational structure* and it offers on top of this data in terms of interlinked Web pages, and the visual *presentation* of those Web pages. To efficiently cope with this multitude of adaptation issues has become a significant challenge

[1] George Bernard Shaw (1856 - 1950), Man and Superman (1903), Maxims for Revolutionists

for today's Web site providers and developers. It implies additional costs and efforts in the design, implementation, and maintenance of Web information systems.

The development of early Web-based systems was typically characterized by the ad hoc authoring of (mainly static) Web documents. Still, the growing complexity of Web applications has soon made clear that such a straightforward approach is not sufficient when creating and maintaining complex Web sites. The recently emerged *Web engineering* research field tackles this problem by the "establishment and use of sound scientific, engineering and management principles" to Web site development [Murugesan and Deshpande 2001]. Inspired by the principles of traditional software engineering, Web engineering aims at adopting its well-proven methods and concepts to the specific characteristics of Web-based systems. Furthermore, triggered by the aforementioned demand for personalization and device independence, it places an increasing emphasis on considering the specific requirements for engineering adaptive Web sites. This additional consideration of adaptation (*adaptation engineering*) concerns different phases of a Web application's overall life-cycle: its design, implementation, publication, maintenance, testing, evolution, etc. Furthermore, besides the structured development of "new" adaptive Web sites, a research question gaining always more importance is how already existing Web information systems can be extended with additional adaptation functionality.

A basic observation in the Web engineering field was that Web-based applications lacking a systematic underlying design suffer from enormous usability and manageability problems [Murugesan et al. 2001]. Thus, in the last decade, several Web design methods have been proposed by academia (RMM [Isakowitz et al. 1995], OOHDM [Schwabe et al. 1996], WebML [Ceri et al. 2000], WSDM [De Troyer 2001], Hera [Vdovjak et al. 2003], etc.). Their main goal is to simplify the development of Web sites by abstracting from the implementation and separately considering different design aspects. While applying different techniques and notations, a common characteristic of all design methods is to distinguish between a data, a navigation, and a presentation design. Selected methods (e.g. OOHDM, WebML, WSDM, Hera) also offer some support for personalization and adaptation at design-level. However, the provided adaptation is mostly centered around certain content and navigation adaptation aspects. Important context-dependency issues, such as media adaptation or (dynamic) adaptation at presentation design have not been sufficiently addressed, yet. Moreover, there is only limited tool support for the visual specification of adaptation and the (semi-)automatic generation of a corresponding adaptive implementation. Furthermore, no design method facilitates the extension of an already existing Web application with additional adaptation concerns.

Another important and yet not sufficiently addressed shortcoming is the current coarse-grained implementation model of the WWW. While it is well-suited for easy authoring and straightforward publication of documents, it is obviously not a sufficient implementation base for structured Web engineering approaches [Gaedke et al. 2000]. Though some of the aforementioned Web design methods provide a (semi-)automatic implementation generation, fine-granular semantic, navigational, and presentational design elements get lost during the implementation phase while being transformed into (X)HTML, cHTML, WML documents, etc. These document formats have significant disadvantages concerning their applicability for implementing and managing adaptive, dynamic Web information systems. The missing separation of content, layout, and structure prevents to uniformly create, update, and reuse content for different user preferences and platforms [Gellersen et al. 1997]. Furthermore, no mechanisms are provided to describe the adaptive behavior of reusable content pieces (Web code) in a generic way. This lack of structure and configurability in Web application code prevents the reuse of independent, configurable, and adaptable implementation artefacts both

within an application and for other applications and target systems. As a consequence, most Web site providers today use to create and manage Web code for different platforms and usage contexts separately.

At the same time, the efficient reuse of application code is a main task of traditional software technology, and has already been successfully addressed by component-based software engineering (CBSE [Szyperski 1998]). The advantages of component models are numerous: reusability, system-independence, configurability, flexibility, composability, etc. Traditional component models consider components as binary units of composition, mostly based on imperative programming languages. Still, in the recent years different approaches have been proposed to apply their principles to the document-centric nature of Web and multimedia applications [Gellersen et al. 1997, Aßmann 2005]. Meanwhile, there exist a number of structured, declarative, component-based document formats for different application areas, such as hypermedia presentations (HMDoc [Westbomke 2001]), Web applications (WebComposition [Gaedke et al. 2000]), eLearning applications (CHAMELEON [Wehner and Lorz 2001]) or even for Web-based three-dimensional user interfaces (CONTIGRA [Dachselt et al. 2002]). Still, even though all these approaches benefit from the reuse of declarative and configurable implementation artefacts in a component-like manner, none of them provides support for the aforementioned adaptation issues, such as device-independence, personalization, or localization. Moreover, there is a lack of solutions for the automatic generation of a component-based implementation based on high-level Web design specifications.

To fill this gap, the approach proposed in this dissertation focuses on the component-based development of adaptive Web applications. The vision is the intuitive composition of personalized, device-independent Web presentations from declarative, reusable, and adaptive "building blocks" (components), aided by a structured design and development process. In order to fulfill this vision, this thesis proposes a *concern-oriented component model* for adaptive Web applications[2,3]. It is based on the notion of declarative *document components* that encapsulate separate application aspects on different abstraction levels and can be automatically adjusted to varying user preferences and client platforms. For the development of component-based adaptive Web documents a structured *multi-stage, model-based authoring process* is presented. Distinguishing between different phases of design and component-based implementation, it allows component authors to systematically take into account various application and adaptation concerns. The intuitive creation and composition of document components is supported by a visual authoring tool. What more, it is illustrated how a component-based implementation can be automatically generated from a high-level design specification in a model-driven way, allowing to add automation to the overall process of design and implementation. The resulting development process supports different kinds of content, navigation, and presentation layer adaptation, and is demonstrated by a number of representative examples. Finally, it is shown how the lessons learned from authoring component-based adaptive Web applications can be generalized in order to add adaptation to existing (not component-based) Web applications.

As its main benefit, the proposed approach enables Web engineers to efficiently develop adaptive Web applications from reusable components, by systematically taking into account different application and adaptation concerns from design to implementation. According to

[2]The component model presented in this dissertation was developed within the scope of the AMACONT research project [@AMACONT] and is also often referred to as the AMACONT component model. The thesis presents the author's contributions to the model, based on requirements towards the efficient authoring of adaptive Web applications from reusable components.

[3]The term *concern-oriented* denotes the model's support for the clear separation of concerns involved in a Web application, each being dealt with on different component levels.

our best knowledge, it is the first development method for personalized, context-adaptive Web applications that combines the advantages of model-based Web design methodologies (e.g. high-level specification, thorough separation of design concerns, etc.) with the benefits of a component-based implementation techniques (such as reusability, configurability, or self-adaptation). To achieve this overall goal, a number of scientific contributions are provided, among them the design of a novel-component model for adaptive Web applications, its combination with a model-based Web design method, the provision of design-time support for presentation layer adaptation, as well as a means for easily adding adaptation to existing Web presentations.

1.2 Problems, Theses, and Research Goals

After outlining the research context and the main vision of this dissertation, this section recapitulates the concrete problems to be solved and derives research theses and goals resulting from them. First, the main shortcomings to be addressed are comprised:

Problems

- The development of adaptive Web applications requires significantly higher efforts compared to non-adaptive Web-based systems, there are lacking design guidelines and hardly any reuse concepts. The consideration and realization of different adaptation aspects (personalization, device independence, localisation, etc.) is a great challenge in designing, implementing, and maintaining Web sites.
- Current Web document formats are not suitable for developing personalized, device independent Web applications. The coarse-grained implementation model of the WWW prevents the efficient reuse of fine-granular implementation artefacts in a component-wise manner. Furthermore, there is a lack of mechanisms for describing the adaptive behavior of reusable content pieces (Web code) in a generic way.
- Due to their lacking support for adaptation, existing authoring tools for engineering dynamic Web applications are not suitable for personalized, device independent applications
- Existing authoring tools for adaptive hypermedia and Web-based systems are mostly restricted to the abstract level of conceptual modeling, not allowing for the visual creation of reusable adaptive implementation artefacts.
- Current design models for hypermedia and Web applications are only inadequately suitable for the specification of personalized ubiquitous Web applications. They do not provide sufficient support for specifying important concerns such as adaptation at presentation design or dynamic adaptation.
- There is a lack of solutions allowing for translating high-level design artefacts to an adaptive implementation layer in a model-driven way, thus adding automation to the overall development process of dynamic adaptive Web applications.
- Current solutions for engineering adaptive hypermedia and Web-based applications assume their development “from scratch”. There is no sufficient support (neither at design nor at implementation level) for easily adding adaptation to an existing Web application.

Theses

Based by these shortcomings and the situation of this dissertation in the research context, the following list comprises the main research theses behind the proposed approach. These theses also serve as the motivation of this work.

- Declarative component-based XML document formats combined with visual authoring tools are well applicable for engineering Web-based multimedia applications, among them personalized adaptive Web presentations. Still, existing component-based formats do not provide sufficient support for adaptive Web applications.
- The development process of component-based adaptive Web applications has to be based on a structured design and authoring process that is aided by appropriate authoring tools. These have to support a strict separation of concerns as well as varying aspects of adaptation in all different authoring steps.
- The combination of formalized, high-level design models with declarative, component-based document formats allows to automate the overall process of designing and implementing adaptive Web sites in a model-driven way.
- Adaptive Web Information Systems (AWIS) can be reduced to a series of data transformations. Furthermore, major parts of the adaptation-specific data transformations can be clearly separated from the rest of a Web application. Thus, the mechanisms for realizing adaptation in component-based Web applications can be generalized for adding adaptation to existing XML-based Web applications by using generic transcoding tools.
- Based on a declarative rule language, such generic adaptation components can be easily configured to implement various kinds of adaptation in existing Web Information Systems.

Research Goals

Triggered by the outlined shortcomings and theses, the vision of this dissertation is the efficient component-based development of adaptive Web applications in combination with a structured, model-based authoring process as well as visual authoring tools. In order to fulfill this vision and to elaborate the proposed theses, the following research goals have to be accomplished within the scope of this work:

- Design of a novel, XML-based, concern-oriented component model for engineering dynamic adaptive Web applications. Compact introduction and overview of the model's language constructs, its XML-based description language, as well as its selected benefits.
- Design of a structured authoring process for the component-based development adaptive Web sites. Adoption of the model-based Hera design method for engineering data-driven Web Information Systems from declarative document components. Explanation of the resulting Hera-AMACONT[4] methodology based on a running example.
- Design and prototypical implementation of a visual authoring tool called AMACONT-Builder for component-based adaptive Web applications. Compact introduction of its basic concepts and main modules from the author's point of view.

[4]Note that the work described in this dissertation was carried out as part of the AMACONT research project [@AMACONT].

- Design and prototypical implementation of a solution for the model-driven generation of component-based adaptive Web presentations. RDF(S)-based formalization of the presentation design phase of the Hera-AMACONT methodology. Realization of a model-driven transformation architecture for automatically translating high-level design artefacts to a component-based implementation supporting different aspects of static and dynamic adaptation.

- Design and implementation of mechanisms for separating adaptation-specific transformations from adaptive Web applications. Development of the GAC (Generic Adaptation Component), a generic transcoding component for making existing Web applications adaptable and adaptive.

- Specification of an RDF-based declarative GAC configuration language for the application-independent definition of adaptation and context data rules. Implementation of the GAC as well as demonstration of its main functionality by adding adaptation to an existing Web application.

1.3 Outline of the Dissertation

The rest of the dissertation is structured as follows.

Chapter 2 - Adaptive Hypermedia and Web-based Systems

Chapter 2 provides a short introduction to adaptive hypermedia and Web-based applications. Basic definitions are provided as well as main application areas, methods, and techniques of hypermedia adaptation are summarized. Furthermore, existing reference models for adaptive hypermedia and Web applications are described.

Chapter 3 - Development of Adaptive Web Applications: State of the Art

Chapter 3 deals with the development of adaptive Web applications and covers existing work on related Web engineering approaches. A main focus is on component-based and document-oriented approaches aiming at the implementation of multimedia and Web applications from declarative reusable implementation entities. Furthermore, model-based approaches supporting the structured design and development of hypermedia and Web-based systems are also discussed in detail. While examining related work, a special focus is on the question of how adaptation and personalization are supported. Limitations of existing approaches are observed as well as additional requirements for the development of adaptive Web-based systems are identified and discussed.

Chapter 4 - A Concern-Oriented Component Model for Adaptive Web Applications

Chapter 4 presents a concern-oriented component model for dynamic adaptive Web applications. The concept of declarative adaptable document components is introduced, and a corresponding XML-based component description language is presented. The document model enables to encapsulate adaptable content to document components on different levels of abstraction and provides support for describing both their adaptive behavior and adaptive

presentation layout. Furthermore, the concept of dynamic component templates is introduced, allowing to realize data-driven adaptive Web presentations. For the on-the-fly publishing of component-based adaptive Web applications a pipeline-based document generator is presented. Finally, significant model benefits are mentioned.

Chapter 5 - The Authoring Process and its Tool Support

Chapter 5 deals with the authoring process of component-based adaptive Web applications. Different application areas and possible process models are discussed, but the main focus is on the structured development of data-driven adaptive Web presentations. For this purpose the model-based Hera design method [Vdovjak et al. 2003] is adopted and extended to the context of component-based Web engineering, resulting in the so-called Hera-AMACONT methodology. Considering the different design steps identified by Hera-AMACONT as a guideline, it is shown how component authors can systematically create, configure, aggregate, and interlink document components to complex adaptive Web presentations.

As a flexible authoring tool for component developers the AMACONTBuilder is introduced. Based on an extensible set of graphical editor modules, it allows to create adaptive document components (and templates) on different abstraction levels. The tool is shown to facilitate different authoring scenarios, as it is independent of any one specific methodology. Furthermore, selected implementation and extensibility issues are also briefly presented.

While the AMACONTBuilder facilitates flexible component authoring (implementation) independent of a specific design method, in some cases it is desirable to add automation to the overall process of design and implementation. Therefore, Chapter 5.3 deals with the research question of how high-level design specifications can be automatically translated to a component-based adaptive Web presentation in a model-driven way. After identifying main automation requirements, an RDF(S)-based formalization of the presentation design phase of the Hera-AMACONT methodology is proposed. According to this formalization, high-level model specifications can be automatically mapped to a component-based implementation, thus exploiting its flexible presentation and adaptation capabilities. The resulting multi-stage hypermedia generation process is exemplified by a prototype application.

Chapter 6 - A Generic Transcoding Tool for Making Web Applications Adaptive

The combination of the component-based document format with an authoring process relying on a structured design method provides an efficient framework for developing adaptive Web-based systems. Still, the resulting engineering process assumes to build adaptive Web applications "from scratch", not providing support for developers aimed at adding adaptation to already existing applications. Therefore, Chapter 6 addresses the research question of how the lessons learned from developing component-based adaptive Web presentations can be generalized for extending a broader range of Web presentations with additional adaptation functionality. For this purpose a flexible transcoding tool called the Generic Adaptation Component (GAC) is introduced. Configured by an RDF-based rule language, it allows the addition of both static and dynamic adaptation to Web presentations. To prove the concept's feasibility, it is shown how GACs can be configured to add adaptation to an existing Web application.

Chapter 7 - Conclusion and Future Work

Chapter 7 provides a summary of the thesis and outlines its main contributions, achievements, but also its boundaries and limitations. Furthermore, possible extensions as well as targeted future work are discussed.

1.4 Writing Conventions

When reading this dissertation, following issues concerning the writing style should be taken into account:

- Some of the names of companies and/or products mentioned in this dissertation are registered trademarks. They are used without any warranty and are not explicitly marked in the text.
- Whenever a new (technical) term is mentioned or introduced in the thesis, its first occurrence is made stand out by using *italics*. Any further occurrences are not explicitly emphasized.
- Often-used abbreviations are listed and resolved in the Table of Abbreviations at the end of the dissertation.
- All cited literature references can be found in the Bibliography at the end of the thesis. References to Web sites of companies, products, projects, and prototype demonstrators are labeled by an additional @ prefix, e.g. [@AMACONT].
- At the end of the dissertation, there is an Index of the most important terms, concepts, and names.
- Listings and source code snippets are presented with numbered lines with syntax highlighting. Whenever there are deviations between the presented source code examples and their original form (e.g. for the sake of better readability or understandability), these differences are explicitly mentioned. A list of all source examples can be found at the beginning of the dissertation.
- The figures and tables presented in this dissertation were either created by the author, or their origin is explicitly mentioned (e.g. by an explicit remark in the text or the appropriate literature reference to their source). A list of all figures (List of Figures) and tables (List of Tables) can be found at the beginning of the dissertation.

Chapter 2

Adaptive Hypermedia and Web-based Systems

> *"The human mind (. . .) operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain."*[1]

While Chapter 1 introduced the main goals and the outline of this thesis, the aim of this chapter is to provide background knowledge on adaptive hypermedia and Web-based systems. First, Section 2.1 gives a short overview to hypermedia. It states basic definitions and presents the Dexter reference model. Then, Section 2.2 provides an introduction to the field of adaptive hypermedia and Web-based systems. Again, main definitions are presented, and the most important methods, techniques, and application areas of hypermedia adaptation are described. Finally, the AHAM reference model for adaptive hypermedia systems is summarized. The presented definitions, taxonomies, and reference models provide necessary background information that will be often referred to in the subsequent chapters.

2.1 Hypermedia and Web-based Systems

2.1.1 Definitions

The emergence of the notion of hypermedia can be traced back to 1945, when Vannevar Bush published his paper entitled "As We May Think" [Bush 1945]. In that paper he pointed out the shortcomings of linear information indexing systems and envisioned a device called *memex*. A *memex* is a tool that allows users to store textual information (books, notes, communications) and to add at any arbitrary location a pointer to another piece of text. This non-linear structuring of text would facilitate a more efficient management of information based on associations.

The vision of Bush inspired a number of researchers, among them Ted Nelson, who firstly used the term "hypertext". He was the creator of Xanadu [Nelson 1965], a system aimed at versioning documents and creating non-linear associations between pieces of text. In [Nelson 1987] he defines the terms hypertext and hypermedia as follows:

Definition 2.1 (Hypertext) *"I mean non-sequential writing - text that branches and allows choices to the reader,. . . this is a series of text chunks connected by links which offer the reader different pathways. . . "*

Definition 2.2 (Hypermedia) *"Hypermedia simply extends the notion of the text in hypertext by including visual information, sound animation and other forms of data. . . "*

[1] Vannevar Bush (1890-1974), "As We May Think", The Atlantic Monthly (1945) [Bush 1945]

According to these definitions, a hypertext (or hypermedia) system organizes its information as a set of nodes, i.e. units of information that contain textual or media content. Nodes are interconnected by pointers (links) and can thus be traversed in a non-linear order. As a more technical and also widely referenced definition focusing on the dynamic nature of hypertext from a database perspective, we mention the one from Shneiderman and Kearsley [Shneiderman and Kearsley 1989]:

Definition 2.3 (Hypertext) *"... a database that has active cross-references and allows the reader to "jump" to other parts of the database as desired".*

In the last decades a large number of hypertext and hypermedia systems have been introduced. Among the best known and most popular historical approaches we mention Notecard [Halasz 1987], Hyperties [Shneiderman 1987], Intermedia [Yankelovich et al. 1988], and HyperCard [Goodman 1987][2]. Still, it was the World Wide Web [Berners-Lee et al. 1992] that made the concept of hypertext known and available for the general public, thus becoming without doubt the most widespread hypermedia system. Before turning to the specifics of the World Wide Web as a hypermedia system in Section 2.1.3, the next section introduces Dexter, a reference model attempting to identify the most common features of hypermedia systems.

2.1.2 The Dexter Reference Model

The objectives of hypertext or hypermedia reference models are "to capture important abstractions found in current hypermedia applications, to describe their basic concepts, to provide a basis to compare the systems, and to develop a standard" [Koch 2001]. Recently, different reference models for hypermedia applications have been proposed. As one of the first and without doubt the most widely referenced model we will discuss in more detail the Dexter model [Halasz and Schwartz 1994], but point out that there are also other approaches, such as Trellis [Furuta and Stotts 1989], the Devise Hypermedia Model [Grønbæk and Trigg 1996], or the Dortmund Reference Model [Tochtermann and Dittrich 1996].

The Dexter reference model was published in 1990 as the result of two workshops of hypermedia experts. Their goal was to capture, both formally and informally, the important abstractions found in a wide range of existing and future hypertext systems. Dexter is formalized in the Z language [Spivey 1989], a specification language based on set theory. An overview of the Dexter model is shown in Figure 2.1. As depicted there, it identifies three main layers of a hypermedia application: the *Run-Time Layer*, the *Within-Component Layer*, and the *Storage Layer*. The connection between these layers is established by the two interface layers called *Presentation Specifications* and *Anchoring*.

The main focus of Dexter lies on the *Storage Layer*. It describes the basic node/link network structure of a hypertext system as a hierarchy of "components". A component is characterized by a unique identifier and is accessible through an accessor function. It can be either an atom, a link, or a composite entity made up from other components. Atomic components are basic content containers that are handled as primitives with regard to the Storage Layer. Their internal substructure is described in the Within-Component Layer. Links are typed entities that represent uni- or bidirectional relations between components and are specified by at least two "endpoint specifications", each of which refers to (parts

[2]For a more detailed introduction to the history of hypermedia systems the reader is referred to [Nielsen 1995, Casteleyn 2005].

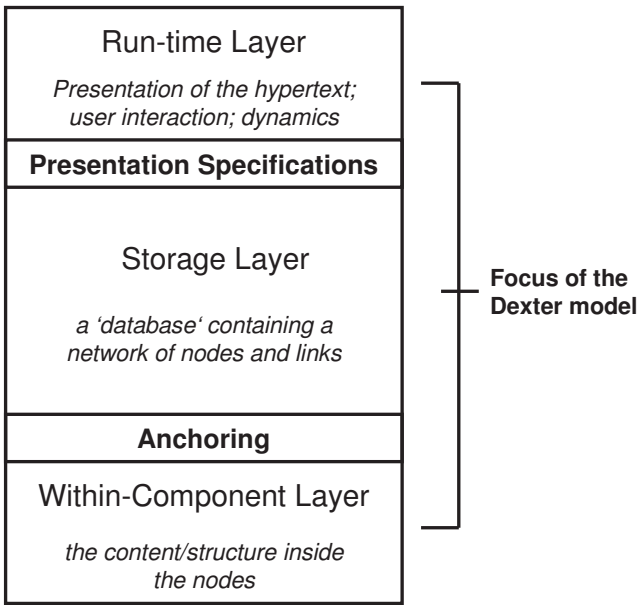

Figure 2.1: The Dexter reference model [Halasz and Schwartz 1994]

of) a component. Since they are also handled as components, links between links are also allowed. Finally, composite components build a hierarchy of components based on aggregation relationships between them.

The *Within-Component Layer* describes the concrete content and structure (e.g. media elements, page fragments, etc.) *within* the components described in the Storage Layer and is not further specified by Dexter. Its interface to the Storage Layer is constituted by the interface layer *Anchoring* that allows to address (refer to) locations within the content of an atomic component.

While the Storage Layer and the Within-Component layer handle hypertext as an essentially passive data structure, the *Run-time Layer* is concerned with the presentation of components to the user, allowing him to access, view, and manipulate the overall network structure. The fundamental concept of the Run-time Layer is the *instantiation* of a component which means its presentation to a user and can be thought of as a kind of run-time cache for the component. At a particular moment, the user of the hypertext can be viewing and manipulating a number of component instantiations. His interactions with the hypertext system are managed by a *session* entity aimed at keeping track of his current component instantiations. The Run-time Layer provides a number of abstract functions, e.g. for starting or ending sessions, manipulating component instantiations, following links, etc.

Again, the interface between the Run-time Layer and the Storage Layer is accomplished by an interface layer called *Presentation Specifications*. Presentation specifications are a mechanism by which information about how a component/network has to be presented to the user can be encoded into the hypertext network at the storage layer. That is to say, the way in which a component is presented to the user can be a function not only of the specific hypertext tool that is doing the presentation (i.e. the specific run-time layer), but also a property of the component itself. For more detailed information on the constructs and the formalization of the Dexter reference model the reader is referred to [Halasz and Schwartz 1994].

Even though Dexter covers the basic functionality provided by hypermedia applications,

there are some more specific concerns (such as multimedia synchronization or adaptation) which are not explicitly addressed by it. Therefore, a number of more specialized reference models have been proposed, recently. For example, the Amsterdam Reference Model adds the notion of time and synchronization to the Dexter model, thus allowing to describe hypermedia applications using multimedia elements, too [Hardman et al. 1994]. Similarly, there exist also extensions aimed at explicitly including adaptation, such as AHAM [De Bra et al. 1999] or the Munich Reference Model [Koch and Wirsing 2002]. Since AHAM is the most widely used reference model in the field of adaptive hypermedia and Web-based systems, it will be described in more detail in Section 2.2.5.

2.1.3 The World Wide Web as a Hypermedia System

In the recent years, the World Wide Web (WWW) has become without doubt the best-known and most widely used hypermedia system. Assuming that its basic concepts are well known to the reader, it suffices to mention that it provides significant hypermedia functionality, such as interconnectivity, non-linearity, and the possibility to integrate different (textual and non-textual) media elements. Nevertheless, the Web does not support all the functionality that a full Dexter compliant hypermedia system could offer. As the most important restrictions the following can be mentioned [Casteleyn 2005].

- Hyperlinks in the World Wide Web are unidirectional and untyped, i.e. they do not explicitly carry a semantic meaning. There is no support for links with more than one endpoint, nor for hyperlinks between hyperlinks.
- Hyperlinks in the WWW are not considered stand-alone, "full-fledged" objects of the hyperspace. Instead of being stored as separate components, they are embedded in their source documents, i.e. merged with the actual content. Consequently, it is not possible to alter or manipulate hyperlink structures independent of the underlying content nodes.
- The nodes of the Web hyperspace are coarse-grained file-based resources primarily represented as HTML documents. They describe all relevant aspects of a hypermedia presentation (content, navigation, presentation) intertwined in one document, there is lacking support for the separation of concerns and the effective reuse of fine-grained content fragments.
- There is no inherent support for customizing (parts of) Web presentations, it is difficult to adjust the hyperspace to the characteristics and preferences of specific users.
- The WWW was originally invented as a simple presentation medium, implying that users cannot modify the hyperspace and/or add new nodes or links to it.

As a matter of course, these limitations mean restrictions compared to the functionality that a full hypermedia system can offer. Still, this relatively simple nature of the Web is also without doubt one of the main reasons why it found its way to the general public so quickly. This first generation of Web-based systems presented information in terms of carefully authored hypermedia documents. Typically, it involved the manual creation of a static set of HTML pages in order to convey information to the users. However, the Web's growing popularity has soon lead to the need for interactive Web-based systems that would publish up-to-date content. As a consequence, so-called Web Information Systems (WIS) have emerged.

Definition 2.4 (Web Information System) *A Web Information System (WIS) is an information system that uses the Web to present data to its users [Isakowitz et al. 1998].*

In contrast to Web presentations built of static Web pages (also often referred to as the "surface Web" [Houben 2004]), a WIS is typically tightly integrated with dynamic data sources (the "deep Web" [Ghanem and Aref 2004]). It *generates* Web presentations based on the data retrieved from these sources on-the-fly. Typical application areas of WIS are online news papers, e-galleries, electronic shops, etc.

Web Information Systems are also different from traditional information systems. They require new approaches to design and development [Fraternali 1999], have the potential to reach a much wider audience, and are usually a result of grass-roots efforts [Isakowitz et al. 1998]. The structured development process of (adaptive) WIS will be subject to Chapter 3 and Chapter 5 of this thesis.

2.2 Adaptive Hypermedia and Web-based Systems

According to Vannevar Bush's vision, hypermedia indeed changed "the way we think", access, and process information. Especially the success of the Web facilitated the world-wide publication of huge amounts of interlinked information, allowing a heterogeneous group of users to traverse it in a non-linear manner. However, this rapid growth of the "information universe" and the heterogeneity of its audience also showed a main shortcoming of traditional hypermedia systems: the fact that they provide the same page content and the same set of links to all users. It became obvious that this "one size fits all" approach would not be sufficient, requiring hypermedia systems to adjust (*adapt*) themselves to the user to better facilitate his navigation through the information space. This requirement is addressed by so-called *adaptive hypermedia and Web-based systems* [Brusilovsky 1996, De Bra et al. 2004].

2.2.1 Definitions

Brusilovsky [Brusilovsky 1996, Brusilovsky 2001] defines adaptive hypermedia systems (AHS) as follows:

Definition 2.5 (Adaptive Hypermedia Systems) *"By adaptive hypermedia systems we mean all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user. In other words, the system should satisfy three criteria: it should be a hypertext or hypermedia system, it should have a user model, and it should be able to adapt the hypermedia using this model."*

When classifying AHS, a further distinction is made between *adaptivity* (or static adaptation) and *adaptability* (also called dynamic adaptation). Systems that allow the user to change certain system parameters and adapt their behavior accordingly are called *adaptable*[3]. On the other hand, systems that adapt to the user automatically based on the system's assumptions about user needs are called *adaptive*. In the rest of this thesis the following definitions of adaptability and adaptivity stated by Frasincar et al. [Frasincar et al. 2002] will be used.

Definition 2.6 (Adaptability) *"Adaptability (or* static adaptation*) means that the generation process is based on available information that describes the situation in which the user will use the generated presentation [Frasincar et al. 2002]."*

[3]Some authors refer to adaptable systems as *configurable* or *customisable* systems [Kobsa et al. 2001].

Definition 2.7 (Adaptivity) *"Adaptivity (or* dynamic adaptation*) is the kind of adaptation included in the generated adaptive hypermedia presentation, i.e. the generated hypermedia presentation changes while being browsed [Frasincar et al. 2002]."*

2.2.2 Adapting to what?

As stated in Definition 2.5, an adaptive hypermedia system adapts its various visible aspects to a user model. Depending on the given application scenario, this model maintains information on varying features describing the actual user. While there exist different definitions of the term user model, we mention one of the first ones that was published by Timothy W. Finin [Finin 1989].

Definition 2.8 (User Model) *"A user model is that knowledge about the user, either explicitly or implicitly encoded, which is used by the system to improve the interaction."*

Based on the given application (adaptation) scenario, a user model can maintain information on different user features. In his first survey on adaptive hypermedia systems Brusilovsky identifies the following five features [Brusilovsky 1996]:

- **Knowledge:** The *user's knowledge* about the concepts presented in a hypermedia system is one of the main adaptation features considered by current adaptive hypermedia and Web-based applications. It is most often represented in form of an *overlay model* which sees the individual user's knowledge of the subject as an "overlay" of the domain knowledge. Another popular representation form are *stereotype models* aimed at distinguishing between different user stereotypes (e.g. novice, beginner or expert). Typically, the information maintained on users' knowledge is continually updated during their interaction with an AHS.

- **Goals:** User *goals* (also often referred to as user tasks) are features related with the context of the user's work in an AHS [Brusilovsky 1996]. Based on the given application area one can distinguish between different user goals, such as learning goals (typical for Adaptive Educational Hypermedia Systems), search goals (e.g. in Adaptive IR Systems), etc. Similar to user knowledge, user goals can also dynamically change during a browser session.

- **Background and Experience:** The *user's background* relates to the user's previous experience outside the subject of the hypermedia system. On the other hand, the *user's experience* denotes his/her familiarity with the hyperspace or the given hypermedia system.

- **Preferences** are a very important (and wide-ranging) user feature considered by adaptive hypermedia and Web-based systems. As an example, when interacting with an AHS, users might prefer some nodes and links over others and some parts of a page over others [Brusilovsky 1996]. Further preferences may concern the media types involved in a hypermedia presentation (i.e. a user might favor multimedia elements [Jörding 1999] to pure textual content) but also its layout/design (such as colors, font sizes, buttons, etc. [Fiala et al. 2004a]). Unlike other user features, user preferences cannot be deduced by the adaptive hypermedia system, i.e. the user has to inform the system directly or indirectly (e.g. by a simple feedback) about such preferences.

While these features are exclusively centered around characteristics of the user, the evolution of Web-based systems (and especially the emergence of heterogeneous mobile Web client appliances) made it necessary to adapt hypermedia applications to different features (e.g. devices characteristics or location), as well. Kobsa et al. [Kobsa et al. 2001] (but also the updated survey of Brusilovsky [Brusilovsky 2001]) suggest to distinguish between *user data*, *usage data*, and *environment data*.

User Data denotes information about personal characteristics of the user. While it mainly corresponds to the user features already mentioned above, Kobsa et al. [Kobsa et al. 2001] also mention further features such as *demographic data* (name, address, sex, education, etc.) or *user interests*.

Usage Data relates to information describing the user's interaction with the system that may be directly observed and recorded, or acquired by analyzing observable data. As possible observable interactions Kobsa et al. [Kobsa et al. 2001] mention *selective actions* (e.g. following a given link or selecting an option from a select list), *temporal viewing behavior* (i.e. the time a user spends on a Web page), *ratings*, as well as *purchases and purchase-related actions*. Based on these interactions the system can derive so-called *usage regularities*, such as typical action sequences or usage frequency.

Environment Data is related to the environment of the user and describes his *software environment* (e.g. browser version, available plug-ins, and client-side scripting technologies), *hardware environment* (e.g. device type, display size, supported interaction techniques, bandwidth, processing speed), or even *locale* (information on the physical location of the user). According to the targeted application scenario, the granularity of location information used for adaptation can vary from very fine (e.g. distinguishing between streets or even rooms in a handheld tourist guide) to rather coarse (e.g. providing different presentations of a company in different countries) [De Troyer and Casteleyn 2004]

The adjustment of software systems (among them of Web applications) to environment data is also the main focus of the recently emerged computing paradigms *ubiquitous computing* and *context-aware computing*. Their goal is to make users' interactions with applications easier by taking into account his *context*, i.e. the actual situation in which he interacts with applications. The term *context* is typically used in a broad sense and can refer to various features such as the user's social context, emotional state, device, location, surroundings, the appropriate time of day, etc. While there exist different definitions of the terms context and context awareness ([Schilit et al. 1994, Schmidt et al. 1999, Abowd et al. 1999]), we mention one of the most well-known and most often referenced definitions from Anind K. Dey [Dey 2001].

Definition 2.9 (Context) *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [Dey 2001]."*

Definition 2.10 (Context-Awareness) *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [Dey 2001]."*

Note that the generality of these definitions allows to handle the formerly mentioned user features (both user data and usage data) also as parts of the user's context under which he

interacts with an adaptive hypermedia or Web-based application. Therefore, in the rest of this thesis the term context will be used to characterize both the user and all other information describing his situation.

2.2.3 What can be adapted?

Besides the elaboration of user/context features being relevant for adaptation ("adapting to what?"), another important question is "what to adapt", i.e. which parts of an AHS can differ for different users/contexts. Brusilovsky distinguishes in his surveys [Brusilovsky 1996, Brusilovsky 2001] between *content-level* and *link-level* adaptation and denotes these adaptation classes as *adaptive presentation* and *adaptive navigation*, respectively. Whereas adaptive presentation aims at adapting the content presented on a hypermedia page (or node), adaptive navigation adjusts the interlinking of those nodes. For both adaptation classes he identifies the following techniques:

Techniques for Adaptive Presentation

- **Adaptive Text Presentation** means that the textual representation of the content offered by a hypermedia system is adjusted. The corresponding adaptation techniques comprise conditional text, fragment or page variants, sorting of page fragments, dimming of fragments [Hothi and Hall 1998], frame-based techniques [Hohl et al. 1996], adaptive natural language generation (NLG [Dale et al. 1998]), and stretchtext. The latter is a special kind of "collapsible" text fragment that offers information in varying depths of detail and can be collapsed/uncollapsed according to the knowledge or preferences of the current user, respectively [Boyle and Encarnacion 1994].

- **Adaptive Multimedia Presentation** aims at adjusting the non-textual media elements of a hypermedia presentation. It means either the adaptation of the actual media (such as resizing an image, reducing its color depth, transcoding a video, etc.) or the selection from different media representations (e.g. showing a picture instead of a video). The latter adaptation technique is also often referred to as *adaptation of modality*. Furthermore, note that some of the techniques mentioned above for adaptive text presentation can be also effectively used for adapting multimedia content (e.g. conditional inclusion of media elements, media elements with variants, sorting of non-textual fragments).

Techniques for Adaptive Navigation

- **Direct Guidance** is one of the simplest technologies of adaptive navigation support. It means that the adaptive hypermedia system determines the "next best" link for the user so that he cannot decide to follow another path. A main disadvantage of this technique is its restrictive nature, not supporting for users who would not like to follow the system's suggestion.

- **Link Sorting** means the ordering of a set of anchors, so that links are presented in decreasing order of their relevance to the user. The disadvantage of adaptive ordering is that each time the user enters the same page, the ordering of anchors may be different [Koch et al. 2001].

- **Link Hiding** means that a link is available, yet presented as normal text. The goal of this technique is to protect users from the complexity of the unrestricted hyperspace by "hiding" links to irrelevant pages.
- **Link Disabling** removes the functionality of a link but leaves its visual appearance nearly untouched. That is to say, the link anchor still looks like a link, but the user cannot follow it. The disadvantage of this technique is that unexperienced users might be confused by the "different" behavior of normal and disabled links.
- **Link Removal** means that a link is completely removed, i.e. both the link anchor and the associated link functionality are filtered out. The underlying motivation is to reduce the hypermedia space by removing links pointing to information that is not relevant for the actual user.
- **Link Annotation** aims at the visual augmentation of links in accordance to their importance for the user. Annotations can be realized in both textual form but also visually, e.g. by using different icons, colors, or font sizes. As an example, Dolog et al. use the so-called *traffic light metaphor* to express whether a link target is too difficult (red), recommended (green), or has been already seen (gray) by the user [Dolog et al. 2003].
- **Link Generation** means that the hypermedia presentation is automatically enriched by links being not present at the time of hypertext authoring. This technique is especially very popular in adaptive information retrieval systems (see Section 2.2.4.2), but also in adaptive recommender systems used in e-commerce applications [Burke 2002].
- **Map Adaptation** comprises various ways of adapting local and global hypermedia maps (such as sitemaps of large Web applications), especially by adapting their presentation or granularity.

Even though Brusilovsky uses the term *adaptive presentation*, note that the corresponding techniques mentioned by him primarily concern adaptation at content-level. Therefore, Paterno and Mancini identify in [Paterno and Mancini 1999] additional adaptation techniques that explicitly focus on the presentation-level of an AHS, i.e. adaptations of the layout (such as colours, font types, font sizes) that do not effect the underlying content. As corresponding adaptation techniques they mention *layout variants* and *styleguiding*. While the former one concerns the arrangement of a hypermedia page's content elements according to a given layout schema, the latter means the usage of different style guides that are used alternately for specific layout variants. A more thorough elaboration of presentation layer adaptation will be given in Chapter 5.

For a detailed discussion and comparison of adaptation methods and their corresponding adaptation techniques the reader is referred to [Brusilovsky 1996, Paterno and Mancini 1999, Brusilovsky 2001].

2.2.4 Application areas of AHS

In his surveys [Brusilovsky 1996, Brusilovsky 2001], Brusilovsky classifies adaptive hypermedia systems according to their application areas. He identifies three main fields: *adaptive educational hypermedia systems*, *adaptive information retrieval hypermedia systems*, and *adaptive online information systems*. Taking Brusilovsky's classification as a basis, this section summarizes the most important characteristics and "subareas" of these application fields. Thereby, the main focus is on the broad range of adaptive online information systems, a "subset" of which will be subject to further investigation in the rest of the thesis.

2.2.4.1 Adaptive Educational Hypermedia Systems

The first and most popular application area for adaptive hypermedia research was educational hypermedia. Adaptive Educational Hypermedia Systems (AEHS) are e-learning systems that use adaptive hypermedia techniques to adjust online courses to users' varying and changing goals or knowledge. They examine students' learning efforts and adapt the courses or exercises presented for them according to their improvements.

The hyperspace of an AEHS is typically relatively small and well structured by a designer. Most often it represents a particular course or section of learning material on a given subject [Brusilovsky 1996] in form of concepts (knowledge units) and their relationships. The goal of the student is usually to learn all this material or a reasonable part of it. To achieve this goal, the application of adaptive hypermedia techniques can help him to optimally find his way through the available material (link-level adaptation) and to present the course material adjusted to his current knowledge (content-level adaptation). The user model of an AEHS is typically an overlay model, i.e. it contains user-specific information related to the concepts (knowledge units) of the application domain. AEHS examine students' learning efforts either in an explicit way (i.e. based on exercises or questionnaires) or implicitly by observing their navigation through the course material. Based on this information they update students' user models (also often referred to as learner models) by means of specific learning algorithms.

As prominent examples of AEHS we mention Interbook [Brusilovsky et al. 1996], the AHA! (Adaptive Hypermedia for All!) platform [@AHA, De Bra and Ruiter 2001], and the KBS Hyperbook System [Henze and Nejdl 2001]. While the first two systems use deterministic rule-based approaches for modeling students' knowledge, KBS utilizes a stochastic approach based on Bayesian networks [Burke 2002]. For a thorough review of the history and the most important representatives of Adaptive Educational Hypermedia Systems the interested reader is referred to [Brusilovsky 2004].

2.2.4.2 Adaptive Information Retrieval Hypermedia Systems

Adaptive Information Retrieval Hypermedia Systems are Information Retrieval (IR) systems that make use of the hypertext paradigm. They "combine traditional information retrieval techniques with a hypertext-like access from the index terms to documents and provide the possibility of browsing the hyperspace of documents using similarity links between documents [Brusilovsky 1996]".

Triggered by the rapid development of the WWW, the most challenging problem of current IR hypermedia systems is to support the user's information retrieval tasks in the unrestricted Web hyperspace. Since this hyperspace cannot be structured "by hand", the similarity links between documents are not provided (i.e. prepared) by a designer, rather calculated by the system, e.g. using similarity measurements. Adaptive IR hypermedia systems take into account users' search requests, relevance feedbacks, and usually build a long-term model of their goals and interests. They mainly utilize navigation adaptation techniques, especially link generation, link annotation, and link sorting. Brusilovsky distinguishes between two groups of adaptive information retrieval hypermedia systems: search-oriented systems and browsing-oriented systems [Brusilovsky 2001]. Whereas the former ones (e.g. CASPER [Smyth et al. 2002] or the system of Marinilli et al. [Marinilli et al. 1999]) aim at creating a list of links to documents that satisfy the user's current information request, the latter (such as [Fu et al. 2000]) support their users more implicitly in the process of search-driven browsing.

2.2.4.3 Adaptive Online Information Systems

Adaptive Online Information Systems are data-intensive hypermedia information systems providing reference access to highly structured and typically volatile information. Similar to the other application areas of adaptive hypermedia, their development was significantly boosted by the success of the World Wide Web. Web-based adaptive online information systems are often referred to as Adaptive Web-based Information Systems (AWIS)[4], i.e. systems extending the functionality of Web Information Systems (WIS) with different aspects of adaptation.

AWIS provide different kinds of content, navigation, and presentation adaptation to various features of both the user (knowledge, interests, preferences) and his usage context (device, environment, or location) [Houben 2004]. Depending on the targeted application domain, their hyperspace can range from reasonably small to very large. Most typically, AWIS provide not only hypermedia access to their information base, they also allow users to manipulate this data based on some application logic (e.g. in electronic commerce applications). The following (not exhaustive) list comprises the most important application areas of adaptive online information systems.

Electronic encyclopedias are information systems that present highly-structured information on a well-defined subject in form of a data-driven hypermedia application. They observe users' knowledge about different objects described in the encyclopedia and provide adaptive comparisons to other objects. Similarly, they can trace the user's browsing, deduce his or her interest, and offer a lists of most relevant articles. As typical examples PEBA-II [Milosavljevic 1997] and ILEX [Oberlander et al. 1998] can be mentioned, both providing adaptive comparative explanations of the stored concepts based on the user's navigation by means of natural language generation. Note, however, that besides "real" electronic encyclopedias, this application category also comprises a number of other adaptive electronic online information systems, among them online newspapers [Ardissono et al. 1999], digital libraries [Hicks and Tochtermann 2001], Web-based movie or music databases, electronic yellow pages, timetable systems, etc.

Information kiosks (such as AVANTI [Fink et al. 1998]) are information systems installed at public places, e.g. at fairs, exhibitions, or showrooms. Typically, they need to support "walk up and use" by first time users or infrequent users [Kobsa et al. 2001]. Besides user and usage modeling, these systems put a main focus on adaptations to characteristics of the environment or locale, such as noise or reduced privacy due to the proximity of other people. For example, the brightness or loudness of a multimedia presentation can be adjusted to the current time of day or noise level, respectively.

Virtual museums or tourist guides provide adaptive guided tours to support the user's exploration of a virtual or real museum or a touristic place with context-adapted narration. Their main goal is to adjust the presentation of every visited object to the user's knowledge, interests, and individual navigation path [Aroyo et al. 2005]. With the emergence of Location-based Services (LBS [Küpper 2005]), a new generation of such systems, so-called handheld guides have appeared, recently. As prominent representatives HyperAudio [Petrelli et al. 1999], Guide [Cheverst et al. 2000], or Lol@ [Pospischil et al. 2002] can be mentioned. By determining the user's location and behavior, such handheld guides can better support his navigation both in the physical space and the virtual hyperspace.

[4] see the definition of the term Web Information System provided in Section 2.1.3

E-commerce systems use methods and techniques of hypermedia adaptation to optimally support the shopping activities of their users. They cover a broad range of applications, among them online shops, auction systems, virtual marketplaces, etc. While a hyperspace of information items still constitutes a major part of these systems, the browsing of this hyperspace is not a major activity, rather a byproduct of the major activity (i.e. shopping) [Brusilovsky 2001]. The application of adaptation methods and techniques in an E-commerce application addresses the presentation of specific products (e.g. based on users background knowledge, media preferences, and client devices [Jörding 1999, Ardissono et al. 2002]), the recommendation of (lists of) products that best suit the user's interests [Linden et al. 2003], or (in the case of location-based services) even the recommendation of a list of shops or dealers closest to their current location [Tsalgatidou and Veijalainen 2000].

Performance support systems can be seen as a combination of domain expert systems and domain information systems [Brusilovsky 2001]. Their main goal is to help users solve problems in a typically very specific field, such as technical repair or medical treatment [Francisco-Revilla and Shipman 2000]. The adaptation provided by these systems is based on the user's actual work context and goals. Based on these features appropriate support information (e.g. background knowledge, technical documentations, reviews of related problems, and solutions, etc.) can be provided in a suitable way. As a prominent example ADAPTS [Brusilovsky and Cooper 2002] can be mentioned. It is an electronic performance support system (EPSS) for maintenance technicians that integrates an adaptive diagnostics engine with adaptive access to technical information.

2.2.5 The AHAM Reference Model

As discussed in this section, adaptive hypermedia systems can be considered as a specialized class of traditional hypermedia systems that are additionally characterized by the usage of a user (or context) model and the possibility to adjust their content, navigation, and presentation to it. As an attempt to describe, characterize, and compare them in a formal way, De Bra et al. introduced the AHAM reference model [De Bra et al. 1999, Wu 2001]. It is an extension of the already introduced Dexter model (see Section 2.1.2) that further specifies its Storage Layer by dividing it into three sub models: the *domain model*, the *user model*, and the *teaching model* (see Figure 2.2).

The *domain model* (DM) describes how the information presented by an adaptive hypermedia system is structured and linked together. It is composed of atomic concept components, composite concept components, and concept relationship components. Atomic concept components are basic information fragments that are considered as primitives and are not adaptable. Composite concept components are hierarchical aggregates that may contain both a number of atomic or composite concept subcomponents. Finally, concept relationship components constitute a relation between at least two (atomic or composite) concept components. These can be both link components used for hypertext navigation, as well as other types of rather conceptual relationships that play an important role for adaptation. As an example, the concept relationship *prerequisite* can mean that the source concept A represents prerequisite knowledge for the destination concept B, i.e. the user should already have visited (and thus learned) A in order to get access to B [De Bra et al. 1999]. However, AHAM makes no restrictions to the possible link types and their interpretation.

The *user model* (UM) contains information which the system records about the user. It associates a number of user model attributes to each concept component of the domain

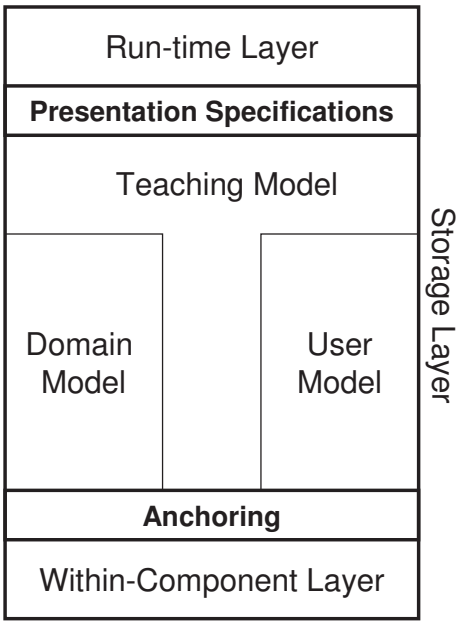

Figure 2.2: The AHAM reference model [De Bra et al. 1999]

model, i.e. for each user a table containing these associations and their concrete values is maintained. That is to say, the user model of AHAM is an *overlay model* represented as a view on the domain model. As a matter of course, the names, types, and roles of user model attributes are not prescribed by AHAM and are hence specific to each given adaptive hypermedia application. Unfortunately, AHAM does not support the definition of user model (or context model) parameters that are independent of the underlying domain model (such as the user's identification, age, background, location, etc.).

The *teaching model* (TM) defines how the domain model and the user model are combined for performing adaptation. It contains a set of *pedagogical rules* that are parametrized by user model parameters and describe adaptation operations on the domain model, such as including or excluding page fragment components, hiding or annotating links, etc. For instance, pedagogical rules might exclude specific concepts if the current user has not sufficient knowledge on their prerequisites. Furthermore, they can also update user model values according to the user's navigation through the concept structure of the domain model. However, the syntax of pedagogical rules is not specified by AHAM.

Finally, AHAM defines an adaptive hypermedia system (AHS) as a 4-tuple consisting of the DM, UM, TM, as well as an *adaptive engine* (AE)[5]. The task of the AE is to adapt the node/link structure of the hypermedia system based on the three models and to keep the user model up-to-date, respectively. Thus, for each individual user a different navigation structure is generated and sent to the Run-Time Layer.

As an implementation of the AHAM reference model De Bra et al. introduced the AHA! system [De Bra and Ruiter 2001, De Bra et al. 2002]. Apart from some simplifications, it realizes all basic models and notions of AHAM. Furthermore, it also offers a number of graphical authoring tools for the visual definition of concepts, concept relationships, and adaptation rules.

[5]We note that in [Aroyo et al. 2003] Aroyo et al. add the notion of a Retrieval Model (RM) to AHAM, thus allowing to describe Adaptive Information Retrieval Hypermedia Systems.

Though claimed to be a universal reference model aimed at capturing common characteristics of arbitrary adaptive hypermedia systems, we note that AHAM (and AHA!) are primarily suitable for describing (and implementing) Adaptive Educational Hypermedia Systems. They put a main focus on user-specific adaptations of a hypermedia system's conceptual and navigational structure, but provide no support for a broader range of adaptations (e.g. of presentation or modality) and/or context models (such as device parameters, document formats or environment characteristics that are not necessarily associated with concepts of the domain model). Furthermore, its central focus is on static hypermedia presentations, not explicitly addressing the particular characteristics of adaptive data-driven information systems.

As a reference model, AHAM provides a common vocabulary for the analysis of adaptive hypermedia and Web-based systems, yet it does not deal with their design and implementation process. The state of the art on the field of the development of adaptive Web applications is subject to the investigations of the following chapter.

2.3 Summary

The aim of this chapter was to provide basic information on adaptive hypermedia and Web-based systems. Main definitions and taxonomies were stated as well as the most important characteristics, application areas, and reference models of hypermedia adaptation were summarized. The presented information provides necessary background knowledge for the reader and will be often referred to in the rest of the thesis.

As discussed in Section 2.2.4, adaptive hypermedia and Web-based systems cover a broad range of application areas, each having its specific characteristics and requirements. As a matter of course, the approach presented in this dissertation focuses only on a specific subset of these fields, namely on the development process of online adaptive Web information systems aimed at presenting structured, dynamic multimedia data in a device-independent, personalized way. Furthermore, while a main focus will be on designing and implementing content, navigation, and presentation adaptation, other kinds of adaptation (e.g. the adjustment of a Web application's interaction behavior or business logic) will be only marginally discussed.

After introducing the reader to the foundations of adaptive hypermedia systems in this chapter, the next chapter will summarize and discuss related work on engineering adaptive Web information systems. Then, the rest of the dissertation will propose an approach aimed at the structured design and component-based development of personalized, ubiquitous Web applications.

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-

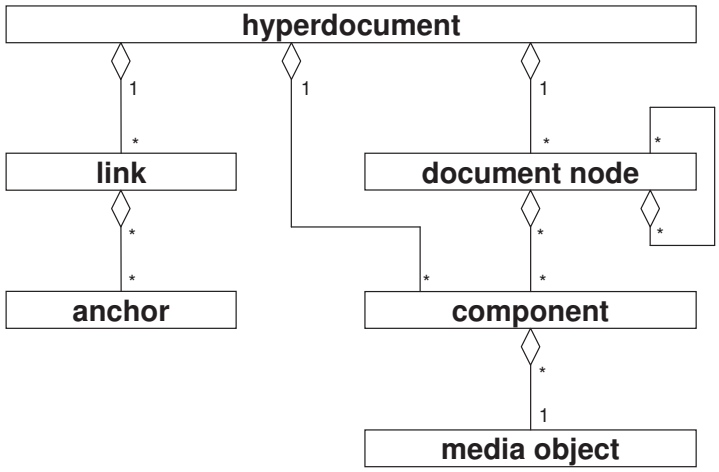

Figure 3.1: Basic structure of an HMDoc hyperdocument [Westbomke and Dittrich 2002]

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".

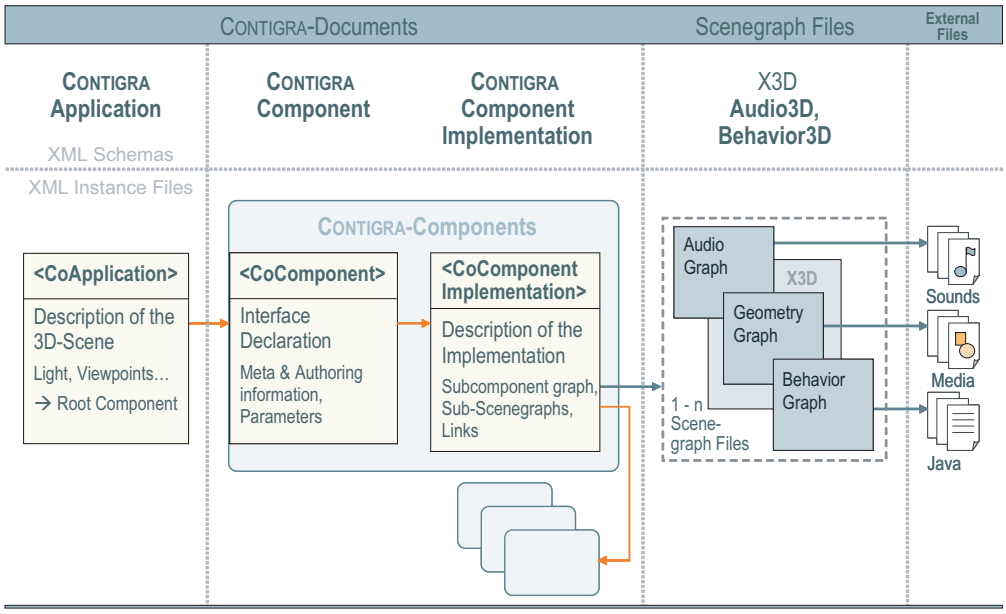

Figure 3.2: Overview of the CONTIGRA markup languages [Dachselt 2004]

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

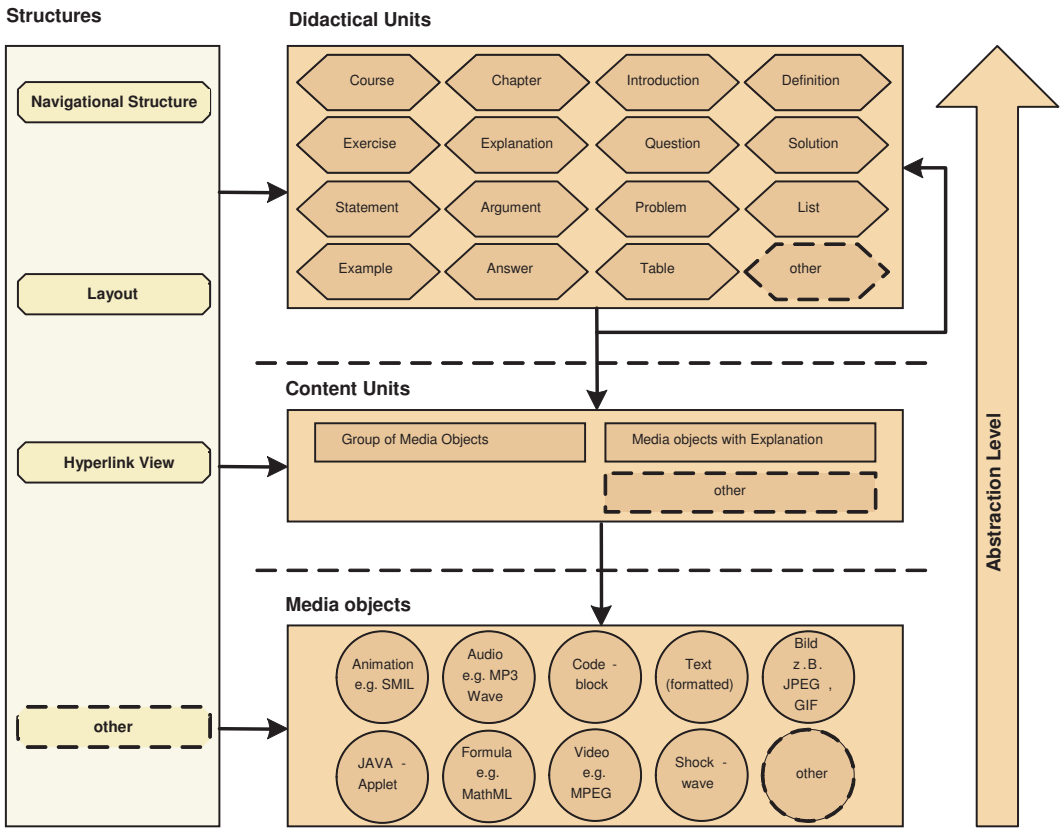

Figure 3.3: Overview of the TeachML document model [@CHAMELEON]

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].

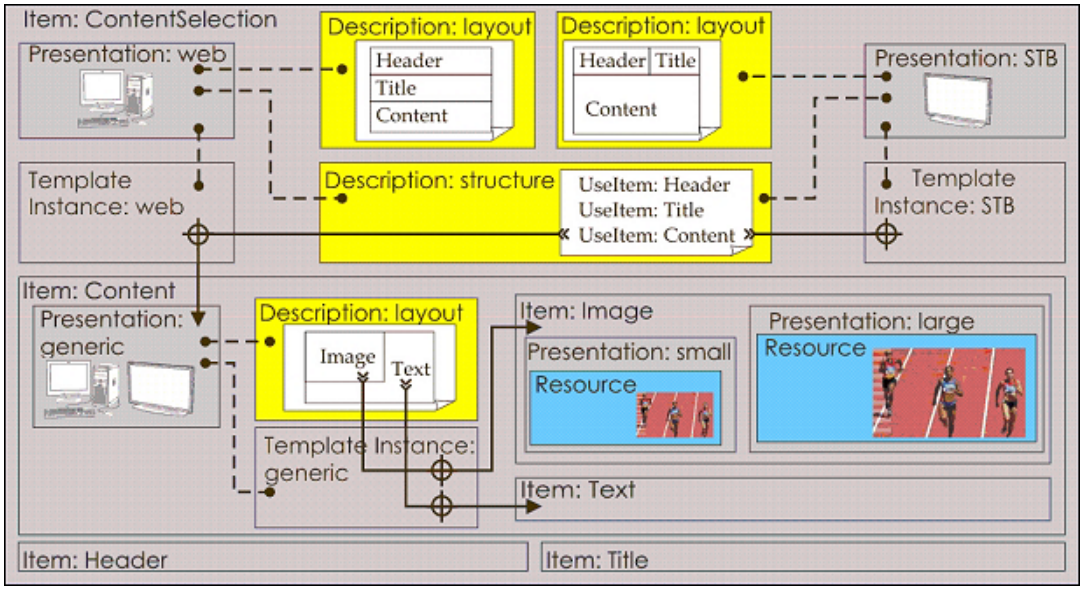

Figure 3.4: Schematic outline of an XiMPF document[5] [Hendrickx et al. 2005]

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).

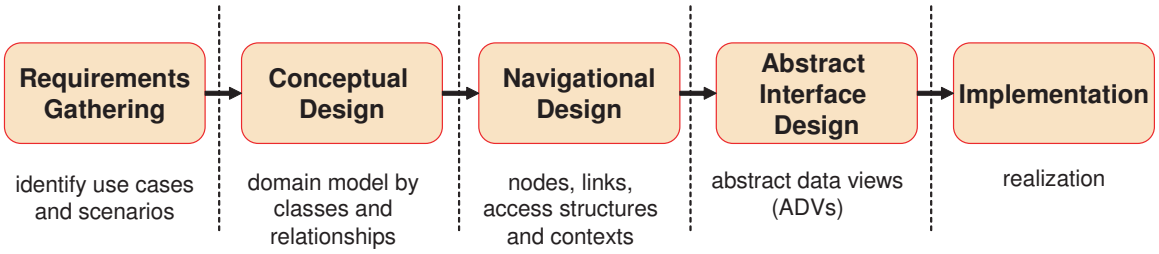

Figure 3.5: OOHDM/SHDM overview

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.

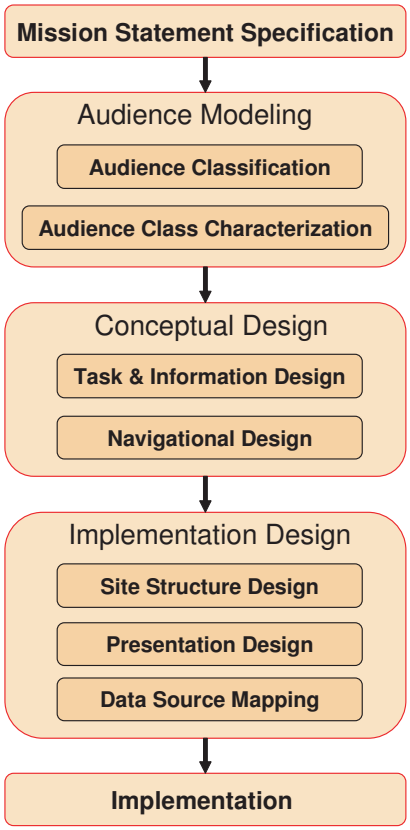

Figure 3.6: WSDM overview

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].

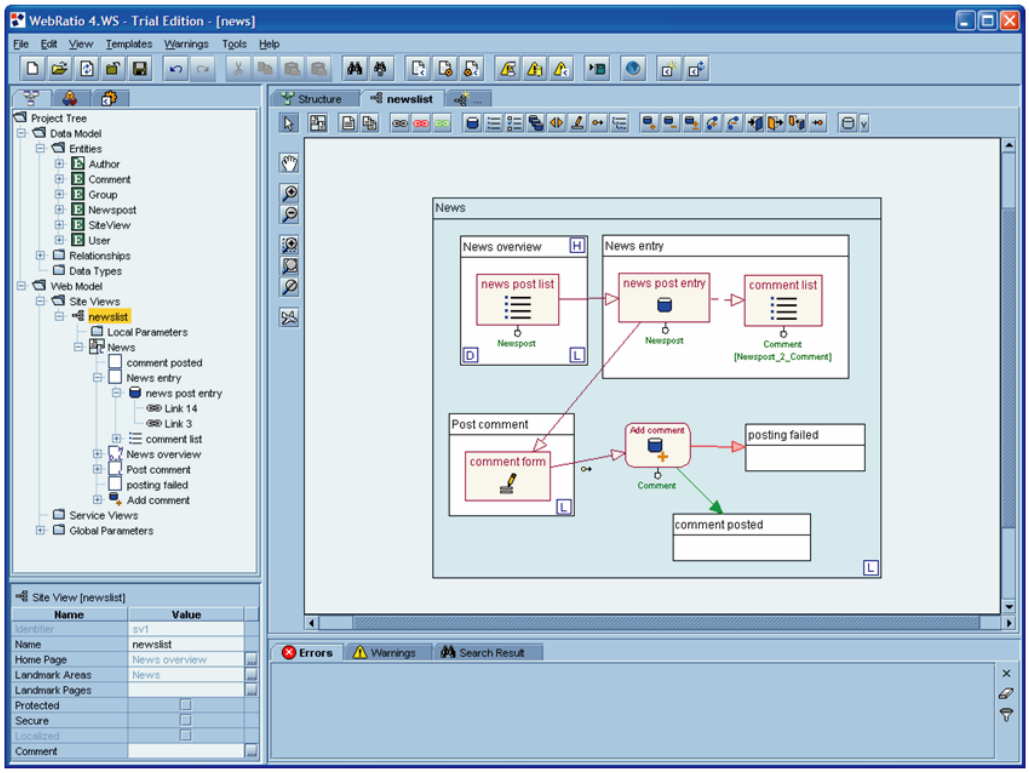

Figure 3.7: WebRatio site view example [@WebRatio]

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.

Chapter 4

A Concern-Oriented Component Model for Adaptive Web Applications

"If we knew what it was we were doing, it would not be called research, would it?"[1]

The previous chapter reviewed and compared existing Web engineering solutions aimed at the development of adaptive Web-based systems. It investigated both model-driven approaches addressing the conceptual design and high-level specification of Web applications, as well as component-oriented and document-centric solutions primarily focusing on their presentation and implementation aspects. It was pointed out that component-based reuse is a crucial issue of Web engineering. Still, there is lacking support for the efficient creation of adaptive multimedia Web presentations from reusable and configurable implementation entities.

To fill this gap, this chapter presents a *concern-oriented component model*[2] for adaptive dynamic Web applications. The term *concert-oriented* denotes its explicit support for the clear separation of concerns involved in a Web application, i.e. it enables to compose adaptive Web presentations by the aggregation and linkage of reusable *document components* that encapsulate different application *concerns* such as content, structure, navigation, semantics, presentation (as well as their corresponding adaptation issues) on different abstraction levels. The resulting document component structures can be automatically translated to Web presentations that are adapted to a specific user, device, output format, or other context information.

The remainder of this chapter is structured as follows. First, in Section 4.1, the document-centric component concept is discussed, and a number of requirements towards a document model for adaptive Web presentations are mentioned. Based on these requirements, the following sections (4.2 to 4.4) present the component-based document model and its XML-based description language in detail. The different abstraction levels of document components, their support for adaptation, as well as the concept of document component templates are explained by examples. In Section 4.5 a pipeline-based document generator aimed at the on-the-fly publishing of component-based adaptive Web applications is presented. Finally, selected benefits of the proposed model are discussed in Section 4.6.

[1] Albert Einstein (1879 - 1955)

[2] The component model was developed within the scope of the AMACONT project and is also often referred to as the AMACONT component model.The thesis presents the author's contributions to the model, based on requirements towards the efficient authoring of adaptive Web applications from reusable components.

4.1 Declarative Document Components

Szypersky defines in [Szyperski 1998] the notion of a software component as follows:

Definition 4.1 (Software component) *A software component is a unit of composition with contextually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition [Szyperski 1998].*

This original definition considers a software component as a *binary* unit of composition that is typically based on an imperative implementation. Still, in recent years a number of declarative component approaches have emerged, aiming at describing not only the interfaces and properties, but even the functionality of reusable implementation entities in a declarative human-readable form [Dachselt 2004, Aßmann 2005]. This shift from the traditional program-centric to a document-centric application development paradigm has been especially characteristic for the World Wide Web. As prominent examples well-established standards such as SMIL (Synchronized Multimedia Integration Language [Bulterman et al. 2005]) or SVG (Scalable Vector Graphics [Ferraiolo and Jackson 2003]) can be mentioned, that facilitate to create complex Web and multimedia applications on top of XML-based declarative document descriptions. Furthermore, as discussed in Section 3.2, there already exists a number of approaches (WCML, CONTIGRA, CHAMELEON) that explicitly focus on reusing declarative Web implementation entities in a component-wise manner.

The component-based document model presented in this chapter was inspired by the previously discussed approaches WebComposition (WCML), CONTIGRA, and CHAMELEON. Adopting their document-centric component concept, it supports the development of personalized ubiquitous Web presentations from declarative reusable implementation entities called *document components* [Fiala et al. 2003a, Fiala et al. 2003b].

Document components are XML documents, instances of a specific XML-grammar describing adaptive Web content. They can be defined on different abstraction levels, each representing a separate application concern (e.g. content, structure, semantics, navigation, presentation) involved in a Web presentation. Document components are unequivocally identified by a unique identifier and further described by appropriate metadata. Acting as their interface definition, this metadata specifies their properties (such as their structure, layout) as well as their adaptive behavior. Web sites are constructed by configuring, aggregating, and interlinking components to complex component structures. During document generation, these abstract document structures are dynamically translated into Web pages in a concrete output format and are automatically adapted to the current usage context.

Even though document fragments are no (binary) software components according to the above mentioned software engineering definition of Szyperski, note that they show a lot of similarities to the classic component concept. They are system independent and reusable units, representing a certain functionality that can be combined to complex applications. Furthermore, they provide a clearly defined interface described by specific metadata, allowing for configuration and aggregation on higher component levels[3].

While the specifics of the component model and its XML-based description language are described in detail in the subsequent sections, the following list comprises the most important

[3]Though being different from the component definition of Szypersky, note that the concept of document components also corresponds to the notion of components of a hypermedia system as defined by the Dexter reference model [Halasz and Schwartz 1994].

requirements that were considered during their design. These requirements were derived from the previously discussed shortcoming of existing solutions (see Section 3.2.10) as well as the main goal of the thesis: the efficient component-based authoring of adaptive Web applications from reusable implementation artefacts[4].

1. **Rigorous separation of different concerns** involved in a Web presentation, such as content, structure, navigation, layout, and adaptation.
2. **Reusability** of (parts) of Web presentations on both different abstraction levels and of different granularity (i.e. from fine-granular atomic resources to coarse-grained complex Web document structures).
3. **Composability** of reusable document parts to aggregates (composites) that again act as reusable units and can be subject to further composition.
4. **Ease of configuration and adjustment** of parts of Web presentations of different granularity through well-defined interfaces described by appropriate descriptive metadata.
5. **Inherent adaptation support** by built-in language constructs allowing to refer to user and context model parameters. Provision of generic facilities for defining conditional alternatives (of different concerns such as content, structure, navigation, presentation) depending on the actual client device, user preferences, and the entire usage context.
6. **Platform and device independence** by abstraction from concrete implementation platforms, client devices, Web browsers, and specific Web output formats.
7. **Support for media integration** allowing to incorporate both existing and future media and internet document formats.
8. **Support for data-driven Web presentations** based on component templates that can be extended (i.e. "filled") with dynamically retrieved data at run-time.
9. **Interoperability** with existing internet standards by extensive usage of approved XML technologies.
10. **Extensibility and flexibility** support through modularity as well as well-defined metadata interfaces.

4.2 A Component-based Document Model and its XML Description Language

As discussed above, the proposed document model is based on the notion of declarative *document components*. These are instances of a specific XML-grammar that represent different application and adaptation concerns on various abstraction levels and can thus be configured, aggregated, and interlinked to complex adaptive Web presentations.

In order to support platform independence and interoperability, the document format was specified based on XML-technology. It is defined by a set of XML Schema documents

[4]Note that these requirements also served as the basis for comparing existing component-based and document-oriented approaches in Section 3.2.10

[Fallside and Walmsley 2004], each specifying a separate aspect of the document model, such as composition, metadata, adaptation, presentation, etc. Note that XML Schema allows for more powerful definitions than a DTD by supporting namespaces, type safety, reuse of type definitions, more exact expressions of cardinality, self-documentation as well as type inheritance.

The document model allows to define components on different abstraction levels (also referred to as component levels), each responsible for a given application concern. In Figure 4.1 the corresponding level-based architecture is illustrated. It is constituted of *media components*, *content unit components*, *document components*, and the *hyperlink view*. Based on a number of representative examples, the following sections introduce each component level in more detail.

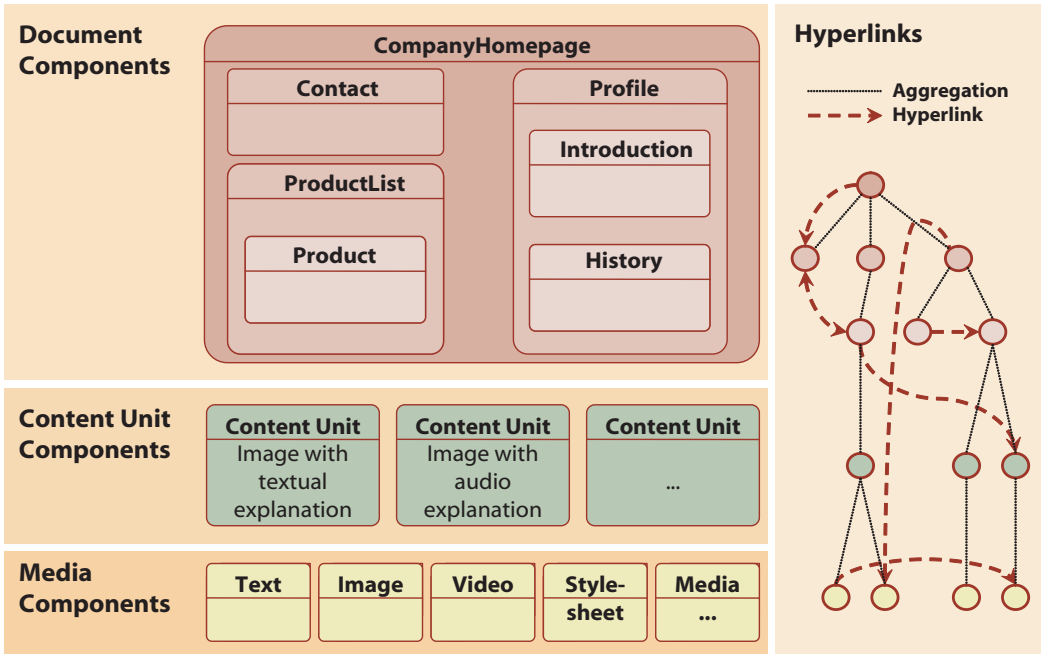

Figure 4.1: A concern-oriented component model for adaptive Web sites [Fiala et al. 2003a]

4.2.1 Media Components

On the lowest level of the component hierarchy there are *media components* that encapsulate particular media assets by describing them with specific XML-based metadata. The set of supported media assets comprises text, toggle-text, structured text (e.g. HTML or XML code fragments), images, sound, video, Java applets, Flash and Director presentations, but may be extended arbitrarily. Dynamically created media components (e.g. HTML fragments or pictures generated on the fly) are also supported, provided that the corresponding metadata is delivered, as well. Furthermore, even whole documents (such as HTML pages) might be wrapped to media components in order to optimize page generation and support flexible authoring.

As an example, the code snippet in Listing 4.1 describes a simple image component. It is unequivocally identified by its *name* attribute which is unique for all components. The *layer* attribute (see line 1) dictates that it is a component from the level of media components.

The namespace *aco* identifies the XML schema definition AmaComponent.xsd which specifies the different levels, types as well as the composition hierarchy of components. Similarly, the namespace *amet* identifies the XML schema document AmaMetaInformation.xsd aimed at defining the possible metadata attributes of components. The metadata attributes of media components (aimed at describing their technical properties) were inspired by the appropriate descriptors of the MPEG-7 standard [Manjunath et al. 2002]. In the presented example they describe the image object's size dimensions (*width* and *height*) as well as its source location.

```
1 <aco:AmaImageComponent name="myImage" layer="Media">
2   <aco:MetaInformation>
3     <amet:ImageMetaData>
4       <amet:source>images/myimage.jpg</amet:source>
5       <amet:width>500</amet:width>
6       <amet:height>300</amet:height>
7     </amet:ImageMetaData>
8   </aco:MetaInformation>
9 </aco:AmaImageComponent>
```

Listing 4.1: Simple media component example

In order to support for form-based user interactions, the component-based document format allows to define Web form elements (such as input fields, select lists, check boxes, etc.) as media components [Hoja 2005]. This can be done either by using a *structured text component* that contains a form description in a specific Web output format (e.g. HTML) or by using a dedicated *XForms component* encapsulating a form description based on the XForms standard [Dubinko 2004]. By separating the data model, the behavior and the presentation aspects of Web forms, XForms allows for the specification of interaction elements in a device independent way.

4.2.2 Content Unit Components

On the second component level, media components are grouped to so-called *content units*. The purpose of such groupings is the explicit combination of media elements that belong together concerning their content[5] and thus should not be handled separately. For example, an image with its appropriate textual description can constitute a content unit. Further predefined content unit types are *audio component with text component* or *collection of media components*, but arbitrary extensions are supported. Note that the definition of such collections of media objects is a key factor of component reuse.

As an example, the code snippet in Listing 4.2 illustrates a simple content unit containing an image and a text component (both from the media component layer). Whereas in this case these subcomponents are "physically" aggregated to a content unit (see the *SubComponents* tag in line 5), note that it is also possible to include subcomponents by reference from another XML document. The latter mechanism allows to efficiently reuse components in different composition scenarios.

Since the media components constituting a content unit belong together, they also have to be presented together in a Web presentation. Therefore, appropriate metadata describing their layout, i.e. their relative spatial arrangement on the generated hypermedia pages is needed. For this purpose content unit components contain additional layout descriptions (as part of their own meta data information). Inspired by the layout manager mechanism of

[5] e.g. in order to represent a given concept of an application domain


```
1  <aco:AmaImageTextComponent name="myUnit" layer="ContentUnit">
2    <aco:MetaInformation>
3      ...
4    </aco:MetaInformation>
5    <aco:SubComponents>
6       <aco:AmaTextComponent name="myText" layer="Media">
7          ...
8       </aco:AmaTextComponent>
9       <aco:AmaImageComponent name="myImage" layer="Media">
10         ...
11      </aco:AmaTextComponent>
12   </aco:SubComponents>
13 </aco:AmaImageTextComponent>
```

Listing 4.2: Simple content unit example

the Java language, these properties describe a size- and client-independent layout allowing to abstract from the exact resolution of the display or the browser's window. A detailed description of these layout descriptors will be given in Section 4.3.2.

4.2.3 Document Components

The uppermost component level contains so-called *document components*. These are meaningful presentation units of a Web application that typically play a certain semantic role, such as a news column, a product presentation, a navigation bar or even a whole Web page. Take for example a document component bearing the semantic role "company product". It could aggregate (and thus be represented by) the content unit from Listing 4.2 that again contains an image component and a text component.

Document components can not only contain content units, they can also aggregate other document components. This aggregation results in an arbitrary deep hierarchy of document components which describes the logical structure of a component-based Web document. The root component (element) of a component-based adaptive Web document has to be always a component from the document component level. Such a top-level document component contains all the information to be shown on the user's display at a particular moment. According to the component hierarchy example depicted in Figure 4.1, Listing 4.3 demonstrates the corresponding aggregation of components[6].

The document components constituting a Web presentation typically portray some meaningful concepts that are represented by (a set of) content units or aggregated document components. Still, the component-based document format does not prescribe to explicitly specify the semantics of document components by unequivocally assigning them to parts (i.e. concepts or attributes) of a specific conceptual model or domain ontology. The component hierarchy of a Web page is specified by component authors in the authoring process which will be described in detail in Chapter 5. There it will be also shown how component-based Web applications presenting information on a well-defined application domain can be systematically developed.

In analogy to content units, the presentation of document components on a Web page is specified by layout properties defining the spatial adjustment of their aggregated document

[6]Similar to content unit components, document components can not only "physically" contain their subcomponents (as aggregated XML subelements), they can also include subcomponents contained in separate XML documents by reference.


```
1  <aco:AmaCompanyHomepageComponent name="Company" layer="DocumentComponent">
2    <aco:SubComponents>
3      <aco:AmaProductListComponent name="ProdList" layer="DocumentComponent">
4        <aco:SubComponents>
5          ...
6          <aco:AmaProductComponent name="Product" ...>
7            <aco:SubComponents>
8              <aco:AmaImageTextComponent ... layer="ContentUnit">
9                <aco:SubComponents>
10                 <aco:AmaImageComponent name="myImage" layer="Media">
11                  ...
12                 </aco:AmaImageComponent>
13                 <aco:AmaTextComponent name="myText" layer="Media">
14                   ...
15                 </aco:AmaTextComponent>
16               </ac:SubComponents>
17             </aco:AmaImageTextComponent>
18           </aco:SubComponents>
19         </aco:AmaProductComponent>
20         ...
21       </aco:SubComponents>
22     </aco:AmaProductListComponent>
23      ...
24     <aco:AmaContactComponent name="Contact" layer="DocumentComponent">
25      ...
26     </aco:AmaContactComponent>
27     ...
28   </aco:SubComponents>
29 </aco:AmaCompanyHomepageComponent>
```

Listing 4.3: Document component composition example

components. However, the layout properties of a given component only describe the presentation of its immediate subcomponents which encapsulate their own layout information in a standard component-based way. Thus, in order to provide reuse and configuration, each composite component stores and manages its layout information on its own. The concept of adaptive layout managers will be introduced in detail in Section 4.3.2.

4.2.4 Hyperlink Components

Whereas aggregation relationships between components are expressed on the level of content unit components and document components, navigational relationships between components are defined by so-called *hyperlink components*, each defining an (optionally typed) directed link between two "non-hyperlink" components[7]. Two kinds of of hyperlink components exist: simple hyperlink components and hyperlink list components. While a simple hyperlink component constitutes a directed (and optionally typed) navigational relationship between two components (that act as its *source* and *destination* anchors), a hyperlink list component is a collection of simple links aimed at the easier definition of index-like navigation structures.

For the sake of efficient document reuse, hyperlinks are always defined within the scope

[7]Even though hyperlinks are also considered as components, the document model does not allow to specify hyperlinks between hyperlinks, i.e. the end points of a hyperlink may be only media components, content unit components or document components.

of a document component. The fact that a hyperlink component is specified within the scope of a document component means that at least its source anchor is contained in (the subcomponent hierarchy of) that document component. Consequently, the reuse of that document component in another composition scenario also implies the reuse of its associated hyperlinks.

The specification of hyperlink components is based on the XPath [Berglund et al. 2004] and XPointer [DeRose et al. 2002] standards of the W3C. The source anchor of a hyperlink component is unequivocally described by the identifier of its source component and an optional offset. The source component of a hyperlink component is (if not further specified) either the document component in the scope of which it was defined, or an arbitrary component of its subcomponent hierarchy. Furthermore, an (optional) XPointer expression defining an offset of the source link anchor in that component can be also specified. Typically, this offset is needed in order to define a hyperlink anchor that is assigned only to a fragment of a text component.

The destination of a hyperlink component is either a component in the subcomponent hierarchy of the document component in which it was defined, or an arbitrary component in another component-based Web document. Besides, external link destinations pointing to arbitrary URIs are also allowed, thus facilitating to reference any external content.

Finally, hyperlink components can be optionally assigned a *type* and/or a *class* attribute, as well. The former one allows for the specification of typed navigational relationships between components. However, the possible hyperlink types are not prescribed by the document format and can thus be specified by component authors (e.g. by exploiting existing link type classifications [Casteleyn and De Troyer 2002]). The latter one (class attribute) aims at assigning a presentation class to hyperlinks. Links with different class attributes can be visualized differently (by the definition of appropriate CSS media components attached to their containing components) and thus be used for realizing link adaptation techniques such as link annotation or link disabling (see Section 2.2.3).

The code snippet depicted in Listing 4.4 shows a simple hyperlink component defining a navigational relationship between two components. It was defined in the scope of the document component that was visually shown in Figure 4.1, representing a company homepage. The source anchor of this link is the component called “Introduction” (see the *From* element in line 5), its destination is the component called “Contact”. Since in this case both components are subcomponents of “Company”, its reuse in another composition scenario also implies the reuse of this hyperlink component. Note that in this example no link type was defined.

4.3 Adaptation Support

The component-based document format supports a separation of concerns by distinguishing between different abstraction levels of components as well as their interlinking. Important aspects of a Web presentation, such as content, structure, semantics, navigation, etc. are handled on different component levels, thus enabling a better reuse and configurability of (even parts of) a Web presentation. Furthermore, this level-based model also facilitates the efficient separation of different adaptation targets (i.e. parts or aspects of a Web presentation to be adapted). The following paragraphs mention typical adaptation concerns to be considered on the different component levels.

Adaptation of Media Components: Adaptation on the level of media components pri-


```
1  <aco:AmaCompanyHomepageComponent name="Company">
2    <aco:Hyperlinks>
3     ...
4      <aco:AmaComponentLinkComponent name="Link_1" layer="Hyperlink">
5        <aco:From component="Introduction"/>
6        <aco:To component="Contact"/>
7      </aco:AmaComponentLinkComponent>
8     ...
9    </aco:Hyperlinks>
10   <aco:SubComponents>
11      ...
12   </aco:SubComponents>
13 </aco:AmaCompanyHomepageComponent>
```

Listing 4.4: Link component example

marily concerns media quality and is required to consider various device capabilities or other technical constraints. For instance, in a device independent Web presentation it is necessary to provide different instances of a certain picture (image component) with variable size, color depth or resolution in order to automatically adapt to various display types. Similarly, a video component should be provided with different bit rates to be adjusted to the currently available bandwidth.

Adaptation of Content Unit Components: Adaptation on the level of content units concerns the type and number of the included media components (that actually constitute the structure of a content unit) and can be defined for different purposes. On the one hand, technical properties of client devices can be taken into account. For example, on a company homepage a user with a high performance client could be shown a short video clip of the presented article, while others with low-performance terminals would be presented an image and a textual description of that product. On the other hand, the adaptation of content units may also consider semantic user preferences. Consider again the case of two customers, one of them preferring detailed textual descriptions, the other visual information. While the presentation for the first user might include content units primarily referring to textual objects, the other could be shown multimedia information, respectively. These kinds of adaptation were approved as very profitable e.g. in the TELLIM project [Jörding 1999, Hölldobler 2001] focusing on electronic shopping applications.

Adaptation of Document Components: Adaptation of document components concerns the overall component hierarchy, i.e. the way document components are nested. This results in different variations of component trees (and thus Web page structures) depending on user preferences and client properties. For instance, consider the component hierarchy shown in Figure 4.1 that represents a company homepage. Depending on the interests and previous knowledge of users, the resulting Web page could be generated differently. For internal users working as employees of the company, the document component presenting the company's history should not be inserted in the generated Web page.

Nevertheless, note that the adjustment of document components may also depend on other parameters, e.g. client capabilities. Let us consider the Web portal of a railway company as an example. Whereas the desktop PC version of such a Web site might include numerous parts such as timetable, online-shop, online travel agency, etc., the

WAP version is mostly restricted to a minimal set of services, e.g. merely an online timetable.

Adaptation of Hyperlink Components: Finally, adaptation on the level of hyperlink components implies the adjustment of hyperlink targets as well as navigation structures according to information describing the actual usage context. It might concern the usage of different adaptive navigation techniques, such as the conditional inclusion or annotation of hyperlinks, the offering of navigation alternatives, etc. This adjustment of hyperlink structures can allow a personalized navigation through a component-based Web presentation.

In order to realize these adaptation scenarios, the component model facilitates two generic adaptation mechanisms. First, it is possible to specify context-dependent *adaptation variants* for (different aspects of) components. Second, a facility is provided for describing the *adaptive layout* of components by means of client-independent layout descriptors that can be automatically adapted to different output formats. The rest of this section describes these two mechanisms in more detail and illustrates them by representative examples.

4.3.1 Describing Adaptation Variants

To provide generic adaptation support, components (but also their parameters) may include a number of (conditional) alternatives. As an example, the definition of an image component might include two variants for color and monochrome displays. Similarly, the number, structure, spatial arrangement, and linking of subcomponents within a composite component can also vary depending on the current usage context. The decision, which alternative is selected, is made during document generation (see Section 4.5) according to a *selection method* which is also encapsulated by the component.

Such selection methods are chosen by component developers at authoring time and can represent arbitrarily complex conditional expressions parameterized by context model parameters. These parameters describe the user's actual usage context (e.g. his knowledge, characteristics, preferences, client device, location, etc.) and will be described in more detail in Section 4.5.3. The XML-grammar for selection methods was specified as an XML Schema definition and allows for the declaration of user model parameters, constants, variables, and operators, as well as complex conditional expressions (such as IF-THEN-ELSE or SWITCH-CASE) of arbitrary complexity [Fiala et al. 2003a, Fiala et al. 2003c]. The appropriate reconfiguration (adjustment) of a component's adaptation logic allows to reuse it in different adaptation scenarios.

The example code in Listing 4.5 demonstrates the definition of a media component's variants as well as a corresponding selection method. It is taken from a Web presentation offering different versions of media elements for different end devices (in high quality for desktop computers and low quality for mobile clients), i.e. the appropriate variant is chosen based on the actual client device. The possible variants to be selected are defined in the *Variant* elements (see the lines 21 and 24). Again, the the namespace *aada* references the XML schema definition AmaAdaptation.xsd which specifies the grammar for describing variants and their corresponding selection methods.

The selection method is contained in the *Logic* tag (line 5), which is in this particular example an IF-THEN-ELSE construct. The condition to be evaluated is specified within the *Expr* element. Based on the Polish Notation (PN)[8], it is an arithmetic expression consisting

[8]The PN-based definition of arithmetic expressions allows for their proper validation, as well as evaluation


```
1  <aco:AmaImageComponent name="myAdaptiveImage">
2    <aco:MetaInformation>
3      <amet:ImageMetaData>
4        <aada:Variants>
5          <aada:Logic>
6             <aada:If>
7               <aada:Expr>
8                 <aada:Term type="=">
9                   <aada:UserParam>Device</aada:UserParam>
10                  <aada:Const>Desktop</aada:Const>
11                </aada:Term>
12              </aada:Expr>
13              <aada:Then>
14                 <aada:ChooseVariant>HQ_Picture</aada:ChooseVariant>
15              </aada:Then>
16              <aada:Else>
17                <aada:ChooseVariant>LQ_Picture</aada:ChooseVariant>
18              </aada:Else>
19            </aada:If>
20         </aada:Logic>
21         <aada:Variant name="HQ_Picture">
22              ...
23         </aada:Variant>
24         <aada:Variant name="LQ_Picture">
25              ...
26         </aada:Variant>
27       </aada:Variants>
28     </amet:ImageMetaData>
29   </aco:MetaInformation>
30 </aco:AmaImageComponent>
```

Listing 4.5: Describing adaptive variants

of a simple term (*Term*) that compares the context model parameter *Device* (referred to as `<UserParam>Device</UserParam>`) with the constant "Desktop"[9]. The elements *Then* and *Else* specify either (like in this case) the appropriate variants to be selected if the condition holds or not, or contain another IF-THEN-ELSE or SWITCH-CASE construct. However, the definition of the else-branch is optional. Whenever it is missing, the selection logic describes *conditional inclusion*, i.e. the addressed variant is presented if and only if the condition holds.

Note that such a component containing adaptation variants acts as an adaptive and reusable "Web site building block". Based on its internal adaptation logic, it can automatically adjust (or reconfigure) itself to the current usage context. Furthermore, since both this logic and the corresponding adaptation variants are inherently contained by it, it can be reused as an adaptive unit of composition in different Web documents. As a consequence, it even fulfills the definition of a *self-adaptive software* provided by [Oreizy et al. 1999][10].

While the adaptation logic described in Listing 4.5 is based on a simple conditional expression in the IF-THEN-ELSE style, Listing 4.6 shows another example that uses a SWITCH-

by XSLT stylesheets [Kay 2004].

[9] The children elements of a Term element might be again Term elements, thus allowing to define arbitrarily complex arithmetic expressions.

[10] According to Oreizy et al., "self-adaptive software modifies its own behavior in response to changes in its operation environment" [Oreizy et al. 1999].

CASE construct allowing to easily select from more than two different alternatives. It is taken from an eLearning example and aims at the selection of an appropriate document component based on the expertise level of the actual user (Beginner, Advanced or Expert). Note the optional *Default* element in line 19 aimed at determining the variant to be selected if none of the other cases holds. It can be used to guarantee that the resulting component is not empty.

```
1  <aada:Variants>
2    <aada:Logic>
3       <aada:Switch>
4         <aada:Expr>
5           <aada:Term>
6             <aada:UserParam>Expertise</aada:UserParam>
7           </aada:Term>
8         </aada:Expr>
9         <aada:Cases>
10          <aada:Case value="Beginner">
11            <aada:ChooseVariant>Beginner_Version</aada:ChooseVariant>
12          </aada:Case>
13          <aada:Case value="Advanced">
14            <aada:ChooseVariant>Advanced_Version</aada:ChooseVariant>
15          </aada:Case>
16          <aada:Case value="Expert">
17            <aada:ChooseVariant>Expert_Version</aada:ChooseVariant>
18          </aada:Case>
19          <aada:Default>
20            <aada:ChooseVariant>Beginner_Version</aada:ChooseVariant>
21          </aada:Default>
22        </aada:Cases>
23      </aada:Switch>
24   <aada:Logic>
25 </aada:Variants>
```

Listing 4.6: Describing adaptation variants (Example 2)

As a matter of course, adaptation variants may be defined on all component levels. When adjusting complex document structures to the current usage context the appropriate selection methods are processed recursively, i.e. in a top-down-manner (beginning from the top-level document component). The run-time evaluation process of such selection methods will be described in detail in Section 4.5.

Parameter Substitution While the concept of adaptation variants allows to adjust various component properties to parameters of the actual usage context, in some cases it is also required to include the values of those parameters into the generated Web presentation. The reason for this can be the intention to inform the user about his/her context or the need for a better personalization of the Web application. For this reason the component-based document format allows to utilize *parameter substitution*. As an example, the code snippet shown in Listing 4.7 depicts a text component, the content of which is parameterized by the context parameter *LastName* denoting the surname of the current user. When substituted by the surname of the author of this thesis following text would be created: “You are logged in as Mr/Ms Fiala.”.

Component (variants) combined with selection methods and parameter substitution allow to describe the adaptation behavior of parts of a Web presentation in a declarative component-


```
1  <aco:AmaTextComponent name="GreetingText" layer="Media">
2    <aco:MetaInformation>
3      <amet:TextMetaData>
4        <amet:value>
5         You are logged in as <aada:UserParam>Salutation</aada:UserParam>
6         <aada:UserParam>LastName</aada:UserParam>.
7        </amet:value>
8      </amet:TextMetaData>
9    </aco:MetaInformation>
10 </aco:AmaTextComponent>
```

Listing 4.7: Context parameter substitution example

wise manner. Nevertheless, the complexity of their underlying XML grammar makes it very difficult to manually author such logical expressions and calls for intuitive visual tools facilitating the graphical definition of appropriate XML documents. For this purpose the AMACONTBuilder, a visual authoring tool for component-based Web documents will be presented in Section 5.2.

4.3.2 Describing Adaptive Layout

In order to describe the spatial arrangement (layout) of components, the component-based document format allows to attach XML-based layout descriptions [Fiala et al. 2003a] to them. Inspired by the layout manager mechanism of the Java language (AWT and Swing) and the abstract user interface representations of UIML [Abrams and Helms 2002] or XIML [Puerta and Eisenstein 2002], they describe a client-independent layout allowing to abstract from the exact resolution of the browser's display. Note that layout managers of a given component only describe the presentation of its immediate subcomponents which encapsulate their own layout information in a standard component-based way.

The available layout managers are depicted in Figure 4.2. `OverlayLayout` allows to present components on top of each other. `BoxLayout` lays out multiple components either vertically or horizontally. `BorderLayout` arranges components to fit into five regions: north, south, east, west, and center. It is especially useful to specify the layout of Web presentations containing a header, a footer, sidebars, and a main content area. Finally, `GridTableLayout` enables to lay out components in a grid with a configurable number of columns and rows. Though it can be also realized by nested `BoxLayouts`, it was implemented separately because Web applications often present dynamically retrieved sets of data in a tabular way.

The code snippet in Listing 4.8 depicts a possible layout description of the content unit from Listing 4.2 based on the layout manager `BoxLayout`. The contained image component (aligned right) and the text component (aligned left) are arranged above each other, taking 30 and 70 percent of the available vertical space. Note that the *alay* namespace references the XML schema definition AmaLayout.xsd which specifies the possible layout managers and their specific attributes.

Layout managers are formalized as XML tags with specific attributes [Fiala et al. 2004a]. Two kinds of attributes exist: *layout attributes* and *subcomponent attributes*. Layout attributes declare properties concerning the overall layout and are defined in the corresponding layout tags. As an example the `axis` attribute of `BoxLayout` (see line 5 in Listing 4.8) determines whether it is laid out horizontally or vertically. Subcomponent attributes describe

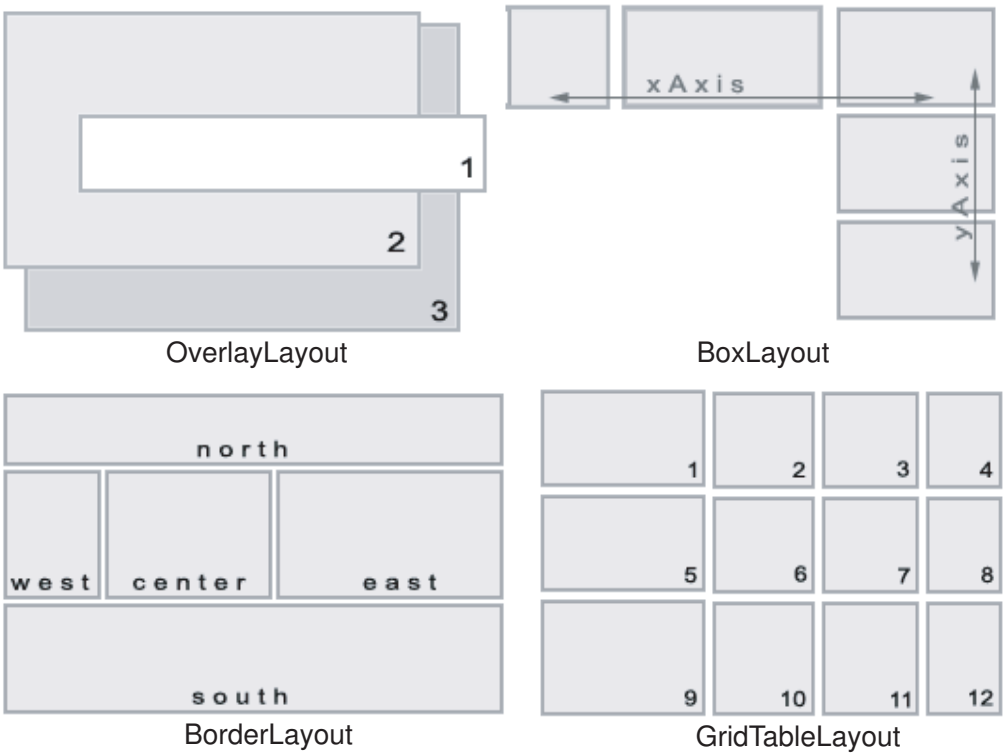

Figure 4.2: Abstract layout managers

how each referenced subcomponent has to be arranged in its surrounding layout. For instance, the `align` attribute of `myImage` declares it to be right-justified. Table 4.1 summarizes the possible attributes of the layout manager `BoxLayout` by describing their names, role, usage (required or optional), and possible values.

Layout Attributes	Meaning	Usage	Values
`axis`	Orientation of the BoxLayout	req.	xAxis|yAxis
`space`	Space between subcomponents	opt.	int
`width`	Width of the whole layout	opt.	string
`height`	Height of the whole layout	opt.	string
`border`	Width of border between subcomponents	opt.	int
Subcomponent Attributes	Meaning	Usage	Values
`align`	Horizontal alignment of subcomp.	opt.	left|center|right
`valign`	Vertical alignment of subcomp.	opt.	top|center|bottom
`ratio`	Space taken by subcomponent	opt.	percentage
`wml_visible`	Should be shown on same WML card?	opt.	boolean
`wml_desc`	Link description for WML	opt.	string

Table 4.1: BoxLayout attributes [Fiala et al. 2004a]


```
1  <aco:AmaImageTextComponent name="myUnit" layer="ContentUnit">
2    <aco:MetaInformation>
3      <amet:LayoutProperties>
4        <alay:LayoutManager>
5          <alay:BoxLayout axis="yAxis" border="1">
6            <alay:ComponentRef ratio="30%" align="right">
7                myImage
8            </alay:ComponentRef>
9            <alay:ComponentRef ratio="70%" align="left">
10               myText
11           </alay:ComponentRef>
12         </alay:BoxLayout>
13       </alay:LayoutManager>
14     </amet:LayoutProperties>
15   </aco:MetaInformation>
16   <aco:SubComponents>
17     ....
18   </aco:SubComponents>
19 </aco:AmaImageTextComponent>
```

Listing 4.8: Layout manager example

Even though most attributes are device independent, two platform-dependent attributes were also added in order to consider the specific card-based structure of WML presentations. Note the optional attribute `wml_visible` that determines whether in a WML presentation the given subcomponent should be shown on the same card. If not, it is put onto a separate card that is accessible by an automatically generated hyperlink, the text of which is defined in `wml_description`. This mechanism of content separation is used since the displays of WAP-capable mobile phones are very small.

The exact rendering of media objects is done at run time by XSLT stylesheets that transform components with such abstract layout properties to Web document fragments in a specific output format (see Section 4.5). A number of stylesheets for converting those descriptions to formats such as XHTML (and its different modules), cHTML, WML, etc. have been developed [Fiala et al. 2003a, Hinz et al. 2004].

While adaptive layout managers support the automatic adaptation of abstract layout descriptions to different output formats, component authors can also use them in combination with adaptation variants, thus being able to specify even more precise layout adaptations that are explicitly parameterized by the current usage context. For example, while the presentation of a content unit containing a list of images could be realized as a GridTableLayout on a browser with sufficient horizontal resolution, another client device with a small display width should render it according to a vertical BoxLayout. The code snippet in Listing 4.9 depicts such combined layout adaptation definition. The appropriate layout manager is selected according to the horizontal resolution (denoted by the context parameter *InnerSizeX*) of the current browser window. Thus, the layout of a document component can be adapted independently of its other aspects, such as its subcomponent, hyperlinks, etc.

Again, the complexity of the XML code shown in Listing 4.8 and Listing 4.9 makes it obvious that the manual creation of layout manager descriptions with a text or XML editor might be a cumbersome task. For the visual authoring of adaptable layouts Section 5.2.4 will introduce the Layout Editor module of the authoring tool AMACONTBuilder.


```
<alay:LayoutManager>
  <alay:Variants>
    <aada:Logic>
      <aada:If>
        <aada:Expr>
          <aada:Term type="gt">
            <aada:UserParam>InnerSizeX</aada:UserParam>
            <aada:Const>600</aada:Const>
          </aada:Term>
        </aada:Expr>
        <aada:Then>
           <aada:ChooseVariant>GridTableLayout_Variant</aada:ChooseVariant>
        </aada:Then>
        <aada:Else>
           <aada:ChooseVariant>BoxLayout_Variant</aada:ChooseVariant>
        </aada:Else>
      </aada:If>
    </aada:Logic>
    ...
  </alay:Variants>
  <aada:Variant name="GridTableLayout_Variant">
     ...
  </aada:Variant>
  <aada:Variant name="BoxLayout_Variant">
     ...
  </aada:Variant>
</alay:LayoutManager>
```

Listing 4.9: Combined context dependent layout adaptation

4.4 Document Component Templates

The document components introduced above are static, i.e. they represent a concrete piece of (adaptable) Web content, such as a specific instance of an image (as a media component instance with optional quality alternatives) or a chapter in an eLearning course (as a document component instance). Still, in order to provide support for data-driven Web applications, like online-shops, product presentations, e-galleries, etc., there is a need for components that are created from dynamic data sources on-the-fly.

For this purpose so-called document component templates have been introduced. These are component skeletons (i.e. component instances containing placeholders) that declare the structural, behavioral and layout aspects of components independent of their actual content [Fiala et al. 2004b]. At run-time, component templates are extended (i.e. filled) with content that is dynamically queried (retrieved) from a data source. Therefore, they are associated with a query that can be parametrized by arbitrary request and/or context model parameters. As an example, the XML-code in Listing 4.10 describes a simple media component template[11]:

The namespace *t_aco* dictates that the component acts as a component template. The query associated with it is described in the *query* attribute of its starting tag. In this particular case this is an SQL expression querying a table of a relational database that contains

[11] As a matter of course, the same mechanism is applicable for content unit component templates, document component templates, and hyperlink component templates.


```
1  <t_aco:AmaImageComponent name="productimage" type="template"
2                           query="SELECT source, width, height
3                                  FROM productimages
4                                  WHERE ID=substitute(id)">
5    <aco:MetaInformation>
6     <amet:ImageMetaData>
7       <amet:source><t_temp:query field="source"/></amet:source>
8       <amet:width><t_temp:query field="width"/></amet:width>
9       <amet:height><t_temp:query field="height"/></amet:height>
10    </amet:ImageMetaData>
11   </aco:MetaInformation>
12 </t_aco:AmaImageComponent>
```

Listing 4.10: Simple component template example

images of a company's products described by appropriate metadata[12]. The expression *substitute(id)* references the *id* request parameter of the actual HTTP request. The values from the corresponding result set are referred to as `<t_temp:query field="myname"/>`, where *myname* is the name of a given field. As an example, the resulting media component's *source* attribute is substituted by the value of the database field *picturesource*.

While in this example all metadata attributes of the image component are dynamically retrieved, note that it is also possible to define selected attributes as constants so that they remain unchanged for all instantiations. Furthermore, it is also possible to parameterize a template's query by arbitrary context model attributes. In such a case these parameters are substituted by their corresponding values before the query is executed, i.e. the data to be inserted is queried in a personalized way.

The above example describes a single media component template, the actual content of which is delivered by a dynamic data source. Still, in a data-driven Web application it is not only required to dynamically retrieve single content (component) elements, but also component sets, such as all books of a given author (in an electronic book store), or all employees of a department (in an institutional Web site). For such cases the component-based document format allows to define so-called *iterative component templates*. Again, a simple example of a content unit containing a dynamic set of image components is depicted in Listing 4.11.

The content unit component template defined in this example contains (as its subcomponent) a dynamically iterated image component (see the *iterate* attribute in line 16). Consequently, this media component is iterated (repeated) according to the size of the result set delivered by the template's query (line 3) so that each iteration is parameterized by the corresponding result. To ensure that the resulting image components have unequivocal *name* attributes the *idField* attribute denoting a unique identifier field in the query's result set is used (see line 1). It dictates that for each repetition (iteration) the name attribute of the iterated image component is complemented with a unique postfix (in this case the 'id' field of the query). This explicit definition of the result set's unique identifier field is necessary since the component-based document format is not by definition associated with a given underlying data model. On the other hand, the flexibility of the template mechanism allows to refer to arbitrary data sources, and even to different ones within the same component-based

[12]The concept of component templates was realized for SQL-based queries on relational databases but can be easily extended for other data sources such as XML or RDF databases in a straightforward manner. In Section 5.3.2 it will be shown how component instances can be automatically generated based on RDF data.


```
1  <t_aco:AmaListComponent name="productimagelist"
2                          type="iterativeTemplate"
3                          query="SELECT id,source,width,height FROM
4                                 productimages"
5                          idField="id">
6     ...
7       <alay:LayoutManager>
8        <alay:BoxLayout axis="yAxis">
9          <t_alay:ComponentRef iterate="yes">
10             picture
11         </t_alay:ComponentRef>
12       </alay:BoxLayout>
13      </alay:LayoutManager>
14    ...
15    <aco:SubComponents>
16       <t_aco:AmaImageComponent name="productimage" iterate="yes">
17         <aco:MetaInformation>
18           <amet:source><t_temp:query field="source"/></amet:source>
19           <amet:width><t_temp:query field="width"/></amet:width>
20           <amet:height><t_temp:query field="height"/></amet:height>
21         </aco:MetaInformation>
22       </t_aco:AmaImageComponent>
23    </aco:SubComponents>
24 </t_aco:AmaListComponent>
```

Listing 4.11: Iterative component template example

document.

As the resulting content unit contains a set of image components, their spatial arrangement has to be specified, as well. However, since the number of these subcomponents is not known at authoring time, only the layout managers *BoxLayout* (with an initially undefined number of cells) and *GridTableLayout* (with only one predefined dimension) are allowed and the missing dimensions have to be automatically computed at run time when evaluating the template's query. In the particular example shown in Listing 4.11 a vertical *BoxLayout* is used.

At run time (see Section 4.5), component templates are dynamically filled with content according to their queries as well as the actual state of the corresponding request parameters. Since component templates might aggregate other component templates, this evaluation process is performed recursively and results in dynamically generated component instances, i.e. the "placeholders" in the original templates (component skeletons) are substituted by the actual query's specific results. Thus, after being evaluated, component templates can be treated in the same way as "conventional" static components. Furthermore, while the examples in Listings 4.10 and 4.11 contain select queries, it is also possible to define update queries in component templates. In this case these queries can be used to add (or manipulate) data to (in) a database.

Component templates provide an effective means for the creation of data-driven component-based Web presentations. For their intuitive creation and manipulation a visual authoring tool called the AMACONTBuilder will be introduced in Section 5.2.

4.5 Document Generation

The component-based document format allows to compose adaptive Web documents by creating, configuring, aggregating, and interlinking reusable components (or component templates) on different abstraction levels. When requested by a particular user, such document structures have to be automatically adjusted to his current usage context and delivered to his end device in an appropriate Web output format. For this purpose a modular document generation architecture was developed [Fiala et al. 2003a, Hinz and Fiala 2004, Hinz and Fiala 2005].

The document generation architecture serves several purposes: 1) the automatic translation of component-based documents to Web presentations according to the actual usage context, 2) the storage of this context information, 3) as well as its continual updating based on user's navigation and interaction history. While not being the central focus of this work, the rest of this section describes this functionality in more detail[13].

4.5.1 Pipeline-based Document Generation

As illustrated in the lower part of Figure 4.3, the process of document (presentation) generation is based on a stepwise pipeline concept[14]. According to the component-based document format introduced above, its inputs are complex document component instances or document component templates. They are retrieved from a component repository (or another source aimed at dynamically generating components) according to a user request that is optionally parameterized by a number of HTTP request parameters. Still unadapted, they encapsulate all variants concerning their content, layout, structure, and interlinking.

In the document generation pipeline document components are processed by a series of transformations. Each transformation deals with a given application (or adaptation) concern and produces output for the next transformation step until a final Web presentation is generated. Note that the possibility to use such a staged architecture is a natural consequence of the clean separation of concerns, a basic design principle of the component-based document format. While the modularity of the document generation pipeline allows for different transformer configurations, Figure 4.3 depicts a typical one.

First, possible component references (to components in other XML documents) are resolved and hierarchical component structures are created. Second, whenever the processed documents contain component templates, these are filled with instance data that is dynamically retrieved by the on-the-fly execution of their appropriate queries. Subsequently, the resulting component instances are subdued to a number of adaptation transformers. Parameterized by the current state of the context model (see Section 4.5.2), each of them considers a certain adaptation aspect by the selection, configuration, or device-specific rendering of component variants.

In Section 4.3 two mechanisms for specifying adaptation were mentioned: one for describing adaptation variants and another one for describing adaptive layout. Consequently, the adaptation of document instances is also performed in two main stages. First, a transformer aimed at processing components containing adaptation variants is invoked. It handles all appropriate selection methods in a recursive (top-down) manner and keeps only the selected

[13]The document generation architecture, its context modeling framework, as well as the investigation of its performance issues is a primary research focus and contribution of Michael Hinz. This section describes these topics as detailed as required to understand the overall context of the work presented in this thesis. For more information we refer to the corresponding publications.

[14]Note that this stepwise pipeline concept corresponds to the architectural style "staged architectures", proposed by Aßmann [Aßmann 2005] for active documents (see Section 3.2.9)

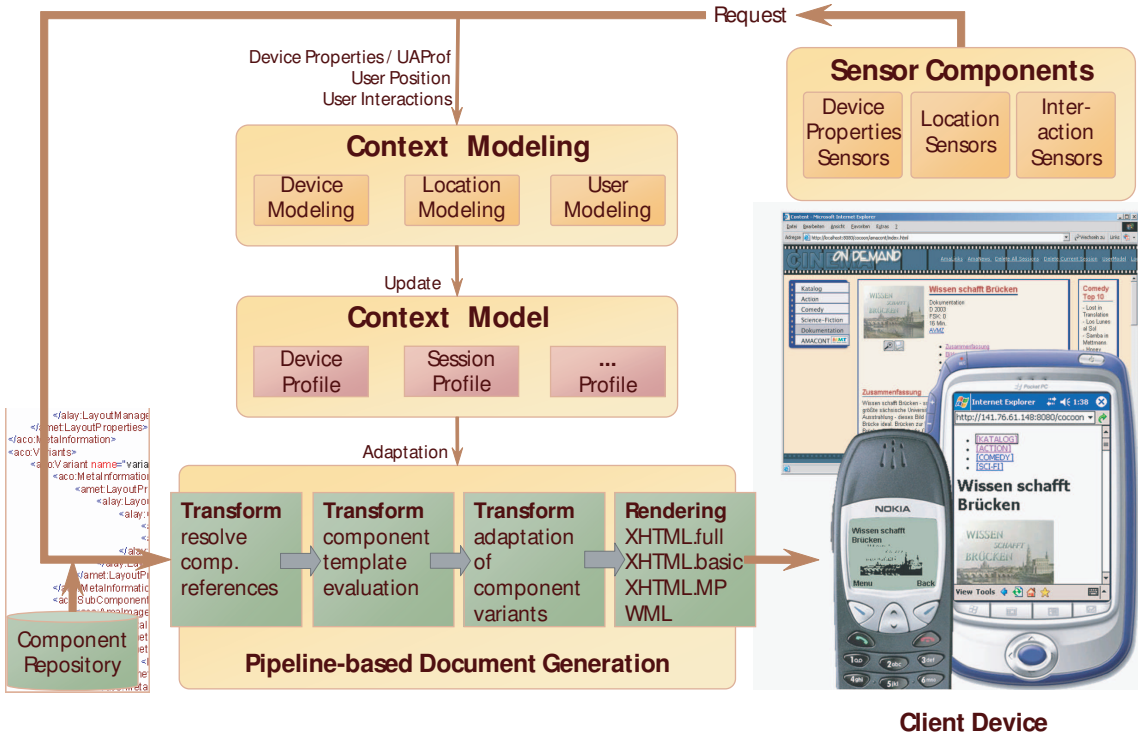

Figure 4.3: Overview of the document generation architecture

content variants in the processed XML stream, i.e. all other variants are omitted from the processed document. The result of this transformation step is a final component hierarchy without adaptation variants.

After all component variants and conditions are evaluated, the last transformer creates a Web page in an output format supported by the user's browser device. For instance, a BoxLayout in XHTML is realized by means of a table (and its specific attributes) with either one column or one row. However, not all layout managers can be visualized properly on all devices. As an example, since PDAs or WAP phones have very small displays, a horizontal BoxLayout is automatically converted to a vertical arrangement of subcomponents on those devices. This kind of adaptation is performed by the system, i.e. no explicit specification from the author is needed. The document generation pipeline was realized based on the Cocoon publishing framework [Ziegeler and Langham 2002].

4.5.2 The Context Model

The information describing the actual user and his usage context is stored in the extensible context model (see the middle part of Figure 4.3). It is represented in XML and consists of a set of context profiles[15], each maintaining up-to-date data on a given user/context feature. The structure of context profiles relies on CC/PP (Composite Capability/Preference Profiles [Klyne et al. 2003]), an RDF grammar for describing device capabilities and user preferences in a standardized way. Still, while the original CC/PP specification defines a

[15]While some literature uses the notion of a *user profile* or *context profile* for describing usage data that is static with regard to a Web session, the profiles of the context model may contain both static and dynamic information. The update process of context information will be described in Section 4.5.3.

profile as a flat two-level hierarchy of *components* and their *attributes* that are represented as (sets of) literals, it is allowed to utilize arbitrary deep XML structures for describing component attributes. As a general grammar, CC/PP makes no specific assumptions on concrete context characteristics. Therefore, for each profile a corresponding schema (e.g. expressed by an XML Schema or RDFS declaration) has to be provided.

Since the component-based document model is not bound to a specific application domain, the context information used in a given adaptation scenario is also typically very application specific. Therefore, the context model can be arbitrarily extended by the introduction of new context profiles, each specified by a given schema definition. Still, there are also some predefined profiles that can be generally used for a broad range of ubiquitous and context-aware adaptive Web presentations. The following list gives a representative overview of them.

- The **Identification Profile** provides basic personal information about the user, such as his name, login ID, email address, age, etc. This data is typically static with regard to a given browsing session and can be acquired by explicitly asking the user e.g. in the beginning of a Web session. For example, the login ID of the user is determined when he starts his browsing session. As a matter of course, the usage of this profile for adaptation purposes is optional.

- The **Device Profile** contains technical information describing the user's client device. It is represented on the basis of the WAP User Agent Profile (UAProf [Wir 2001]), a common CC/PP vocabulary aimed at describing WAP devices. However, to support a broader range of mobile devices (e.g. PDAs) specific extensions of UAProf have been made [Hinz et al. 2004].

  While parts of the Device Profile (such as the device's hardware platform) are static with regard to a Web session, note that it contains also parameters that can change according to user interactions. For example, when the user resizes his/her browser window, the appropriate context parameter should be updated, respectively. The acquisition and update process of context model parameters according to user interactions will be described in Section 4.5.3.

- The **Location Profile** stores the physical (geographical) location of the user. This information is described both by means of physical coordinates as well as semantic location information describing the semantic meaning of a location (e.g. the Multimedia Chair of the Dresden University of Technology) and is retrieved from a landmark store. For more information on the appropriate location context descriptors the reader is referred to [Hinz and Fiala 2005].

- The **Session Profile** collects information on the actual user's browsing and interaction history within a component-based Web presentation. Its main goal is to track user access information, such as the number of the user's previous sessions in a given application, the unique identifiers of the document components he already visited as well as the interactions he performed on selected media components (e.g. starting a video, enlarging an image component, etc.). This profile is automatically updated at each user request and is an important basis for the utilization of user modeling mechanisms facilitating dynamic adaptation (adaptivity).

The example code in Listing 4.12 illustrates a CC/PP-based context model description. Note that while in this case only a small excerpt from the Identification Profile and the Device Profile is shown, the actual context description might contain an arbitrary number


```
<ContextModel>
  ...
  <IdentificationProfile>
    <ccpp:component>
      <UserData>
        <ID>fiala</ID>
        <Title>Mr.</Title>
        <Firstname>Zoltan</Firstname>
        <Lastname>Fiala</Lastname>
        <Age>29</Age>
      </UserData>
    </ccpp:component>
    ...
  <IdentificationProfile>
  ...
  <DeviceProfile>
    <ccpp:component>
      <HardwarePlatform>
        <ColorCapable>Yes</ColorCapable>
        <TextInputCapable>Yes</TextInputCapable>
        <ImageCapable>Yes</ImageCapable>
          ...
      </HardwarePlatform>
    </ccpp:component>
    ...
    <ccpp:component>
      <SoftwarePlatform>
        <CcppAccept-Language>de</CcppAccept-Language>
          ...
      </SoftwarePlatform>
    </ccpp:component>
      ...
  </DeviceProfile>
  ...
</ContextModel>
```

Listing 4.12: Extract from an example context model

of profiles. For more detailed information on the utilized context model and its profiles the reader is referred to [Hinz et al. 2004, Hinz and Fiala 2005].

4.5.3 Support for Context Modeling and Interaction Processing

The pipeline-based document generation process supports adaptability (or static adaptation) by adjusting complex document structures to available information describing the actual usage context. However, to facilitate adaptivity (i.e. dynamic adaptation based on user's browsing behavior), there is a need for additional mechanisms. First, user interactions (or other external events, such as bandwidth fluctuations) have to be acquired that might influence specific parts of the usage context and thus lead to a dynamic reconsideration of the presentation. Second, the context model has to be updated based on this acquired information, respectively.

To facilitate these mechanisms the document generation architecture provides a context modeling framework that allows to utilize an extensible set of sensor components and context

modeling components [Hinz et al. 2006] (see the upper part of Figure 4.3). Whereas the sensor components aim at acquiring user interactions (such as following links, or interacting document with components) device capabilities (e.g. information on the user's client device type or current browser window size) or other kinds of context information (e.g. location), the context modeling components perform updates of the context model according to this information on the server side. As a matter of course, the usage of a given context modeling (or user modeling) strategy is typically strongly dependent on the given application scenario. Still, the document generation architecture provides a number of generic facilities that can be efficiently used for acquiring and processing interactions in a broad range of component-based adaptive Web presentations.

4.5.3.1 Acquiring User Access Information and Device Capabilities

Whenever a user follows a link or submits a form in the generated Web presentation, the request sent to the server contains standard HTTP request information (in form of request parameters). Furthermore, the document generation architecture allows to automatically extend this information by more detailed data gathered both on the user's interactions as well as his actual context (e.g. device capabilities and location information).

In order to gather user access in a component-based adaptive Web presentation, component authors can configure selected components as "observed" by setting their *watched* attribute to *true* at authoring time. Whenever such an observed component is included in the resulting Web presentation (i.e. if it is not omitted by a certain selection method), the generated Web page is automatically enriched with sensor components based on client side code fragments aimed at tracking user interactions on those components. The fact that a component was presented on the user's browser is tracked as a "trivial" interaction with that component (meaning the user saw that component). However, some media components allow for more "complex" interactions (such as starting a video, enlarging an image, etc.).

When requesting another component-based Web document, the identifiers of the observed components as well as the interactions performed on them are automatically sent to the server side where the session profile is updated, respectively. Note that the utilization of such user access information for adaptation purposes is a basic facility in adaptive hypermedia and Web-based systems and was successfully applied in different application scenarios [Jörding 1999, De Bra et al. 2002, Casteleyn 2005].

The acquisition of client device capabilities happens in a similar way as the acquisition of user interactions, i.e. by the insertion of device capability sensors in form of client-side code fragments. Again, the appropriate code fragments are automatically inserted into the generated Web presentation and collect up-to-date information about the user's browser device (such as its type, supported media types, and plug-ins, current browser window size, etc.) and location. The gathered information is encoded in a UAProf like representation and integrated in the HTTP request by a client/server communication component for processing that information on the server. For more information on the technical realization of these mechanisms the reader is referred to [Hinz and Fiala 2005].

4.5.3.2 Context Modeling

As described above, the HTTP requests originating from the client contain besides standard request parameters additional information describing user interactions, device characteristics, etc. Before the next hypermedia page is generated, this information has to be processed

and the context model updated, respectively. For this purpose the document generation architecture allows utilizing an extensible set of context modeling components. Providing a well-defined interface for accessing the request parameters (incl. the information delivered by the sensor components) and manipulating the context model, these components can be programmed or configured to perform arbitrary context model updates based on the newly acquired information. Furthermore, they can also be used to implement server-side application logic, such as performing a database query, invoking a Web service, etc.

The currently existing repertoire of context modeling components comprises modules for device modeling (aimed at updating the device profile), location modeling (for storing the exact geographical position of the user in the context model) as well as a number of solutions for user modeling [Hinz and Fiala 2005] that can be configured and activated depending on the current application scenario. As an example, in a prototypical component-based Web presentation for an online video store a user modeling algorithm based on the incremental learning algorithm CDL4 [Shen 1996, Hinz et al. 2004] was successfully utilized, allowing to predict user's preferences based on their interactions with media objects. Nevertheless, in order to support a broad number of context modeling mechanisms, it is also possible to implement and easily integrate new context modeling components.

When the process of context modeling was performed a new component-based document is retrieved and put through the presentation generation pipeline, respectively. This might be an already existing component-based Web document (or document template) from the component repository, but it is also possible to redirect the request to a backend application that dynamically generates such a document. Thus, a component-based Web presentation can be also effectively used as the adaptive front-end for a more complex back-end application. Such a scenario will be described in more detail in Section 5.3.

4.6 Summary and Model Benefits

This chapter a presented concern-oriented component model for dynamic adaptive Web documents. The concept of declarative document components was introduced and a corresponding XML-based component-description language was presented. The different component layers addressing both different application concerns and adaptation facilities were explained by a number of examples. Furthermore, a pipeline-based document generation architecture for the on-the-fly publishing of component-based Web presentations was also briefly described.

Before turning to the authoring process of component-based adaptive Web presentations and its tool support in Chapter 5, the rest of this section summarizes selected important aspects and characteristics of the document model. First, Section 4.6.1 describes its main analogies and differences to the already presented hypermedia reference models Dexter and AHAM. Then, Sections 4.6.2 to 4.6.5 recapitulate a selection of its main benefits, among them component reuse and configurability, adaptation support, extensibility, as well as support for Web annotations[16].

[16]In Section 3.2.10, a number of requirements towards component models for adaptive Web applications were mentioned, which also served as the basis for the design of our own model. Note, however, that this section recapitulates only a selection of those aspects. Other issues (e.g. the separation of concerns, device independence, or template support) were already in detail discussed throughout this chapter and thus do not need further emphasis.

4.6.1 The Component Model vs. Dexter and AHAM

In Chapter 2, the two reference models Dexter and AHAM were introduced to capture the main characteristics of (adaptive) hypermedia systems. Since the concern-oriented component model presented in this chapter is an approach aimed at implementing adaptive Web applications, this section summarizes its main analogies and differences to those reference models.

First, we note that the concept of document components corresponds to the component concept of Dexter. Nevertheless, by the introduction of different component layers, our model provides a more explicit typing as well as a fine-grained consideration of different concerns involved in a Web or hypermedia application. Furthermore, whereas Dexter keeps the Within-Component layer unspecified, the media component layer of our model specifies in detail the supported atomic content elements and the specification of their attributes. Moreover, while Dexter considers components to be static with regard to their content (and is thus mainly applicable for static hypermedia presentations), the concept of component templates (in our model) supports even data-intensive applications.

Another similarity of our model to Dexter (and AHAM) is the consideration of hyperlinks as components. However, in contrast to both reference models, hyperlinks with several end points and/or bidirectional references are not supported. The reason for these restrictions is the goal to explicitly consider the specific characteristics of the World Wide Web as a hypermedia system (see Section 2.1.3).

The concept of abstract layout descriptions (layout managers) corresponds to Dexter's presentation specifications. Yet, instead of being separated from the actual components (e.g. as part of a specific Run-Time layer), it is one of their inherent properties.

Finally, similar to the Teaching Model of AHAM, the concern-oriented component model also sets a great store by supporting adaptation. However, whereas the adaptation rules of AHAM are stored as separate entities, the concern-oriented component model considers them as parts of components allowing for their adaptation in a component-based manner. Moreover, whereas AHAM (and its reference implementation AHA!) mainly focus on the conditional inclusion/exclusion or the annotation of components, our model allows to implement a broader range of adaptation techniques. As will be described in Section 4.6.4, the combination of component templates, abstract layout descriptions, and their conditional variants allows to implement most of the adaptation techniques introduced by Brusilovsky (see again Section 2.2).

4.6.2 Support for Component Reuse and Configurability

A very important aspect of effectively engineering Web sites is the reuse of formerly developed artefacts. However, the current coarse-grained document-oriented implementation model of the Web makes it difficult for authors to identify and efficiently reuse configurable content fragments of a Web presentation [Gaedke et al. 2000]. The component-based document model presented in this chapter tries to solve this problem by defining fine-granular Web components for creating Web applications. By encapsulating their properties and functionality in a component-wise manner, they can be easily reconfigured and thus be utilized in different application scenarios.

The level-based structure of the document model supports the effective reuse of components of a certain level in components on higher levels. For instance, an adaptive image component being capable to adjust itself to the current screen size can be reused as a "black-box" in different content units. Similarly, a dynamic content unit arranging a list of pictures

in a tabular way might be easily reused by being filled with different image components. Furthermore, the recursive nature of document components facilitates the reuse of component trees of arbitrary depth and granularity. Since each component encapsulates its structure (subcomponents and links), presentation (layout managers), and adaptation behavior (in form of selection methods) in an inherent way, all this functionality is automatically "carried with" when applying the component in another composition scenario.

As a matter of course, a crucial issue of the efficient reuse of Web implementation artefacts is their ease of (re)configuration. Therefore, the description language of document components provides a clear separation of independent concerns (content, structure, presentation, navigation, adaptation) by utilizing separate descriptors (metadata interfaces) for configuring all these different issues. Whereas e.g. in an HTML document all these aspects are "interwoven" to a coarse-grained file-based resource, the component-based document model enables component authors to configure or manipulate them separately. As an example, a component developer might define two different layout variants for a component (e.g. one for a desktop PC and another one for a handheld device) without influencing its content or interlinking.

4.6.3 Extensibility Support

As described in Section 4.2, the component-oriented document model rests upon a level-based architecture. That is to say, all possible components are derived from the basic (abstract) component types *media component*, *content unit component*, *document component*, and *hyperlink component*. Each of these abstract types has a number of predefined concrete derived types (e.g. in the case of media components these are image component, text component, structured text component, audio component, etc.). However, it is also easily possible to introduce new component types on each abstraction level.

As an example, the introduction of a new media component type requires the extension of the schema definition AmaComponent.xsd with a new component type (that inherits from the abstract type AmaMediaComponent based on the substitution group mechanism of XML Schema [Fallside and Walmsley 2004]) and the specification of its metadata properties in the schema definition AmaMetaData.xsd. Furthermore, the existing layout stylesheets for appropriately transforming the new media component descriptions into a given Web output format (e.g. HTML, cHTML, WML) have to be adjusted, accordingly. All other functionality (e.g. the ability to define adaptation variants, layouts, hyperlinks) is defined for the abstract component definitions, i.e. it can be utilized by each instance of the new component type, as well.

Besides component types, it is also easy to extend the component description languages with new layout managers and adaptation logics. For instance, the definition of a new layout manager type implies the extension of the schema definition AmaLayout.xsd by declaring its *layout attributes* and *subcomponent attributes* (see Section 4.3.2). Furthermore, the appropriate stylesheets for transforming abstract layout descriptions to a given output format have to be extended. However, this concerns only the rendering of format specific container elements (e.g. tables, lists, etc.) aimed at the presentation of the actual content represented by the embedded media objects, the rendering of media components is not depending of the actual layout manager.

4.6.4 Adaptation Support

As discussed in Section 4.3, the component-based document format supports two basic adaptation facilities: the possibility to define component alternatives (on different component

levels), as well as to describe the layout of components in an abstract and implementation independent way. Though being simple adaptation mechanisms, note that these facilities can be effectively utilized to realize a number of adaptation techniques. According to the already mentioned taxonomies of Brusilovsky [Brusilovsky 1996, Brusilovsky 2001] as well as Paterno and Mancini [Paterno and Mancini 1999], the following lists comprise the supported content-level, link-level, and presentation-level adaptation facilities and their component-based realization:

Content-Level Adaptation

- **Support for page variants and fragment variants** by the definition of content alternatives on different abstraction levels
- **Conditional inclusion of fragments** by defining conditional variants for sub-components
- **Adaptation of media content** by offering media components with quality alternatives
- **Adaptation of modality** by providing content units with varying types of included media elements

Link-level Adaptation

- **Link Disabling** based on the conditional inclusion/exclusion of hyperlink components
- **Link Removal** by the conditional inclusion of both hyperlink components and the media components serving as their anchors
- **Link Annotation** based on the conditional assignment of style classes to hyperlink components
- **Link Hiding** based on the conditional assignment of style classes to hyperlink components
- **Link Generation** by the dynamic inclusion of hyperlink components based on context-dependent component templates
- **Link Sorting** by the dynamic inclusion and ordering of hyperlink components based on context-dependent component templates

Presentation-Level Adaptation

- **Support for layout variants** based on conditional alternative layout manager definitions
- **Adaptive styling of pages and page fragments** by utilizing alternative CSS components

As can be seen, the adaptation support provided by our model go far beyond the capabilities of related component-based and document-oriented approaches (see Section 3.2.10). Furthermore, as was described in Section 4.5, it also supports the automatic generation of Web presentations in different output formats, among them (X)HTML, cHTML, or WML.

4.6.5 Support for Web Annotations

Annotating Web pages is an important aspect of asynchronous communication on the WWW. Authors and visitors of Web applications attach notes to certain pieces of Web content in order to remember things better, to communicate with each other or to manage information more intelligently. Typical scenarios of Web annotations are Web-based learning systems allowing students and tutors to communicate, distributed authoring environments supporting the concurrent editing of content, and even product presentations, where users give feedback to the system via personal remarks.

Existing annotation systems, like ComMentor [Röscheisen et al. 1995], CritLink [Yee 1998], CoNote [Davis and Huttenlocher 1995], YAWAS [Denoue 1999], iMarkup [@imarkup], Annotator [Ovsiannikov et al. 2000], WebWise [Grønbæk et al. 1999], etc. mainly focus on annotating static Web pages which do not change their content, structure, and layout temporally. In general, an annotation is clearly defined by the URL of the Web page containing it and some anchor points within that page [Denoue and Vignollet 2000]. A significant disadvantage of these tools is the lacking support for separation of content, structure, and layout. Annotations are not attached to the contents itself, rather to the Web pages containing them. Notes go lost, when the same content is presented on a different page, in a new context or with a changed layout. Thus, this mechanism is not suitable for dynamic Web documents generated at runtime for which no persistent state and no constant layout exists.

Abstracting from the coarse-grained model of current Web implementation languages, the proposed document model and its document generation architecture support the creation of annotations to reusable components. That is to say, the smallest objects to be annotated are not whole Web pages but document fragments, i.e. media components, content units or document components. When a user marks a Web page generated on the basis of Web components, his annotations can be reversely mapped to the fine-granular content components and stored as specific component metadata. This reverse mapping is supported by automatically enriching the generated Web page source code (e.g. HTML) by appropriate semantic markup and client-side Java script code fragments. Annotation anchors are unequivocally located by the identifier of the corresponding component and some offset coordinates within it. When the same component is used in another presentation - possibly on a different client or in a different context - the attached annotations can be reused, too.

While not being a central issue of this thesis, we note that concept of attaching fine-granular annotations to reusable, adaptive components was prototypically implemented in an annotation system called DynamicMarks. For more detailed information on it the reader is referred to [Fiala and Meissner 2003].

Chapter 5

The Authoring Process and its Tool Support

> *"We shall not fail or falter; we shall not weaken or tire. Give us the tools and we will finish the job."*[1]

The document model introduced in Chapter 4 allows to create adaptive Web presentations from reusable implementation artefacts (components) that encapsulate adaptable content, navigation, and layout on different abstraction levels. Still, even though component-based reuse is crucial to Web Engineering, the development of adaptive Web applications of such components is typically a complex task that requires systematic process models and appropriate tool support [Fiala et al. 2004b]. Therefore, this chapter deals with the authoring process of component-based adaptive Web presentations and its tool support[2].

Independent of a given application domain, the proposed document model supports different Web application scenarios. Consequently, the resulting authoring process should not be bound to a fixed process model or workflow, rather adjusted to the specific requirements of the targeted application area. These requirements may vary depending on various factors, such as the application's type (e.g. static adaptive hypermedia presentation vs. data-driven dynamic Web application), its targeted user group, size, complexity, etc. As a trivial example, consider the case of a Web author aimed at the rapid development of a small set of Web pages presenting static content on different end devices. He could be best suited by an ad-hoc authoring process allowing to visually create adaptable content components and "plug them together" to a set of Web pages. On the contrary, the development of a dynamic Web Information System providing different features of adaptive navigation and presentation is a significantly more complex Web engineering task. It should be based on a systematic process model that considers separate concerns of the planned application (data, navigation, presentation, personalization, device and context dependency, etc.) in a structured way. Thus, to facilitate different development scenarios, there is a need for flexible authoring support.

The first part of this chapter (Section 5.1) deals with the structured authoring process of component-based adaptive Web presentations. However, instead of suggesting an own methodology, the chosen strategy is the adoption of existing hypermedia design methods for this purpose. The main reason behind this approach is the observation that, given the abstraction gap between high-level hypermedia design models and low-level implementation entities (document components), even different methodologies can be utilized to develop component-based adaptive Web applications [Fiala et al. 2004b]. This thesis focuses on an

[1] Winston Churchill (1874-1965)

[2] As discussed in Section 3.1, the overall life-cycle of adaptive Web applications encompasses different activities, such as requirements engineering, design, implementation, testing, or maintenance. While the author is aware of the importance of all related activities, the focus of this dissertation (and thus this chapter) is on the model-based design and component-based implementation of adaptive Web sites.

important development scenario: the engineering of data-driven Adaptive Web Information Systems (AWIS) from reusable components. Therefore, it adopts and extends the model-based Hera Web design method [Vdovjak et al. 2003] to the context of component-based Web engineering. The resulting design methodology and engineering process is called Hera-AMACONT and supports the structured development of adaptive Web information systems from reusable components. Considering the steps identified by the Hera design models as a guideline, it is shown how component authors can systematically create, configure, aggregate, and link document components (and templates) to complex adaptive Web presentations. It is illustrated how different design issues concerning data, navigation, presentation, as well as their related adaptation concerns can be taken into account at implementation in a structured way. Thus, a possible model-based authoring process is proposed for the developers of component-based adaptive Web presentations.

In order to efficiently put a given design or authoring process (such as the one dictated by Hera-AMACONT) into practice, component authors need appropriate tools for creating, configuring and aggregating components. To this end, the second part of this chapter (Section 5.2) introduces the AMACONTBuilder, a modular authoring tool for the developers of component-based Web applications. Based on an extensible set of graphical editor modules, it allows to visually create, configure, and aggregate adaptive document components on different abstraction levels. Moreover, it also facilitates the creation of component templates, thus allowing to author data-driven adaptive Web presentations. Independent of a specific methodology, it is shown how it can facilitate different authoring workflows. Finally, selected implementation and extensibility issues are also briefly presented.

While the AMACONTBuilder facilitates flexible component authoring (implementation) independent of a specific design method, in some cases it is desirable to take a further step from model-based component authoring to model-driven component generation and to add automation to the overall process of design and component-based implementation. Therefore, the third part of this chapter (Section 5.3) deals with the research question of how a component-based implementation can be automatically generated on basis of a high-level design specification in a model-driven way. Again, this is illustrated by example of the Hera-AMACONT methodology. After identifying main automation requirements, an RDF(S)-based formalization of the presentation design phase of Hera-AMACONT is provided. According to this formalization, high-level model specifications can be automatically mapped to a component-based implementation, thus exploiting its flexible presentation and adaptation capabilities. The resulting multi-stage development process and document generation architecture are described in detail and exemplified by a prototype application.

Finally, Section 5.4 summarizes the resulting multi-stage Web engineering process and provides a representative overview of already realized component-based adaptive Web applications.

5.1 Hera-AMACONT: Model-based Component Development based on a Hypermedia Design Method

In recent years, different methodologies facilitating the structured design of complex Web applications have been developed. A detailed overview of the most significant existing approaches was given in Chapter 3. As discussed there, most of them distinguish between the conceptual design, the navigational design, and the presentation design of a Web application. Furthermore, some of them even explicitly address selected issues of personalization and adaptation.

This section adopts the model-based Hera design method [Vdovjak et al. 2003] to the context of component-based Web engineering. The resulting design methodology and engineering process is denoted as Hera-AMACONT and supports the structured development of data-driven adaptive Web applications (e.g. online-shops, e-galleries, etc.) from reusable implementation artefacts. Note that Hera is suitable for this undertaking for different reasons. First, its main focus lies on the specification of different kinds of adaptation in a Web Information System [Frasincar et al. 2002]. Besides aspects of adaptability (static adaptation), issues of adaptivity (dynamic adaptation) are also concerned [Frasincar 2005]. Second, Hera uses Semantic Web technologies (RDF and RDFS) to explicitly formalize model descriptions. Such a Semantic Web-based approach has a number of benefits: a more explicit description of model semantics, better interoperability and (possibly) model verification support, as well as the possibility to integrate existing ontologies. Furthermore, due to the usage of XML-based models, an automatic translation of high-level Hera design models to a component-based implementation appears to be also possible. Finally, in contrast to several other methodologies, Hera foresees to specify the presentation aspects (layout, look-and-feel) of a Web application at model level.

Therefore, based on a small running example, different phases of designing and implementing component-based adaptive Web presentations are described. Considering the steps identified by the (extended) Hera design models as a guideline, it is shown how component authors can apply those concepts to systematically develop adaptive Web presentations out of reusable document components. The main focus is on the question of how different adaptation issues (both static and dynamic) can be targeted in each design and implementation step.

5.1.1 Conceptual Design

The first step of the Hera design method is the so-called *conceptual design* aimed at representing the application domain using conventional conceptual modeling techniques. It results in the *conceptual model* (CM) consisting of a hierarchy of concepts, their attributes, and relationships. A concept represents a certain entity in a particular application domain. It is further characterized by concept attributes, each being typed. Besides basic types (e.g. `Integer` and `String`), multimedia types (e.g. `Image`, `Audio`, `Video`) are also allowed, thus enabling to assign representative media items to concept attributes. The CM can be expressed both graphically and using RDFS [Brickley and Guha 2003]. For the graphical specification of conceptual models, Hera provides appropriate modeling tools [Frasincar 2005].

The example application used throughout this chapter is a (small part of a) Web Information System providing information on painters, their paintings, and painting techniques [Fiala et al. 2004b, @ICWE2004Demo]. An excerpt of its underlying conceptual model is depicted in Figure 5.1.

The concepts constituting the application domain are illustrated as dark ellipses. They contain concept attributes denoted as light ellipses. As an example, the concept *technique* (representing painting techniques) has two attributes: a *name* and a *description*. Concepts are related to each other by typed concept relationships. For instance, a painting *technique* is associated to a set of *paintings*, all inheriting from the concept *artifact*. This is an 1:n relationship of the type *exemplified_by*. A painting (since inheriting from the concept artifact) is characterized by its *name*, the *year* of its creation, and a corresponding *picture*. Furthermore, via the relationship *painted_by*, *paintings* are associated with *painters* that again inherit from the abstract class *creator*.

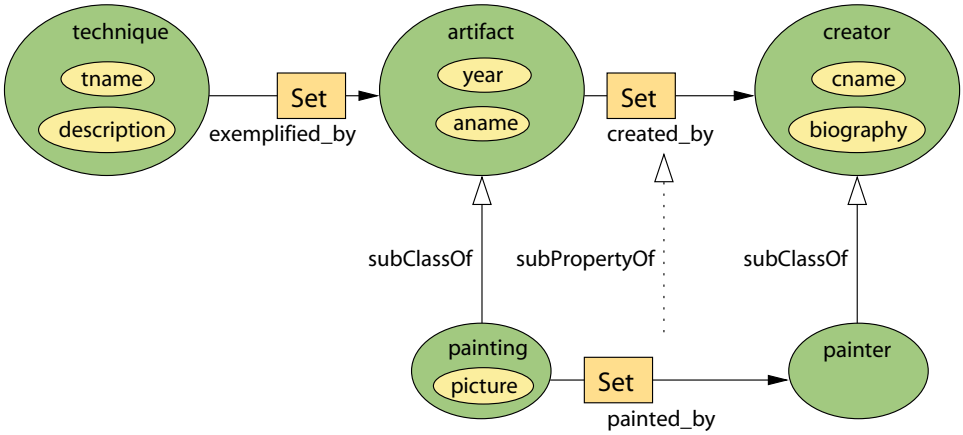

Figure 5.1: CM example [Fiala et al. 2004a]

The media types associated to the concept attributes are described in the *Media Model* (MM), a submodel of CM [Fiala et al. 2004a][3]. It is a hierarchical model composed of media types. The most basic media types are: Text, Image, Audio, and Video. Figure 5.2 shows an excerpt of the MM for the running example. The media types are depicted in dark rectangles.

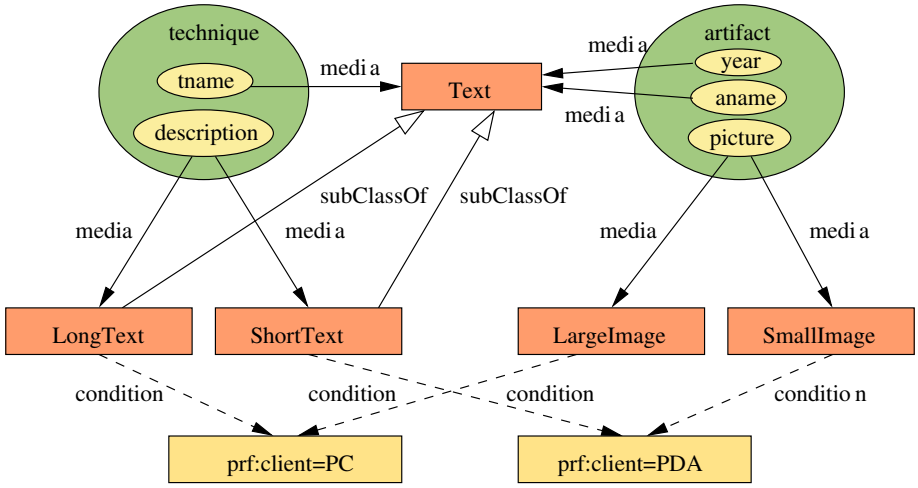

Figure 5.2: MM example [Fiala et al. 2004a]

5.1.1.1 Adaptation at Conceptual Design

The Media Model aims not only at the assignment of media types to concept attributes, it also allows to define media adaptations. These are based on simple Boolean expressions (depicted as light rectangles in Figure 5.2) that reference attributes from the current usage context and dictate the conditional usage of different media types.

In the running example, two conditions addressing the limited screen size constraints of mobile client devices are used. One condition requires to use a long text for the technique

[3]Frasincar also refers to the Media Model as the so-called *media vocabulary* [Frasincar 2005].

description and a large image for the artifact pictures on PCs. The other condition stipulates that a short text for the technique description and a small image for the artifact pictures should be used for PDAs. While in this particular example both conditions specify adaptability (i.e. adaptation based on context parameters that are typically static with regard to a Web session), note that media adaptations can be performed dynamically, as well. As an example, the designer could specify different quality alternatives for a video depending on the currently available bandwidth. Thus, in case of possible bandwidth fluctuations (e.g. in a mobile environment), the usage of a given alternative could be dynamically reconsidered even during a single browser session.

We note, however, that this adaptation of the media representation of concept attributes concerns not only the data presented by a Web application (which is the actual focus of conceptual design) but also its presentation. The reason for this is the fact that a media object (e.g. an image) represents not only content but also inevitably presentation. As a consequence, the media model of a Web application might be reconsidered or extended after specifying its presentation design (in a later design step, see Section 5.1.5). Therefore, we claim that the design of a Web application might be an iterative approach, allowing to refine the separate design models in several turns.

5.1.2 Realization with Document Components

When developing adaptive Web presentations from document components, the conceptual design step as proposed by Hera has to be accompanied by the creation or retrieval of media instances that represent the identified concept attributes. These media instances (as well as the metadata attributes required by the component-based document format for describing their media properties) have to be stored in a structured data store so that they can be dynamically presented in the resulting adaptive Web application. The structure of this data source is to be derived from the CM, respectively. Furthermore, the appropriate media component templates facilitating the dynamic presentation of the created media instances have to be created.

In order to realize different media adaptations, component authors also have to reason about alternative media instances with different quality (e.g. concerning their formats, bandwidth, color depth, bit rate, size, etc.). According to Section 4.3.1, these alternatives have to be defined as media component (template) variants with their corresponding selection methods. For instance, a media component representing a painting's picture should have two variants, one for desktop devices and another one for handhelds.

5.1.3 Application Design

The application design step of Hera is the most important design phase dealing with the logical, structural, and navigational aspects of a Web application. Similar to the navigational models of other methodologies, its main goal is the specification of the overall hypermedia structure of the resulting application, i.e. the design of navigational units (hypermedia nodes and pages), their relationships (aggregation and interlinking), as well as corresponding adaptation issues[4].

[4]Recently, Hera's application model was extended by mechanisms for the specification of form-based user interactions [Frasincar 2005]. Still, the concepts described here focus on the basic Hera models, as also published in [Fiala et al. 2004b, Fiala et al. 2004a]. It is claimed that the mentioned extensions are adoptable for the context of component-based adaptation engineering, which is subject to ongoing cooperation.

In order to model navigational units, Hera uses the notion of *slices*. As in the case of RMM, a *slice* is a meaningful presentation unit that fulfills a certain communication purpose [Isakowitz et al. 1995], i.e. it represents an abstract view over the content described in the conceptual model that should be shown on a hypermedia node.

A slice is always associated with its owner concept which denotes the data (i.e. the concept from the CM) portrayed by it. Furthermore, there are two types of slice relationships: *slice navigation* (a hyperlink abstraction between two slices) and *slice aggregation* (a slice including another slice). An aggregation relationship between two slices with different owner concepts needs to specify the concept relationship between those concepts from the CM that made such an embedding possible. In the case that the cardinality of this concept relationship is one-to-many, the *Set* construct needs to be used. The most primitive slices represent concept attributes and are also referred to as *simple slices*. Slices that aggregate other slices are called *complex slices*. The most complex ones (called *top-level slices*) correspond to pages, which contain all the information presented on the user's display at a particular moment. The creation of AMs is provided by graphical tool support. For a more thorough introduction to Hera's application model vocabulary the reader is referred to [Frasincar 2005].

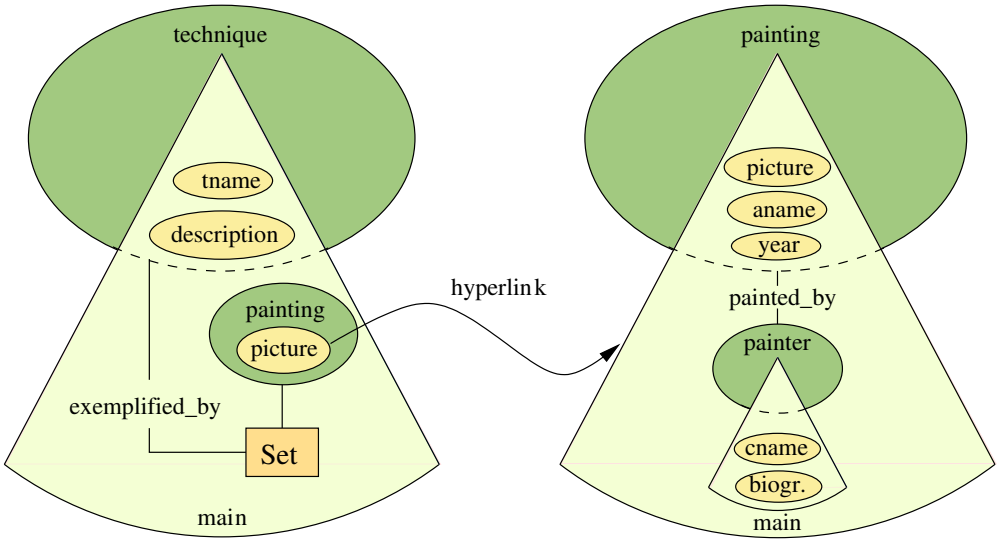

Figure 5.3: AM example

Figure 5.3 depicts (a small excerpt of) the application model of the running example in a graphical way. It consists of two top-level slices, one presenting painting *techniques*, the other *paintings*. As indicated by the underlying dark ellipse, the left top-level slice is associated with the concept *technique*. It shows two attributes of that concept: its name (*tname*) and description (*description*). Furthermore, it also contains a link list pointing to the *paintings* exemplifying the given painting technique. As depicted in the picture, these links are based on the concept relationship *exemplified_by*. The starting anchor of each link is represented by the *picture* attribute of the referenced *painting* concept. Since there are several paintings associated with a technique, a Set construct is used. When following a link, the user can navigate to the corresponding *painting* slice, a composite (top-level) slice presenting the concept *painting*. This slice presents the actual painting's *picture*, its *name*, the *year* when it was painted, as well as (from the bottom slice) some information about its *painter*. Note that besides the graphical representation (called the Application Diagram) there exists also an RDFS-based formalization of the AM [Frasincar et al. 2002].

5.1.3.1 Adaptation at Application Design

Adaptation at application design concerns the adjustment of the Web presentation's logical and navigational structure to the user and his usage context. Generally, different adaptation issues can be considered in the AM. First, it is meaningful to adjust the coarse navigational structure to varying device capabilities (e.g. desktop computer, PDA, cell phone, etc.) or user profiles (preferences, interests, knowledge). Depending on this information the designer can decide which concepts should be presented at all and how they should be assigned to different interlinked slices. Second, the population of each specified slice with concept attributes (or media types) can be adjusted, too. According to the preferences and/or used devices of different users, different media types for presenting the same concept can be utilized. As an example, take the case of two visitors, one of them preferring multimedia content, the other rather textual information. When presenting a painter's biography, the first one could be shown a video and an audio sequence, the second one a detailed textual description. Furthermore, dynamic adaptation (adaptivity) can also be targeted at this step. For example, different versions of a painter's biography could be presented in accordance with the user's changing knowledge on that painter: a long version at the user's first visit and a short one at his later visits. As a matter of course, these are only possible adaptation examples, the consideration of a certain adaptation concern (device dependency, personalization, security, etc.) depends on the designer's choice.

In order to specify adaptation, Hera prescribes that one associates so-called *appearance conditions* to slices [Frasincar et al. 2002, Frasincar 2005]. These are Boolean conditions using attribute-value pairs from the current usage context. Two kinds of AM adaptation are enabled: conditional inclusion of slices and link hiding. Conditional inclusion means that a slice is included (and therefore visible) when it has a valid condition. Similarly, link hiding refers to the mechanism that a link is only included when its destination slice is valid.

Figure 5.4 depicts another version of the application model shown above which is enriched by adaptation definitions. Note that it contains three appearance conditions. The first one (mentioning ExpertiseLevel) supports adaptability by including the painting technique's *description* only for Experts. The second one (mentioning imageCapable) refers to the device profile and dictates that the pictures of paintings should be only shown on devices being capable of presenting images. The third one (mentioning biography) defines adaptivity by presenting different versions of a painter's biography depending on the user's knowledge on that painter.

5.1.4 Realization with Document Components

There are important analogies between (the concepts of) Hera slices and adaptive document components. Both represent meaningful presentation units bearing also some semantic role (e.g. painting, painting technique, newspaper article) and are recursive structures enabling an arbitrary deep hierarchy. Moreover, both top-level slices and top-level document components correspond to hypermedia pages to be presented on the user's display. Furthermore, both may contain adaptation issues according to context model parameters.

Nonetheless, there are also significant differences. First, in contrast to slices, document components also contain information describing their layout. However, as application design concentrates on the navigation and does not deal with presentation issues, the specification of these layout properties can (and should) be postponed to a later stage of the development process (see Section 5.16). Second, whereas AM slices define the structure of presented concepts on the schema level (i.e. independent of concrete instances of those concepts), document

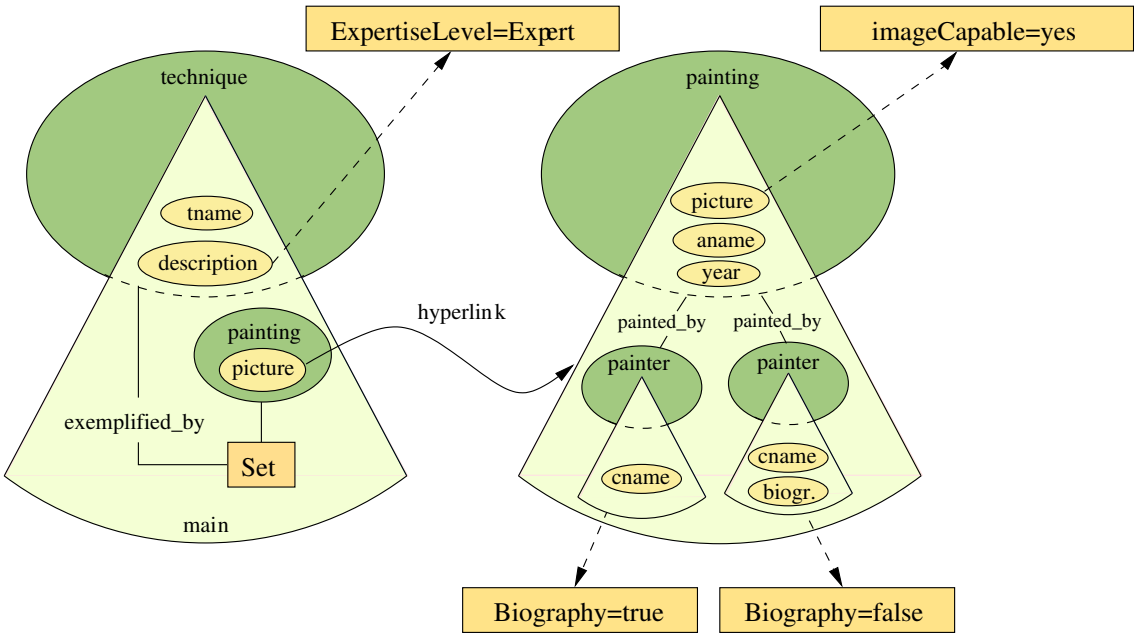

Figure 5.4: AM example with appearance conditions

components represent reusable implementation entities on the instance level. Still, note that this gap can be bridged by considering the notion of component templates. As described in Section 4.4 in detail, these are component skeletons declaring the structural, behavioral, and layout aspects of components independent of their concrete content. For example, a component author might create a component template for presenting dynamic information on painters. Such a template can be instantiated for specific painters by dynamically querying the Web application's underlying data source, which was created accompanying its conceptual design.

The mentioned analogies allow component authors to specify the aggregation hierarchy of component templates in accordance to a given AM design. First, top-level slices have to be mapped to top-level document component templates. Second, by unfolding slice aggregation relationships in a top-down manner, subslices have to be mapped to "sub document components" (or templates). In the case of simple slices, the media items (components) representing the concept attributes associated with those slices have to be additionally considered. Furthermore, both Set structures (of simple and composite slices) as well as slice navigation relationships (e.g. link lists) have to be taken into account. While there are different possibilities to map slice hierarchies to component template structures, we mention a straightforward and easily automatable one:

1. A complex slice that contains other (complex or simple) subslices should be mapped to a document component template. For its aggregated subslices, this mapping process should be performed recursively. As already mentioned, top-level slices correspond to top-level document components.

2. A simple slice (representing a concept attribute) has to be mapped to a document component template that additionally contains a content unit that again contains one or more media component templates. These media components correspond to the media items representing the slice's owner concept attribute, their types have to be determined

by the media type(s) associated with that concept attribute in the Media Model (MM). *Integer* and *String* attributes have to be mapped to text components, media attributes to corresponding media components (image, audio, video, etc.).

3. Whenever a slice (complex or a simple) is part of a Set construct, the appropriate component template that was assigned to it should be defined as an iterative template.

4. Finally, slice navigation relationships between two slices have to be mapped to hyperlink components between the appropriate component templates. Again, when a slice navigation relationship is based on a 1:n concept relationship, an iterative hyperlink list has to be configured.

Figure 5.5 depicts this possible mapping process in a graphical (schematic) way by example of the slice representing painting techniques. The types of the created component templates are denoted by the abbreviations MC (media component), CU (content unit component) and DC (document component). Furthermore, the mappings are illustrated as arrows, each labeled by a number indicating the appropriate mapping rule from the above list.

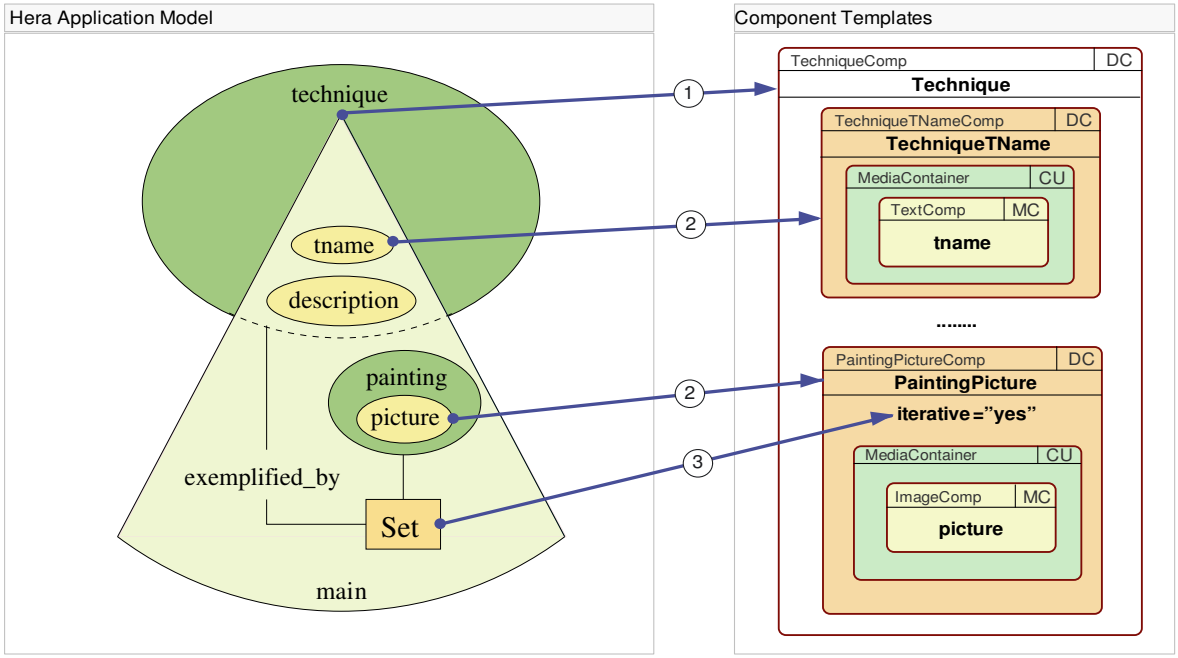

Figure 5.5: Slice to component template mapping

If the AM specifies adaptation aspects via appearance conditions, these have to be expressed in form of corresponding component variants and their selection methods (see Section 4.3.1). Again, this can be done in a straightforward way. Whenever a slice is provided with a Boolean appearance condition, the component associated to it has to be made a variable component containing only one variant. Moreover, according to the slice condition a selection method in the IF-THEN style has to be composed (see Section 4.3.1). Note, however, that the concept of adaptation variants supported by the component-based document model allow for more sophisticated kinds of adaptations than simple appearance conditions attached to navigational elements. To optimally address different client capabilities or user preferences, component authors might flexibly define different variants of the created component templates on all abstraction levels.

Furthermore, whenever a given adaptation specification concerns adaptivity, component authors also have to reason about how to update context model parameters according to users' interactions. For this purpose the context modeling components of the document generation architecture can be utilized (see Section 4.5.3). For instance, the number of a user's requests to a document component instance (e.g. representing a painter's biography) can be easily tracked in the Session Profile (see Section 4.5.2). This information can be utilized to define the appropriate component variants and their corresponding selection methods.

5.1.5 Presentation Design

The presentation design step of Hera bridges the logical level and the actual implementation by introducing the implementation independent Presentation Model (PM). Complementary to the AM, where the designer is concerned with organizing the Web application's overall structure and identifying which concept attributes from the entities of the application domain should be included in slices, the PM specifies how and when the identified slices should be displayed.

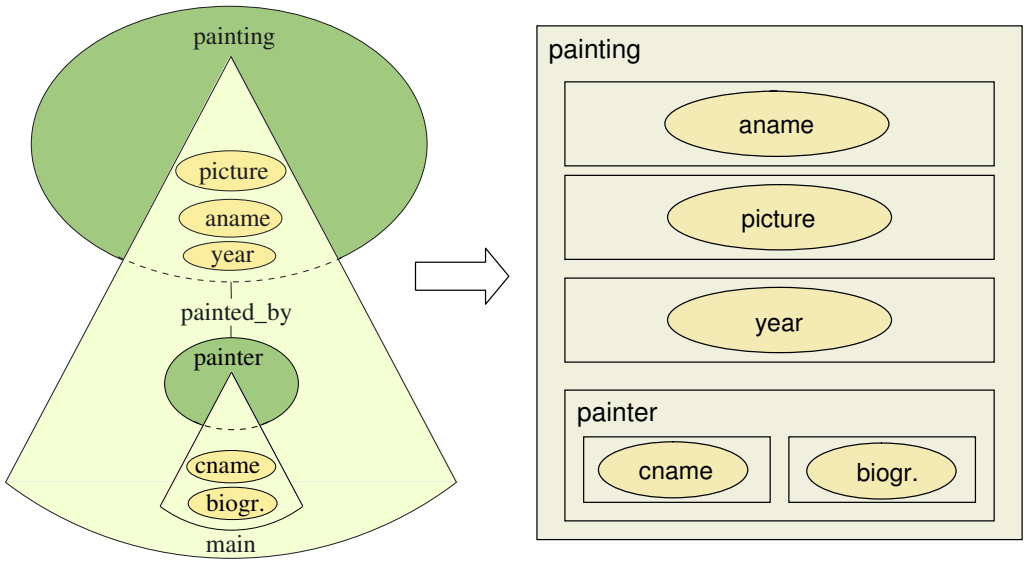

Figure 5.6: Presentation diagram (PD) example: assigning regions to slices

The PM is described by a presentation diagram (PD) consisting of regions and their relationships. A region is an abstraction for a rectangular part of the display area where the content of a slice is to be displayed. During presentation design, the slices introduced in the AM are mapped to regions (and subregions). The PD specifies the organization of regions in an informal, graphical way by means of region relationships that describe their relative position (e.g. above, below, left/right from) to each other [Frasincar et al. 2001]. As an example, Figure 5.6 depicts a possible presentation diagram assigned to the *painting* slice (see Figure 5.3). It dictates to display a painting's name, picture, year, and painter below each other and arranges the name and biography attributes of the painter in a horizontal manner.

Note that at the time when the work described in this chapter was carried out there was no RDF-grammar for expressing PDs in a formal way, nor were style design and adaptation considered in the PM. These issues were addressed in form of the Hera-AMACONT model ex-

tensions as part of the work presented in this thesis [Fiala et al. 2004a, Fiala et al. 2004b] and will be described here, respectively. In the following subsection different issues of presentation layer adaptation as well as a possible component-based realization of the presentation design step are discussed. An RDF(S)-based formalization of the corresponding Hera-AMACONT PM will be provided later in Section 5.3.1.

5.1.5.1 Adaptation at Presentation Design

As personalization and adaptation become prominent issues of Web engineering, it is inevitable to address adaptation at presentation design. Still, while adaptation has been extensively considered in navigation design (see Chapter 3), adaptation in the presentation layer has not been a central issue of Web design methodologies, yet. Nevertheless, it is in a lot of scenarios necessary to adjust a Web application's presentation aspects. As the most significant issues the following can be mentioned:

1. An important adaptation target is the *spatial placement* (layout) of the content elements of a Web page. Depending on varying user preferences and/or device characteristics (e.g. screen size, supported document formats, interaction techniques, etc.), they should be displayed differently. Possible layout adaptation techniques are:

   Reorganization: In this case the arrangement of content elements is adapted. Most typically, the purpose of this adjustment is to optimally fit a Web presentation to the varying display sizes of different client devices. Whereas for example the tabular arrangement of content may look well on conventional desktop computers, it could cause a lot of undesirable horizontal scrolling when being browsed on handhelds with limited display size. Similarly, the writing scheme of a Web presentation's language (e.g. left-to-right, right-to-left, horizontal, or vertical) can also significantly influence the arrangement of content elements like headers, footers, navigation bars, etc. [Evers and Day 1997].

   Exclusion: Information being unsuitable for a particular browser (e.g. a picture gallery for a monochrome mobile phone) or content elements without an important semantic meaning (e.g. company logos in an online shop) can be excluded from Web presentations on mobile devices with small displays or low bandwidth connections.

   Separation: As a (less strict) form of exclusion, it can be advantageous to put certain content pieces onto separate pages and automatically create hyperlinks to them. This mechanism is very useful to keep the structure of Web pages while providing a lot of information easily understandable on handhelds [Hwang et al. 2002].

2. A further adaptation target is the corporate design (i.e. the "look-and-feel") of a Web application, which is typically determined by decorative elements such as logos, background colors, font parameters (size, color, type), buttons, etc. Though not influencing the logical structure of a Web site, such design elements are important to appropriately convey the published information to the user. Consequently, presentation designers might provide alternative style variants in order to address different user properties and usage contexts. As possible adaptation aspects the following can be mentioned:

   Style preferences: The visitors of a Web presentation might have different style preferences based on their interests, education, and/or age. While e.g. a site addressing little children typically utilizes vivid colors and decoration elements [Nielsen 2002],

Web applications targetting a more "serious" audience are characterized by rather modest and simple layouts.

Cultural background: The cultural background of the targeted audience is also an important presentation adaptation feature. Barber and Badre provide an overview of so-called *cultural markers*, i.e. "interface design elements that are prevalent, and possibly preferred, within a particular cultural group" [Barber and Badre 1998]. As an example, specific color combinations, symbols, or decoration elements can have varying interpretations in different countries and cultures.

Accessibility issues: A further adaptation aspect to be considered at style design is accessibility. In order to appropriately address users with visual impairments (limited level of sight, color blindness, etc.), it requires to offer Web presentations with different color schemes, font types, and sizes. The World Wide Web Consortium (W3C) tackles this issue by offering a number of Web Content Accessibility Guidelines [Chisholm and Vanderheiden 1999] for Web designers and developers.

Environment characteristics: Web applications developed for mobile scenarios might adapt their visual appearance based on selected characteristics of the current environment. As an example, the contrast or brightness of a Web page might be adjusted to whether the user is situated in an indoor or anoutdoor context.

Specific events or time periods: Finally, the corporate design of a Web site might be also adjusted to specific events or time periods, such as seasons, anniversaries, or festivals. As a typical example, Web-based online shops or communities are often decorated with a dedicated layout at festivals like Halloween or Christmas, at Valentine's day, or even during sport events like e.g. football world championships.

3. Third, the qualitative adjustment of the media objects included in a Web site (e.g. to the display capabilities of different end devices) is also an important adaptation issue to be considered at presentation design. Still, since the assignment of media types to the application's conceptual model is specified in the already presented Media Model (see Section 5.1.1.1), these adaptations should be also defined there, respectively. As already mentioned, the adaptation definitions concerning the media model might be modified or extended at presentation design.

4. The aforementioned examples represent static adaptation. However, in some cases it is meaningful to consider dynamic adaptation, i.e. adaptation according to parameters that may change while the Web presentation is being browsed. As a possible scenario (in presentation design) we mention the dynamic reorganization of presentation elements on a page when the user resizes his browser window or the automatic reconfiguration of a Web presentations color scheme (colors, contrasts, brightness) when a mobile user enters a differently illuminated area (e.g. changing from an outdoor to an indoor context).

5.1.6 Realization with Document Components

The aggregation hierarchy of component templates was determined at application design. Now, based on the guidelines of the graphical presentation diagram, component authors are expected to specify the layout attributes of those component templates as well as the corporate design of the resulting presentation. Furthermore, to address the possible adaptation issues mentioned above, they also have to consider layout and design style variants.

As mentioned in Section 4.3.2, the component-based document format provides an XML-based mechanism for specifying the spatial adjustment of subcomponents within their container components in a size and client-independent way. Those abstract layout definitions support the automatic conversion of component structures to Web presentations in different output formats and are well suitable for implementing an abstract presentation design consisting of a hierarchy of rectangular regions. Consequently, the spatial relationships between regions defined in the PD have to be mapped to such component layout descriptions.

Again, this mapping can be performed in a straightforward way. Beginning at top-level document component templates and visiting their subcomponents recursively, one has to declare for each component template how its immediate subcomponents are arranged. As an example, the component templates containing the concept attributes describing a painting (see in Figure 5.6) can be arranged according to a vertical *BoxLayout* scheme. Similarly, the name and the biography attributes of the corresponding painting's painter can be organized based on a horizontal *BoxLayout*. When defining such layout managers, component authors can use their various configuration options (widths, heights, alignments, etc.) which were described in detail in Section 4.3.2.

The corporate design of a component-based presentation can be specified by the creation and configuration of corresponding CSS media components. The CSS standard of the W3C [Bos et al. 2006, Lie 2005] allows for the definition of a Web presentation's design and style elements (background colors, font sizes and types, link colors, etc.) and is thus perfectly suitable for this purpose. A CSS media component contained by a composite document component specifies the corporate design of the content it displays. Whenever there are several CSS components in a component-based Web document, they can redefine each other's style definitions by the order of their occurrence in the overall component hierarchy.

In order to cope with the adaptation issues described in Section 5.1.5.1, component authors might create different layout variants for components, each bound to a specific adaptation condition. For instance, the typical small display size and horizontal resolution of handheld devices would require to present not only the attributes of a painting, but also the names and the biographies of their painters below each other (i.e. according to a vertical *BoxLayout* scheme). Similarly, the corporate design of a component-based presentation can be also adapted by specifying different CSS media component variants and their selection methods.

After this authoring step, the content, the structure, and also the layout layout managers of the resulting components (templates) are fully specified. They manifest a component-based implementation of the corresponding design models.

5.1.7 Summary

This section exemplified the development process of component-based adaptive Web presentations according to the design phases dictated by the model-based Hera design method. Based on a small example application, it was shown how during the phases of design and implementation different application concerns (content, navigation, presentation) as well as corresponding adaptations (both static and dynamic) can be taken into account systematically. That is to say, a possible model-based authoring process for the developers of component-based adaptive Web presentations was introduced[5]. As a short summary, Table 5.1 recapitulates the identified design steps, their "implementation recipe" based on adaptive Web document components, but also the adaptation issues to be addressed in each development phase.

[5]A summary of component-based Web application prototypes realized based on the authoring process and tool support described in this chapter will be given in Section 5.4.2.

	Design & Modeling	**Component-based Implementation**
Conceptual Modeling	specification of the application's domain model	creation, retrieval and structured storage of media components representing concept attributes
CM Adaptation	adaptation of media quality	creation of media component variants with quality alternatives
Application Modeling	design of the application's navigational structure by slices and slice relationships	creation and interlinking of composite components (content units, document components) and templates
AM Adaptation	adaptation of slice aggregation and navigation	definition of alternatives for subcomponents and hyperlink structures
Presentation Modeling	design of the user interface based on regions and style definitions	definition of components' layout managers and CSS styling
PM Adaptation	design of layout and style adaptation	definition of alternative layout managers and CSS components

Table 5.1: Summary of design and implementation phases

The mentioned development steps facilitate a structured design and implementation process for component-based adaptive Web presentations. The resulting component templates constitute a dynamic component-based hypermedia presentation realizing the different design (and adaptation) issues expressed by the corresponding design models. Consequently, they can be used as the input of the pipeline-based document generator introduced in Section 4.5. As described there, for each user request the corresponding component template is retrieved and instantiated with the requested data. According to the current usage context, it is then subdued to a series of transformations, each considering a certain adaptation aspect. The resulting Web presentation is automatically adjusted to the actual usage context.

Note, however, that the development process presented in this section is only one possible approach for data-driven component-based adaptive Web applications. As mentioned before, the abstraction gap between design methods and implementation entities (components) allows to use different methodologies for developing component-based adaptive Web sites.

5.2 A Modular Authoring Tool for Component-based Adaptive Web Applications

As illustrated in the previous section, the component-based document format and its document generation architecture provide a sound basis for the development and publication of adaptive Web presentations. Based on a structured authoring process (e.g. Hera-AMACONT), authors can compose adaptive Web applications from reusable components in a disciplined way, by subsequently taking into account different design concerns, their adaptation issues, as well as their corresponding "implementation recipes". Still, the complexity of the component-based document format's underlying XML grammar calls for an intuitive authoring tool that supports this composition process in a graphical way. Such an authoring tool should provide

following functionality:

1. **Visual authoring support:** The authoring tool should offer graphical editor modules for the creation, configuration, and composition of document components. These should hide the low-level details of the XML-based component description language from authors, allowing to create component-based documents in a visual way.

2. **Independent authoring of separate concerns:** The authoring tool should provide a number of specialized editors allowing to separately configure different component properties (such as content, structure, layout, interlinking, adaptation) in different phases of the authoring process.

3. **Support for instance- and template-level authoring:** The authoring tool should facilitate the creation of both component instances and component templates. For the latter case it should allow authors to intuitively access dynamic data sources and configure the appropriate queries.

4. **Preview functionality:** In order to allow authors to test the currently created/edited documents, the authoring tool should provide a flexible preview functionality depending on the actual user, context, and device characteristics.

5. **Flexible authoring workflows:** Instead of being bound to a specific authoring process (e.g. the one presented in Section 5.1), the authoring tool should provide a flexible set of editor modules for the manipulation of different component types and properties. Authors should have the freedom to flexibly use these editors based on the current authoring scenario, thus being able to proceed based on different process models.

6. **Extensibility:** To cope with the flexibility and extensibility of the component-based document format (see Section 4.6.3), the authoring tool should also be based on a modular and extensible architecture. This should facilitate to integrate both alternative editor modules for existing and new editors for future component types.

To fulfill these requirements, a graphical component authoring tool called the AMACONT-Builder was developed [Fiala et al. 2005]. It is based on an extensible set of visual editor plug-ins and allows to graphically create and compose document components on different composition levels. Furthermore, it also supports the configuration of both their adaptation variants and adaptive layout. Note, however, that as a tool designated for component authoring, the AMACONTBuilder is oriented at the phase of (component-based) implementation in the overall Web engineering process (see Section 3.1). That is to say, instead of being bound to a specific methodology (e.g. Hera-AMACONT), it allows to compose adaptive Web components based on different authoring processes[6].

This section gives an introduction to the AMACONTBuilder. First, its basic concepts and main architecture is presented. Then, selected editor modules are described in more detail, supporting different phases of the authoring process of component-based adaptive Web presentations. Finally, a couple of implementation issues are briefly summarized.

[6]The issue of the model-driven generation of component-based adaptive Web applications based on Hera-AMACONT models will be subject to Section 5.3.

5.2.1 AMACONTBuilder: An Overview

The AMACONTBuilder is a modular authoring tool aimed at the visual development of component-based adaptive Web applications. It is based on an extensible authoring framework [Chevchenko 2003] that allows to edit arbitrary XML documents. The framework was developed at the Chair of Multimedia Technology of the Dresden University of Technology and provides generic functionality for parsing XML files into and internal object model, as well as for implementing editor plug-ins dedicated to specific object model (XML) elements. Thus, it can be easily extended by graphical editor modules (plug-ins) to visually author content based on a given XML grammar. Previously, the framework was successfully utilized for the development of courseware in the CHAMELEON project (see Section 3.2.5).

As shown in Figure 5.7, the user interface of the AMACONTBuilder consists of two main parts: the *application frame* and the *document frame*. The *application frame* provides generic functionality for configuration options and file management. It is responsible for parsing XML-based documents to an internal object model, for assigning editor plug-ins to parts of this object model, and for serializing the modified object model to XML, respectively. It contains the *document frame* showing the currently opened (edited) document. This is again divided into two parts: the *navigation frame* and the *editor frame*.

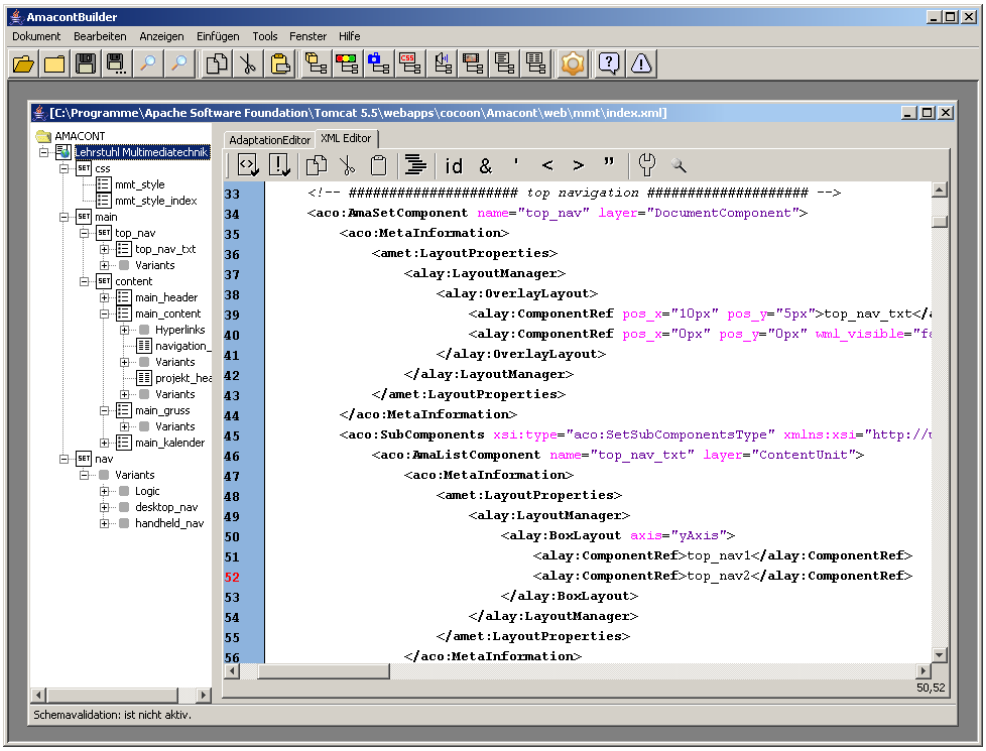

Figure 5.7: AMACONTBuilder overview [Fiala et al. 2005]

The *navigation frame* on the left provides a tree-based view on the node (component) structure of the currently edited document. When navigating through this component hierarchy, the specific editors assigned to the appropriate component types are automatically activated in the *editor frame* (shown on the right). While in Figure 5.7 the *navigation frame* shows the overall component hierarchy of the edited document, it is possible to use predefined filters. For instance, an author aimed at the creation and configuration of image components

has the possibility to display only the corresponding components and filter out all other ones.

Finally, the *editor frame* provides space for the actual editor plug-ins associated to different node (or component) types. It displays the editors associated to the currently edited component (chosen in the navigation frame). A component type might be associated with several editor modules. As an example, content unit components and document components have both editors for manipulating the aggregation of their subcomponents as well as their layout. In this case all corresponding editors associated with the actual author role are shown as separate panes and can be activated by the author, respectively. The assignment of editors modules to component types is determined by an XML-based configuration file that can be individually set for different authoring scenarios.

While there are editor modules being applicable to all kinds of XML content (e.g. the XML code editor shown in Figure 5.7), most modules are assigned only to specific node (component) types and are thus activated at well-defined phases of a given authoring process. The following sections give a short overview of the most important existing editor modules. According to the main steps of the authoring process (and the running example) described in Section 5.1, selected editors for content, navigation, and presentation authoring are presented.

5.2.2 Editors for Content Authoring

For the graphical creation of media components different visual editors (*text editor*, *image editor*, *CSS editor*, etc.) have been created. By example of a picture representing a painting, Figure 5.8 presents the *image editor*. It allows to upload images in different formats (e.g. jpeg, gif, bmp, png), to edit their properties, and to save them as image components. While most image metadata properties can be configured by appropriate input fields, some editing operations can be also performed visually. As an example, the size of an image component can be simply altered by mouse dragging.

To support for adaptation, the media editors (but also all other kinds of component editors) were extended with a generic mechanism for creating content alternatives. For instance, in the image editor it is possible to provide an alternative text for browsers that are not able to present images. Furthermore, image variants with different quality alternatives can be added. Such variants can be created in three ways: by uploading alternative images, by reconfiguring (e.g. resizing) the current image and save it as a new variant, or by generating new images automatically. In the latter case the author can predefine the properties (e.g. pixel size, color depth, image format) of an arbitrary number of variants to be created. According to this configuration, the alternative media instances are generated automatically. This feature was implemented by using the Java API of ImageMagick [@ImageMagick].

After creating media component alternatives, component authors can define their adaptive behavior by attaching adaptation conditions to each variant. As discussed in Section 4.3.1, these conditions are Boolean expressions referencing parameters from the CC/PP-based context model. For the configuration of adaptation conditions the *profile browser* was developed (see Figure 5.9). It allows authors to visually navigate through the hierarchy of profiles, to choose the appropriate parameters, and to insert them into adaptation conditions. The profile browser can be configured by an RDFS document defining the current application's context model. The example in Figure 5.9 declares to use the current picture for browsers with less then 8 bits per pixel color depth and less than 400 pixel horizontal resolution. As can be seen, authors can "click together" complex logical expressions by visually choosing the appropriate parameters from the pop-up window presenting the context model's hierarchical structure.

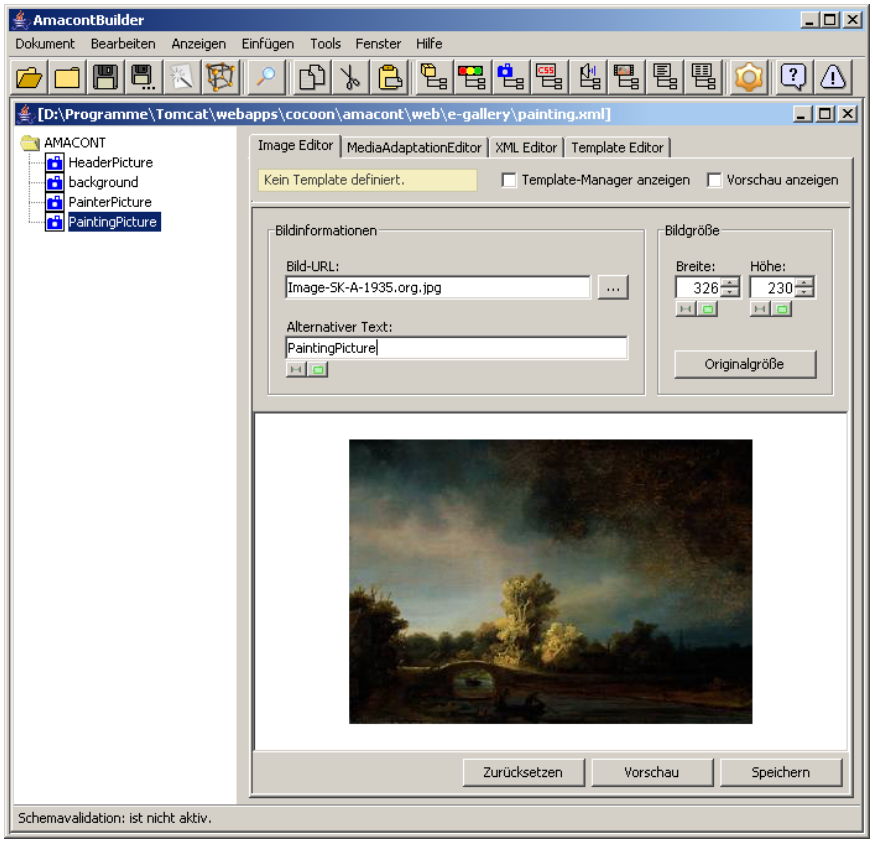

Figure 5.8: Image editor

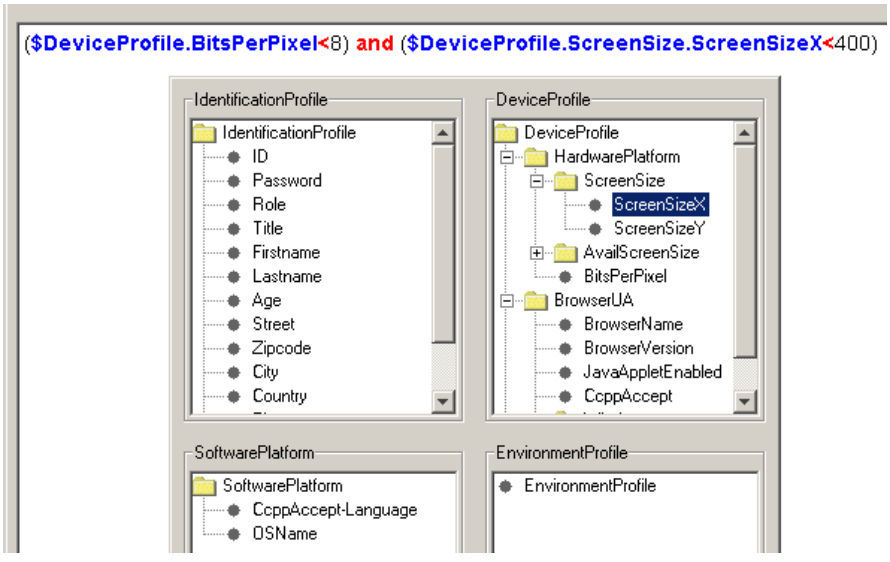

Figure 5.9: Defining adaptation conditions with the profile browser

The mechanisms described above aim at authoring adaptable content (media component) instances. Still, in order to support for data-intensive Web applications, the editor tools can be switched from this “instance mode” to the so-called “template mode”, i.e. authors can

assign a data source query to the currently edited component [Tietz 2006]. The result is a component skeleton (component template) that can be filled with dynamically retrieved data on-the-fly.

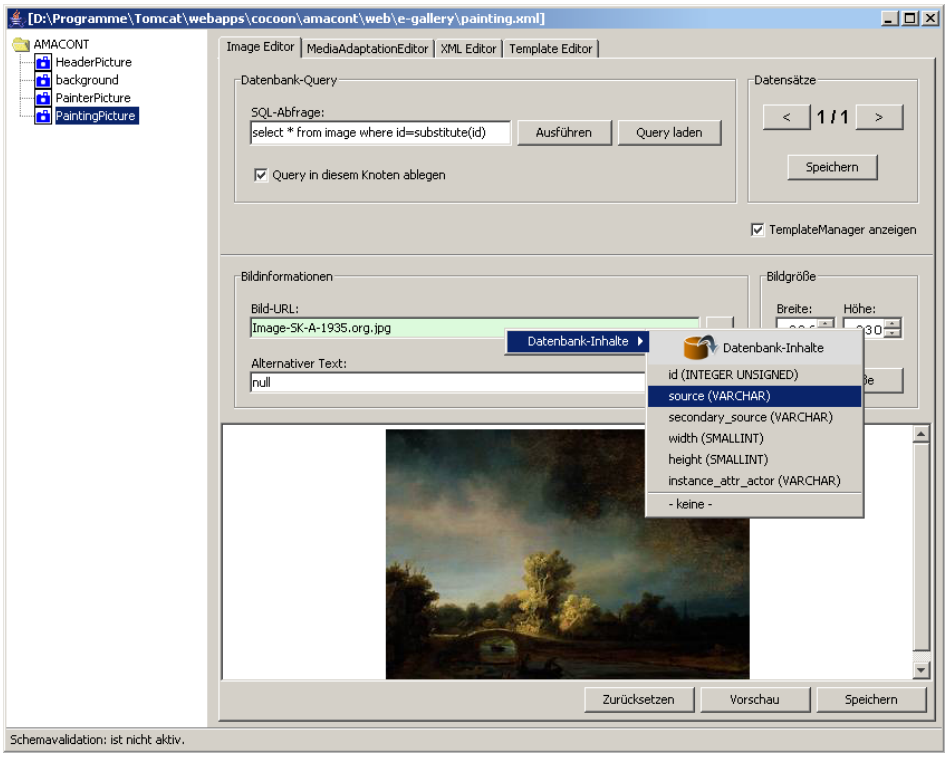

Figure 5.10: Template editor for image components

Figure 5.10 depicts this mechanism by example of the image editor. As can be seen, in this particular case the author defined a simple query retrieving images (of paintings) as well as their metadata from a relational data source. Note, however, that it is also possible to access the results of a query which was defined on a higher level in the component hierarchy. After defining a query, the author has the possibility to assign specific fields of its result set to the attributes of the image component. The drop-down list shown in Figure 5.10 illustrates how he assigns the *source* field of the query's result set to the source attribute of the image component template. Consequently, such dynamic attributes will be filled with the appropriate values on-the-fly. As a matter of course, it is allowed to define some attributes as constants. Furthermore, it is also possible to define adaptation operations on template level by creating a variant of the component template, assigning an alternative query field (containing e.g. the PDA variants of images) to it and defining a corresponding selection method. For more information on the AMACONTBuilder's media editors the reader is referred to [Fiala et al. 2005].

5.2.3 Editors for Hypertext Authoring

Whereas the editors for content authoring facilitate the creation of media components (or templates) and their adaptation variants, the editor modules for hypertext authoring focus on structuring and interlinking components to complex hypermedia structures. They are further divided into two groups: 1) editors for composing component hierarchies and 2) editors for

defining hyperlink structures between those hierarchies.

5.2.3.1 Editors for Creating Component Hierarchies

The visual creation of component hierarchies is facilitated by two modules, the *structure editor* and the *subcomponent editor* [Niederhausen 2006]. While the former one aims at specifying the overall composition structure of a component-based Web document, the latter one allows to manipulate the immediate subcomponents (i.e. child components) of a composite component in more detail.

The main application scenario of the *structure editor* (see Figure 5.11) is the creation of a component-based Web document "from scratch". Starting from an empty document component, authors can easily specify its internal structure by defining its subcomponents (and the subcomponents of those subcomponents) in a visual way. The available component types to be included (created) are visualized on the bottom part of the editor and can be placed into a container component (or moved from one container to another) by using "Drag&Drop" mechanisms. During this composition process, the integrity constraints dictated by the component-based document format are strictly taken into account. For instance, a media component has to be always contained by a content unit component.

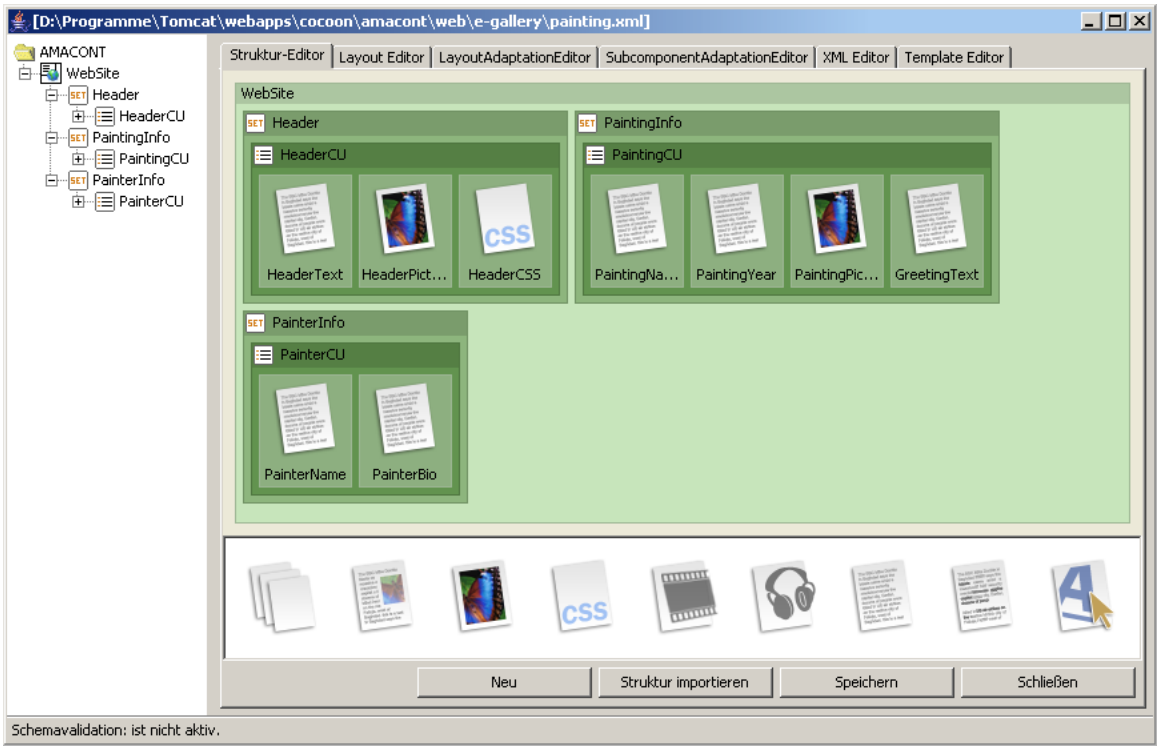

Figure 5.11: Structure editor

As a matter of course, the structure editor is also ideal for visualizing (or manipulating) the composition hierarchy of an already existing component hierarchy. Furthermore, it allows authors to directly access any component contained in a component-based Web document. By double-clicking on an arbitrary component the appropriate editors assigned to it are automatically activated. As an example, the activation of an image component invokes the image editor (shown in Figure 5.8) in a modal editor window. Moreover, the structure

editor is not only associated with top-level document components, but also with all kinds of composite components. In the latter case it only visualizes the subcomponent tree of the currently selected composite component, thus providing partial views on (fragments of) complex component hierarchies.

Whereas the structure editor is ideal for creating and visualizing component hierarchies of arbitrary depth, there are also cases when component authors would like to deal only with the immediate subcomponents of a composite component. A typical use case is the population of a component structure with concrete media components or the definition of adaptation variants: the subcomponent structure of a component might vary according to a given user model or context model parameter. For this reason the *subcomponent editor* shown in Figure 5.12 has been developed. It is associated with composite components (both document components and content unit components) and visualizes their immediate subcomponents as an unordered list.

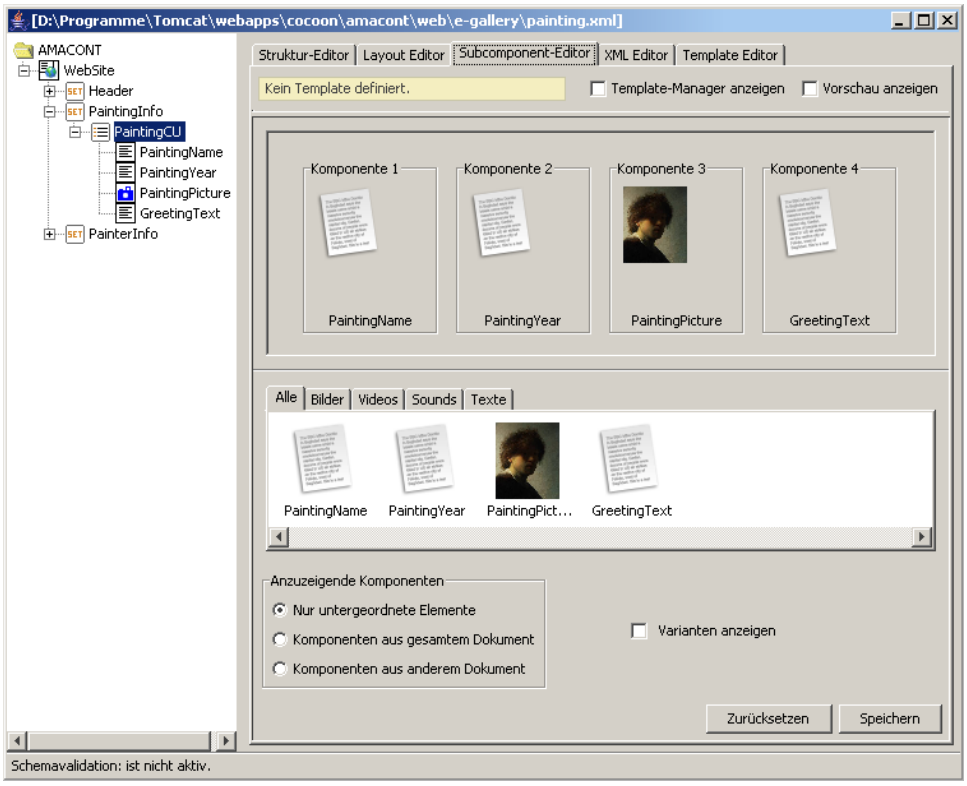

Figure 5.12: The subcomponent editor

According to the running example described in this chapter, the composite component shown in Figure 5.12 contains information on a painting. In this case it is put together from several subcomponents: the painting's name, textual description, creation year, and a greeting text addressing the user. As a matter of course, the subcomponent editor can be also switched to template mode. Taking the example, one could connect it to a database query in order to present dynamic information on paintings selected at run-time. What is more, a composite component may aggregate both static and dynamic (i.e. template-based) subcomponents. For instance, while the media items describing the actual painting could be dynamically retrieved, the greeting text addressing the user is constant for all paintings and is therefore a static component instance. Finally, similar to media editors, the subcomponent editor also facilitates the creation of alternative component structures and selection methods.

As an example, the author might provide additional information on paintings for expert users, or insert some multimedia material for devices capable of presenting it.

5.2.3.2 Editors for Creating Hyperlink Structures

The AMACONTBuilder provides two editor modules for graphically authoring hyperlinks and hyperlink structures [Niederhausen 2006]. While the *graph editor* facilitates the visualization of a component-based Web presentation's overall hypermedia structure, the *hyperlink editor* supports the creation of single hyperlinks or hyperlink lists.

The graph editor shown in Figure 5.13 presents the hypermedia structure of a component-based Web presentation in a graph-like way[7]. The nodes of the graph correspond to top-level document components. A directed edge between two such nodes means that there is at least one hyperlink component connecting them. Still, for better readability, "parallel hyperlinks" between two documents are merged to one edge, i.e. only the connectivity of the corresponding nodes is represented. Furthermore, author can use "filter functions" to display only hyperlinks of a given type (e.g. typed links, template-based links or adaptive links). While mainly serving for visualization purposes, the graph editor is an ideal starting point for further authoring operations. When clicking on a node, the AMACONTBuilder opens the appropriate document that can be then edited in more detail.

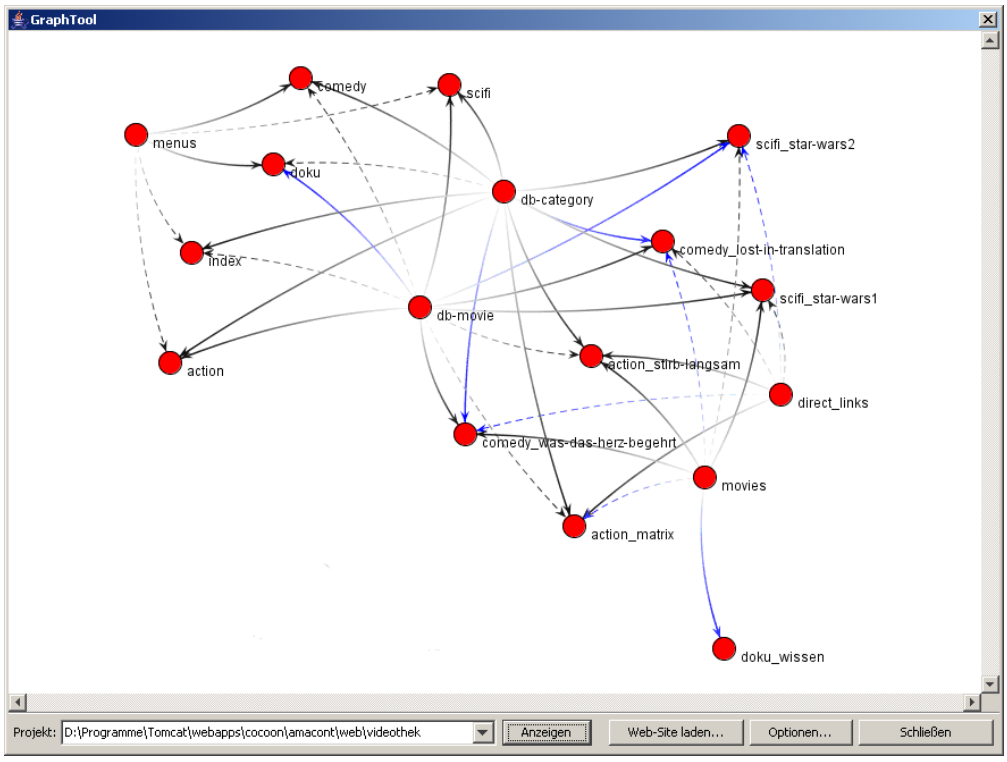

Figure 5.13: The graph editor

In order to create and manipulate single hyperlinks or hyperlink lists the so-called *hyperlink editor* was developed. However, instead of being a stand-alone module for authoring abstract

[7] In order to present the graph editor based on a larger example, Figure 5.13 illustrates the navigation structure of a component-based video rental shop.

hyperlink components independent of the underlying content, it is used in combination with the editor modules aimed at the creation of components that serve as the starting anchor of a hyperlink component. As an example, the author of a text component can mark an arbitrary text fragment and define a hyperlink starting from that component. Furthermore, it is also possible to visually create complex navigation bars (i.e. composite components consisting of a number of anchors and corresponding links) in form of specific document components. As a matter of course, the resulting structures are stored as separate content components (e.g. media component) and hyperlink components. However, this integrated view supports for a more intuitive way of working for component authors.

Similarly to other editor modules, the hyperlink editor can be also switched to the template mode. Again, different parameters of a hyperlink (such as its target, anchor text, or even the request parameters attached to it) can be customized by appropriate data queries. Furthermore, it is also possible to specify the adaptive behavior of a hyperlink component. Four basic adaptation techniques are supported: *link hiding*, *link removal*, *link disabling*, and *link annotation*. The usage of a given adaptation technique can be bound to a condition that references the context model. Again, such conditions can be visually defined by using the profile browser (see Figure 5.9).

5.2.4 Editors for Presentation Authoring

Finally, a number of editor modules for configuring the presentation layout of component-based Web documents have been developed. As discussed in Section 5.1 they serve two purposes: the definition of a component's abstract layout, and the configuration of its styling by using CSS. These tasks are facilitated by the *layout editor* and the *CSS editor*, respectively.

Figure 5.14 shows a screenshot of the layout editor. It facilitates the assignment of layout managers to components and to configure their various attributes in an intuitive visual way. Layout managers are visualized by means of a grid that can be filled by icons representing subcomponents. Various mouse dragging and "Drag&Drop" techniques have been realized in order to perform most operations graphically, such as resizing the grid, placing subcomponents into grid cells, changing their alignment, etc. Besides, various input fields for fine-tuning all possible layout attributes (both layout attributes and subcomponent attributes) can be found on the right editor pane. Furthermore, a preview function for testing the current layout in XHTML has been developed, too.

Figure 5.14 depicts a possible abstract layout for the component presenting paintings (see Section 5.2.3). Based on vertical BoxLayout, the content pieces describing a painting are arranged in a linear structure. Note that even though the figure depicts a concrete "painting instance", this editor can be switched to template mode, as well. However, as far as iterative templates (i.e. templates with an unpredictable number of subcomponents) are concerned, only the layout managers GridTableLayout (with only one predefined dimension) or BoxLayout (with an undefined number of subcomponents) can be utilized. Furthermore, layout adaptations can be easily specified by the creation of appropriate layout alternatives and the definition corresponding selection methods.

On the other hand the CSS editor aims at defining the design of the resulting application. It is a simple (media component) editor module allowing for loading CSS files as well as for manipulating their style definition entries. It allows authors to select different elements of the respective output format and configure their layout attributes (e.g. font sizes, colors, text decorations, etc.). The resulting definitions are stored as CSS media components. Of course, similar to the other editors, component authors can again define alternative CSS variants

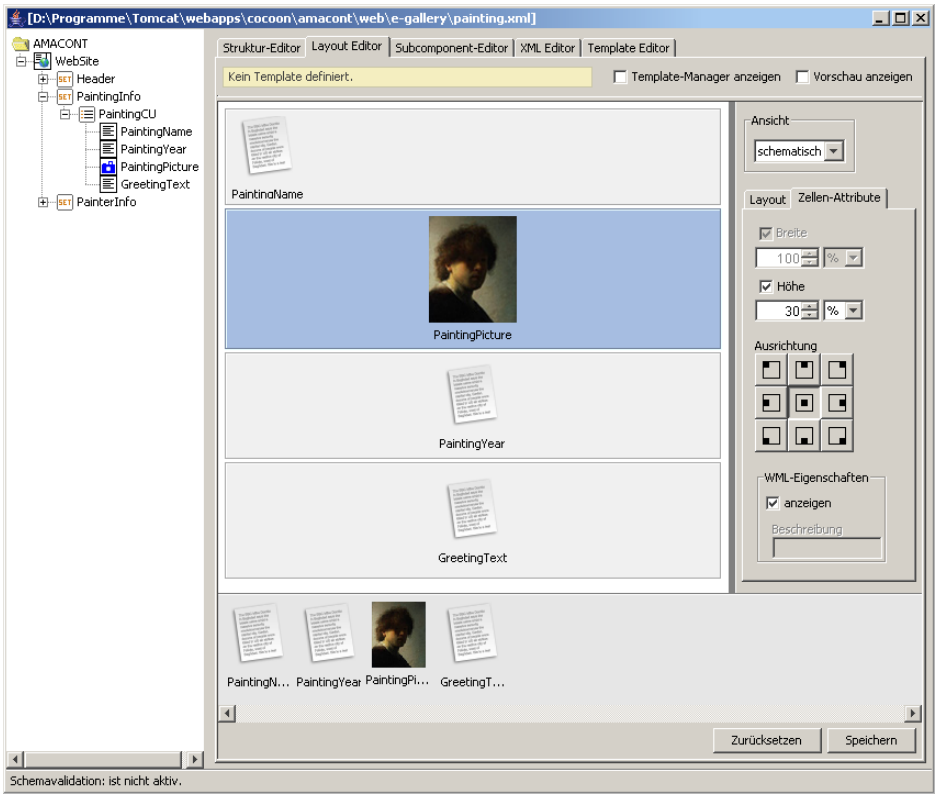

Figure 5.14: Layout editor

and assign them to a specific context model parameter.

5.2.5 The XML editor

The main goal of the AMACONTBuilder is to provide a set of *visual* editor modules that allow component authors to abstract from the the component-based document format's underlying XML grammar. Still, in order to exploit the full range of language features (e.g. the ones for which there are no graphical authoring plugins available, yet) or to debug the XML code generated by other visual editor modules, it is also required to provide "low-level" source code editing support. For this purpose, the built-in XML editor provided by the AMACONTBuilder's underlying plug-in architecture can be utilized[8]. Inspired by the functionality offered by professional XML tools (e.g. XMLSpy [@XMLSpy]), it provides a number of useful features, such as:

- **syntax highlighting** with adjustable font types and sizes, as well as configurable color schemes for different parts of XML documents, like tag names, attribute names (and values), comments, DOCType declarations, etc.
- **partial views** on complex XML documents by displaying only the XML subtree that belongs to the node (component) being currently selected in the navigation frame
- **automatic indentation** of XML tags to an easily readable "pretty-print" layout

[8]Note that the XML editor was already presented in Figure 5.7.

- **well-formedness checking** as well as **validation** of XML documents based on associated XML schemas with appropriate textual feedback on found errors
- **schema-based code completion** for XML (sub)elements and attributes based on well-formedness rules as well as default values provided by a corresponding XML schema

While being primarily used to edit XML documents based on the component-based document format, the XML editor is a generic tool that is applicable for arbitrary XML grammars.

5.2.6 Implementation Issues

The AMACONTBuilder was implemented in Java and is a modular authoring tool allowing for creating and editing arbitrary XML documents. As mentioned above, it is based on a generic framework [Chevchenko 2003] that can be extended by graphical editor plug-ins for visually authoring specific types of XML content. In order to provide programmatic access to all kinds of XML data, the AMACONTBuilder utilizes a flexible internal object model which is based on JDOM [@jdom]. This generic object model was extended by specific classes that provide an API for efficiently manipulating adaptive Web components. That is to say, component-based adaptive Web documents are automatically parsed into a hierarchy of component specific objects when they are opened by the AMACONTBuilder.

The UML diagram shown in Figure 5.15 depicts the most important classes of the object model hierarchy that are specific to the component-based document format. The root of this object hierarchy is the class *AmacontNode* providing a number of generic methods for component manipulation. The specific classes representing concrete component types inherit from it and declare their (additional) specific attributes and methods, respectively. As an example, the developer of an editor plug-in dealing with image components can utilize predefined methods of the class AmaImageComponent for getting and setting image metadata, for creating image component variants and selection methods, etc.

Note that the utilization of such an object model has different advantages. First, plug-in programmers can use a high-level API for manipulating adaptive Web components and do not have to bother about their concrete underlying XML-based format. Second, this solution provides also more robustness regarding to modifications of the utilized XML languages. During the "evolution" of this dissertation different changes to the component model's XML-based description language were made, especially in order to provide less redundant descriptions and better performance in the document generation process. Still, as the plug-ins of the AMACONTBuilder work on an internal object model, it was sufficient to adjust the mappings between that model and the XML-based formats, not needing to modify the application logic of specific plug-ins.

The editor modules (plug-ins) of the AMACONTBuilder have to implement a corresponding interface class[9]. It specifies a number of generic methods, e.g. for accessing the underlying object model, to set up the editor's graphical user interface, to check if the editor performed modifications on the object model, to write back these modifications to the object model, etc. Furthermore, whenever an editor should be extended with the capability to create and manage adaptation variants of the edited component type, it can inherit from a specific predefined class[10] that already implements this functionality in a generic way.

The assignment of editor modules to a specific component type is managed by an XML-based configuration file called `amaplugins.xml`. Listing 5.1 depicts a "fragment" of this

[9] de.tudresden.inf.amacont.plugins.EditorPlugin [Müller et al. 2005]
[10] de.tudresden.inf.amacont.plugins.AbstractAdaptableEditor [Niederhausen 2005a]

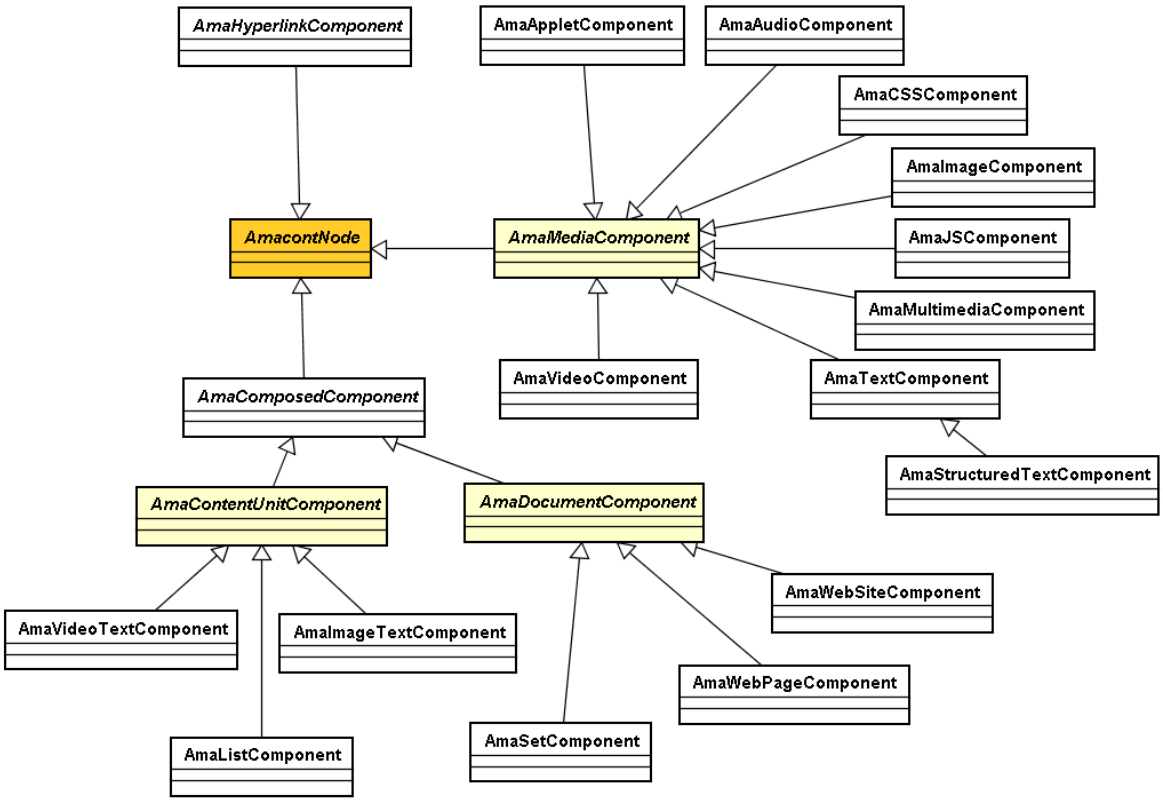

Figure 5.15: AMACONTBuilder object model

file aimed at associating the Java class implementing the image editor with all objects of the type AmaImageComponent (from the AMACONTBuilder's object model). The configuration specifies the plug-in's name, its version number, author, a short textual description, the class that implements it, and the elements from the object model to which it is assigned. While in this case the image editor is associated only with the elements of the type aco:AmaImageComponent, note that one can also associate an editor plug-in (e.g. the XML editor or the structure editor) with a number of classes of the object model.

```
1   <Plugin type="editor">
2     <Name>amacont.imagecomponent.editor</Name>
3     <Version>1.0</Version>
4     <Author>Matthias Niederhausen</Author>
5     <Description>Image Editor</Description>
6     <Class>de.tudresden.inf.amacont.plugins.ImageEditor</Class>
7     <Element name="aco:AmaImageComponent" />
8   </Plugin>
```

Listing 5.1: Assignment of an editor module to a component type

For further detailed information about the AMACONTBuilder's architecture, internal object model, configuration, and the implementation of its various editor modules the reader is referred to [Fiala et al. 2005, Chevchenko 2003, Niederhausen 2005b, Niederhausen 2006, Tietz 2006].

5.3 From Component Authoring Towards Automatic Model-Driven WIS Generation

The first part of this chapter introduced the structured Hera-AMACONT methodology for the development of component-based adaptive Web applications. According to the design steps identified by Hera, it was shown how those concepts can be applied to systematically implement adaptive Web applications by creating, configuring, and aggregating components (or component templates). During the phases of design and implementation, a main focus was put on the consideration of different adaptation issues (concerns). Thus, a possible model-based authoring process for component developers was provided.

Basically, there are two possibilities to put such an identified process model into practice. In the first case component authors can use the existing (and already introduced) modules of the AMACONTBuilder. Considering the different designs and keeping in mind the identified authoring steps, they can then build complex adaptive Web applications by creating, configuring, and composing reusable components (or component templates). The advantage of this "manual mapping" approach is the utilization of a graphical authoring tool that allows for visually editing component properties in detail. Furthermore, in a similar way, it is possible to proceed according to the steps identified by another design or process model. However, such a manual mapping also means that the applied design model serves mainly as a "guideline" (or documentation) for component authors, i.e. its semantics is not explicily exploited when creating component-based adaptive Web applications.

The second possibility is take a further step from model-based to model-driven component engineering and add automation to the overall process of design and implementation. The reason for this is the fact that, besides graphical representations (in form of diagrams), high-level design models can be also expressed in a formal way. As an example, we again consider Hera that provides RDF(S)-based specifications of its different design issues. By explicitly describing model semantics, such specifications can be used for the automatic model-driven generation of a corresponding implementation. This approach is also pursued by the Hera Presentation Generator (HPG [Frasincar et al. 2005]), a tool aimed at creating and implementing Hera models. However, prior to the work described in this dissertation, Hera's presentation model was not formalized, nor was adaptation at the presentation level addressed and implemented in the Hera tools. Moreover, HPG uses conventional Web document formats (such as HTML) as its implementation model, not allowing to reuse the generated implementation artefacts in a component-based manner. Thus, this section aims at the automatic, model-driven generation of component-based adaptive Web presentations from high-level design model specifications. This will allow to combine the modeling power of the (extended) Hera design method with the flexible reuse, presentation, and adaptation capabilities provided by the component-oriented document format and its publication architecture.

To achieve this goal, two requirements have to be fulfilled. First, all design models describing an adaptive Web application have to be expressed in a formal way. Second, a series of model-driven transformation steps is needed to automatically map these model descriptions to a component-based implementation. To meet these requirements, this section provides a facility for the (currently missing) RDF(S)-based formalization of a Web application's presentational aspects (as well their as corresponding adaptation issues) at model level. Bridging the gap between the application model and the actual implementation, this formalization will be utilized to automatically generate an adaptive component-based presentation.

The rest of this section is structured as follows. In Section 5.3.1 the concept of abstract layout managers (from the component-based document format) is adopted to the

Hera-AMACONT presentation model, and its RDFS-based description is provided. According to this formalization, Section 5.3.2 describes how high-level model specifications can be automatically transformed to an implementation utilizing the component-based document format and its document generation architecture. The XML-based transformation steps are explained in detail, and the resulting methodology is exemplified by a prototype application. Furthermore, selected aspects of dynamic adaptation provided by the overall presentation generation process are also discussed.

5.3.1 RDFS-based Specification of the Hera-AMACONT PM

In order to formalize the Hera-AMACONT presentation model, an RDFS-based specification of the PM schema was developed [Fiala et al. 2004a]. The basic idea behind it was to transfer (i.e. adopt) the concept of abstract layout managers from the component-based document format to the model level. The layout manager concept was already introduced in detail in Section 4.3.2. As mentioned there, layout managers aim at describing the spatial arrangement of components in a client-independent way, thus allowing to abstract from the exact presentation capabilities (e.g. window size) of a concrete browser display.

Note that in Section 5.1.4 significant analogies between document components and Hera slices were mentioned. Furthermore, a "recipe" for the mapping of slices to components was also introduced. Taking advantage of these analogies (and the fact that both slices and components rest upon XML technologies), it is thus straightforward to transfer the concept of layout managers to the model level. That is to say, the basic idea is the assignment of abstract layout descriptors to Hera slices in order to specify the arrangement of their subslices in an implementation-independent way. As a consequence, the RDFS-based PM formalization supports two mechanisms: 1) the definition of model-level layout managers and 2) their assignment to AM slices. A slice with an associated layout manager constitutes a so-called *region*: an abstraction for a rectangular part of the display area where the content of that slice will be displayed. These mechanisms will now be explained based on the running example used throughout this chapter.

Figure 5.16 depicts a schematic graphical presentation diagram (PD) of the running example's starting page (presenting painting techniques). Note that it is based on the corresponding application diagram (see Figure 5.3) which is extended by additional presentation specific information, i.e. the presentation diagram acts as an *overlay* of that application diagram aimed at specifying its layout[11]. As an example, the dark rectangle "behind" the top-level slice depicts the top-level region representing its contents. It utilizes the layout manager instance `BoxLayout1` for the spatial arrangement of the corresponding subslices (i.e. of the regions assigned to them). The simple RDF code snippet for specifying this layout assignment is shown in Listing 5.2.

```
1    <Slice rdf:about="#Slice.technique.main">
2        <layout rdf:resource="#BoxLayout1"/>
3    </Slice>
```

Listing 5.2: Layout assignment to a slice

The specific attributes of this layout (`BoxLayout1`) are also schematically shown in the diagram by means of arrows that are labeled with their names and corresponding values. Both

[11]Similar to the AM aimed at grouping the concepts of the CM to slices, the PM leans itself on the AM by further defining the spatial arrangement of those slices.

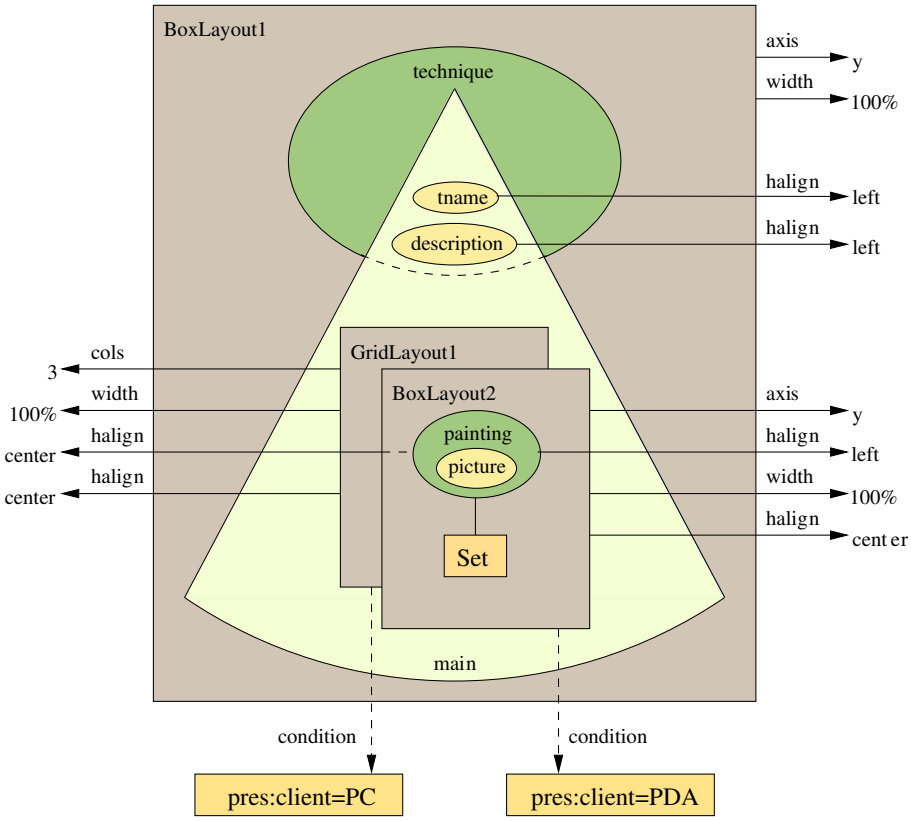

Figure 5.16: A Hera-AMACONT PM example [Fiala et al. 2004a]

attributes describing the overall layout and attributes specifying the arrangement of each referenced subslice (subregion) can be defined. Since these attributes were taken from the component-based document format, the reader is referred to Section 4.3.2 for more detailed information.

In this particular case the subslices (subregions) of the top-level slice (top-level region) are arranged in a vertical way. Concretely, these are the three subregions associated with the subslices `tname`, `description` as well as the link list pointing to the paintings representing the actual painting technique. The RDF-based representation of this layout definition is shown in Listing 5.3.

As already mentioned, layout descriptions of a given region describe only the spatial arrangement of its immediate subregions. Whenever these subregions also contain nested subregions, their appropriate layouts have to be additionally specified. In Figure 5.16 this is the case for the link list (`set-element`) pointing to the associated painting slices. The corresponding layout assignment is specified by the RDF code shown in Listing 5.4.

Note the attribute `pres:condition` that allows to declare simple *adaptation conditions* that reference parameters from the usage context. Whereas for example the paintings exemplifying the presented painting technique are arranged on a desktop in a tabular way (GridTableLayout1), the small screen size of PDAs requires to adjust them below each other (BoxLayout2). In Figure 5.16 these conditional layout assignments are visualized by the two overlapping regions as well as the two dashed arrows pointing to the rectangles containing their conditions.


```
<BoxLayout rdf:ID="BoxLayout1">
  <axis>y</axis>
  <width>100%</width>
  <subregion-ref>
      <subregion pres:align="left">
          <slice-ref rdf:resource="#Slice.technique.tname"/>
      </subregion>
  </subregion-ref>
  <subregion-ref>
      <subregion pres:align="left">
         <slice-ref rdf:resource="#Slice.technique.description"/>
      </subregion>
  </subregion-ref>
  <subregion-ref>
      <subregion pres:align="center" pres:valign="top">
         <set-element-ref rdf:resource="#SetOfLinks_1"/>
      </subregion>
  </subregion-ref>
</BoxLayout>
```

Listing 5.3: High-level BoxLayout definition example

```
    <Set-element rdf:about="#SetOfLinks_1">
       <layout rdf:resource="#GridTableLayout1"
             pres:condition="pres:client='Desktop'"/>
       <layout rdf:resource="#BoxLayout2"
             pres:condition="pres:client='PDA'"/>
    </Set-element>
```

Listing 5.4: Layout assignment to Set elements

Finally, Listing 5.5 presents the RDF code specifying the `GridTableLayout1` layout manager. According to this, the painting pictures (acting as the link anchors to the painting slices) are arranged in a tabular way.

```
<GridTableLayout rdf:ID="GridTableLayout1">
  <rows>2</rows>
  <width>100%</width>
  <height>80%</height>
  <space>10</space>
  <border>0</border>
  <header_align>xAxis</header_align>
  <subregion-ref>
     <subregion pres:align="center" pres:valign="center" ... >
       <slice-ref rdf:resource="#Slice.painting.picture"/>
     </subregion>
  </subregion-ref>
</GridTableLayout>
```

Listing 5.5: High-level GridTableLayout definition example

Note that in contrast to static components defined at instance level, Hera-AMACONT layout assignments have to be specified at *schema level*. Due to the dynamic nature of WIS applications, this means that the number of items in an access element (e.g. the number of

paintings exemplifying a given painting technique) is not known at design time. In such cases one should use either a `BoxLayout` with an undefined number of cells or (as shown in our particular example in Listing 5.5) a `GridTableLayout` so that only one of its dimensions (`columns` or `rows`) is predeclared. The missing dimensions (in this particular example the number of columns in the resulting table) are automatically computed at run time (see later in Section 5.3.2.2).

5.3.2 Automatic Generation of a Component-based Implementation

After the RDF(S)-based specification of the Hera-AMACONT PM, all design models can be expressed in form of RDF(S)-based specifications. Given these specifications, it is now shown how they can be utilized to automatically generate a component-based adaptive Web application that can then be published (and adapted) for different device, user, and context profiles. Furthermore, a prototypical implementation of this automatic hypermedia generation process is presented.

First, the general transformation architecture is described in overview. Then, the automatic model-driven generation of adaptive document component structures as well as their dynamic publishing process based on the actual usage context are explained. Finally, selected issues of dynamic adaptation (adaptivity) provided by the resulting component-based implementation are illustrated.

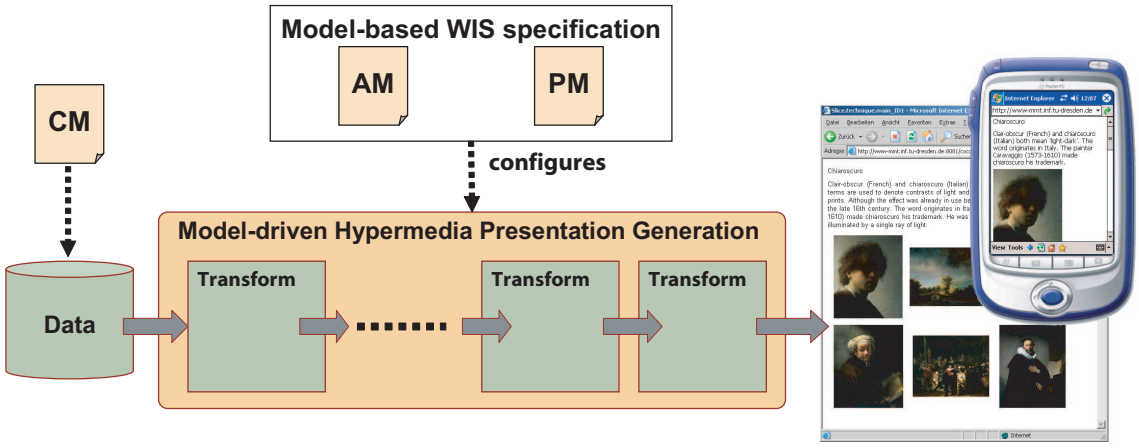

Figure 5.17: Model-driven WIS generation process overview

Figure 5.17 gives a general overview of the targeted model-driven Web presentation generation process. As depicted there, this general architecture consists of a series of transformations that convert some input data to a hypermedia (Web) application. The input data represents structured information that corresponds to the application domain (here defined by the CM). The transformations are configured by a number of models that dictate the application's navigational and presentational behavior. In the case of Hera-AMACONT, these are the AM and the PM, each being "enriched" with corresponding adaptation definitions[12]. We note, however, that this general architecture is also characteristic for other model-driven approaches.

Figure 5.18 depicts the concrete envisioned data transformation process in more detail. Its

[12]Note that the Hera project provides graphical tool support for creating conceptual and application models [Frasincar 2005].

input is a conceptual model instance (CMI), an RDF document (or repository) that contains all the data (among others references to the the media objects) underlying the conceptual model (CM) of a Web application. Thus, as already described in Section 5.1.2, the specification of a Web application's conceptual model has to be accompanied by the creation or retrieval of media instances that represent the identified concept attributes, i.e. constitute the application's underlying data. To cope with the specifics of the component-based document model, it is assumed that those media instances are annotated with appropriate metadata.

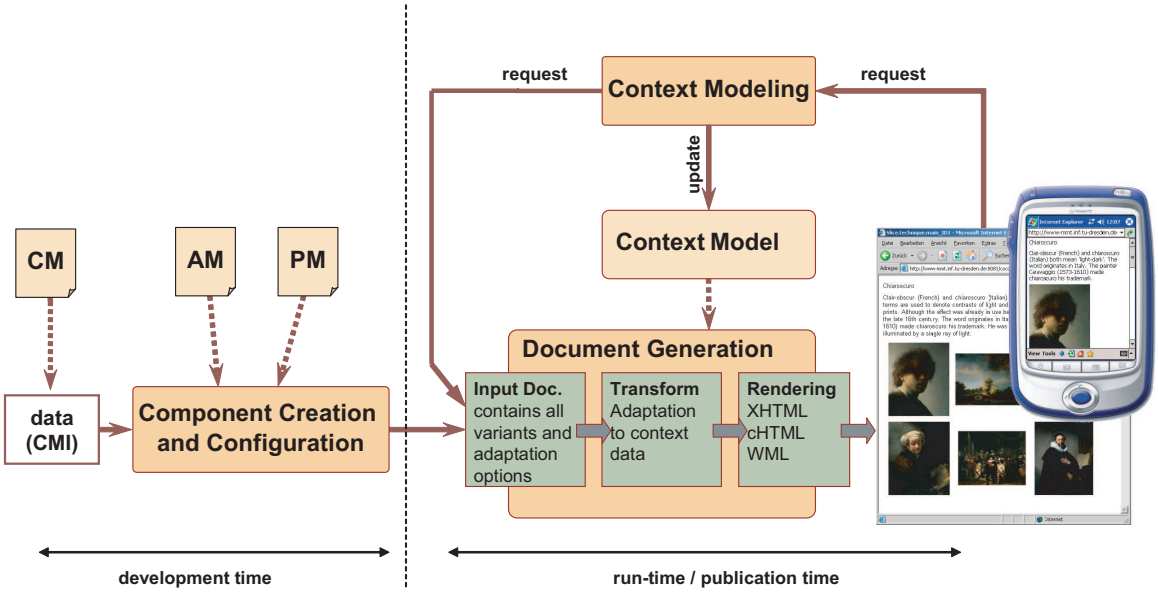

Figure 5.18: Component configuration and publication

In order to deliver Web users an adaptive hypermedia presentation on top of this data, two tasks have to be performed. First, a component-based Web presentation has to be created (see the left part of Figure 5.18). During this phase of "component creation and configuration", the appropriate models (i.e. the AM and the PM) describing the application have to be taken into account. Second, according to the actual user's request and current usage context, the created component structures have to be sent to the document generation pipeline (see the right part of Figure 5.18). While the first task is performed only once (i.e. once for each application), the second one is executed for each user request. However, both tasks can be performed automatically, i.e. no additional participation of the Web designer/developer is needed. The following sections describe the corresponding transformation steps in detail.

5.3.2.1 Model-driven Component Creation and Configuration

As described above, the first part of the overall transformation process aims at the model-driven generation of a component-based adaptive Web application. It was designed and prototypically implemented in a static and a dynamic variant, indicating whether the generated adaptive component structure consists of component instances or component templates. The first variant creates a static component-based adaptive hypermedia presentation, i.e. a network of adaptive document component instances for all the underlying data (e.g. in our running example for all instances of painting techniques, paintings, or painters) at once. Still, the term "static" refers only to the data offered by the resulting hypermedia presentation, it

can be still dynamically adjusted to different usage contexts at the later document generation process (see Section 5.3.2.2). While providing better performance for document generation, the shortcoming of this variant is that the underlying data can not be altered at run-time. On the other hand, the dynamic variant creates a structure of adaptive document component templates (each corresponding to a top-level slice) that are subsequently dynamically filled with volatile data at each request only during document generation. Since (apart from small deviations) the two transformation variants are quite similar, we describe here the static one.

This "component creation and configuration" process is parameterized by the application model (AM) and the presentation model (PM) and consists thus of two phases. The first phase takes the AM into account to convert the original data (CMI) to a component-based Web document structure (still without layout descriptors). This phase consists of two substeps.

In the first substep, a so called application model instance (AMI) is generated. It is an RDF document, an instantiation of the application model (AM) with the data available from the original conceptual model instance (CMI). For this transformation substep existing modules from the Hera Presentation Generator (HPG [Frasincar et al. 2005]) provided by the Hera project have been also utilized. However, in this particular scenario, the delivered application model instance is still unadapted, i.e. it contains appearance conditions referencing (both static and dynamic) parameters from the context model. As described later, this enables to use the various (dynamic) adaptation mechanisms provided by the component-based document format and its document generation architecture.

In the second substep, the incoming application model instance is automatically converted to a document component structure. Based on the mapping "recipe" described in Section 5.1.4, the corresponding XSLT transformation stylesheets take the analogies between slices and adaptive document components automatically into account. Whenever the AM specified appearance conditions, the resulting component structure also contains adaptation variants and selection methods (referencing the context model). Still, as the presentation model has not been considered at this stage, it does not contain layout specific attributes, yet.

In the next phase of the "component creation and configuration process", the layout attributes of the created component structure are configured according to the actual application's presentation model (PM) description. Beginning at top-level document components and visiting their subcomponents recursively, the appropriate layout descriptors are added to the meta-information section of each component's header. Since the layout manager attributes of the Hera-AMACONT PM rest upon the layout concepts of the component-based document format, this mapping is a straightforward process [Fiala et al. 2004a]. Yet, for set elements (containing a variable number of subelements depending on the actual size of the CMI) the concrete dimensions of `BoxLayout` or `GridTableLayout` layout managers have to be computed at run time.

Whenever the PM contains adaptation conditions, these are translated to component layout variants and corresponding selection methods. Thus, for each layout assignment condition defined in the PM a separate layout manager variant is created. Furthermore, a selection method according to the switch-case or the if-then-else mechanism is composed (according to Section 4.3.1). Again, all transformations are implemented as XSLT stylesheets.

Note that the output of this component creation and configuration process is a document component structure (or network) containing both adaptation variants as well as as adaptive layout descriptors.

5.3.2.2 Document Generation

As shown in the right part of Figure 5.18, the generated component structures manifest a component-based implementation of the designed hypermedia application and serve as the input data for the document generation pipeline. As described in detail in Section 4.5, they are subdued to a series of data transformations that are triggered by the user's actual request, parameterized by the current context model, and result in an adapted hypermedia presentation. Based on these transformations, Figure 5.19 shows two versions of the generated hypermedia presentation, one for desktop browsers and another one for PDAs. Note that (as specified above) the limited display size of the handheld does not allow for a tabular arrangement of painting pictures, i.e. they are displayed in a linear way.

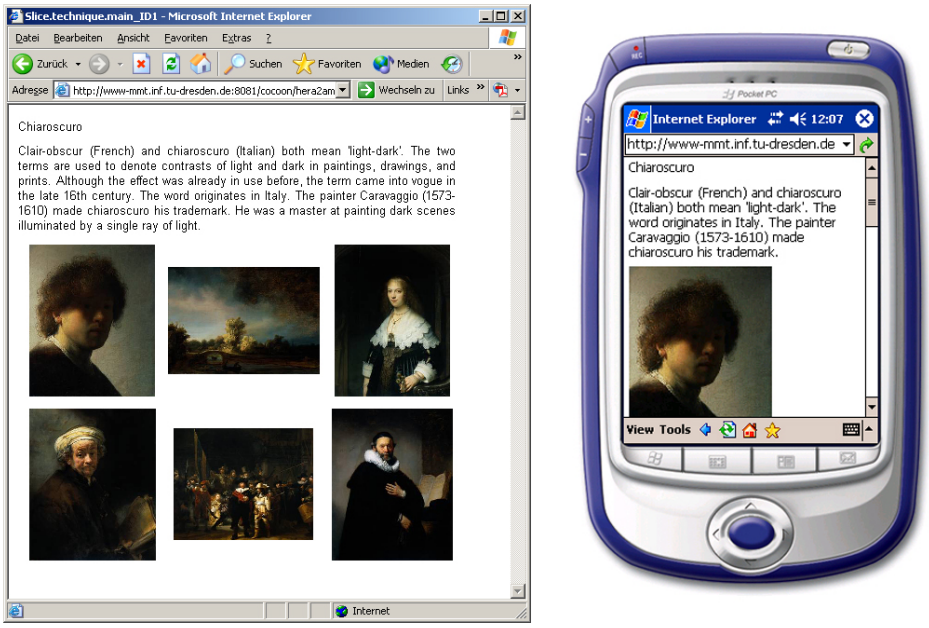

Figure 5.19: Generated hypermedia presentation [Fiala et al. 2004a]

5.3.3 Adaptivity Support

The document generation process described in Section 5.3.2.2 provides for static adaptation (or adaptability) by taking the user's actual usage context into account. However, it also supports selected issues of dynamic adaptation (adaptivity), i.e. the kind of adaptation included in the generated adaptive hypermedia presentation.

As defined in Section 2.2.1, adaptivity is the capability of a hypermedia presentation to dynamically reconfigure itself according to a dynamic usage context that is continually changing during the user's browsing session. These changes can originate from different "events", such as user interactions or even changes to the user's environment. To implement adaptivity, such events have to be acquired, the usage context has to be updated, and the Web presentation has to be regenerated, respectively.

Based on our running example application, Table 5.2 summarizes a number of characteristic examples for adaptability and adaptivity to be considered in the different models specifying a Web application. To be more accurate, it presents factors (context parameters)

that can be the basis for adaptation. While the parameters in the left column (describing the state of the user, his device, and environment) are constant for a single user session, the parameters in the right column can change (within the session) according to the user's interaction behavior. For instance, whereas the color depth of the user's device or its capability to present images influence the data to be presented statically, the available bandwidth can fluctuate and lead to a dynamic adaptation of media instances. Similarly, while the user's expertise level might be considered as constant, his knowledge on specific painters might change dynamically when browsing through the presentation described throughout this session. Finally, while the user's preferences for design elements like font types, sizes, or colors can be viewed as static factors, the current screen size can be influenced dynamically during a user session.

	Adaptability	**Adaptivity**
CM/MM	device type (`Device`) media types supported by end device (`ImageCapable`)	dynamically changing bandwidth (`Bandwidth`)
AM	user's expertise (`ExpertiseLevel`)	user's changing knowledge on painters (`Biography`)
PM	user's layout preferences (`PreferredCSS`)	resizing the browser's window (`InnerSizeX`)

Table 5.2: Adaptability/adaptivity examples across the design and implementation phases

As described in Section 4.5.3, the document generation architecture of the component-based document format utilizes an extensible context modeling framework. Providing different kinds of context modeling components (e.g. for device modeling, location modeling, or for user modeling), it allows to automatically update selected parts of the context model [Hinz and Fiala 2005]. As the focus of this chapter was put on the RDF-based specification of the Hera-AMACONT presentation model, it is now explained how dynamic adaptation specified in the PM can be realized.

When the user resizes his browser window, a JavaScript function aiming at determining the new dimensions and sending them to the server is executed. It is part of a set of client-side scripts for acquiring device capabilities and interactions which is automatically included in the presentation during document generation. Via the next HTTP request (initiated by the user's navigation) this data is sent to the server where the device model is appropriately updated. For this purpose the corresponding device modeling component utilizes the profile-diff mechanism of UAProf implemented by DELI [Butler 2003], an API provided by the Apache Web server for maintaining and updating device and user profiles based on CC/PP. For more information of this device modeling issue the reader is referred to [Hinz et al. 2006].

After updating the device model, the request is further processed and the hypermedia presentation is regenerated (see Figure 5.18). According to the steps described in Section 5.3.2.2, a new component instance is taken and subdued to the document generation pipeline so that a new presentation according to the updated context is generated. Figure 5.20 shows how the generated XHTML presentation is dynamically updated when the user resizes his browser window during browsing.

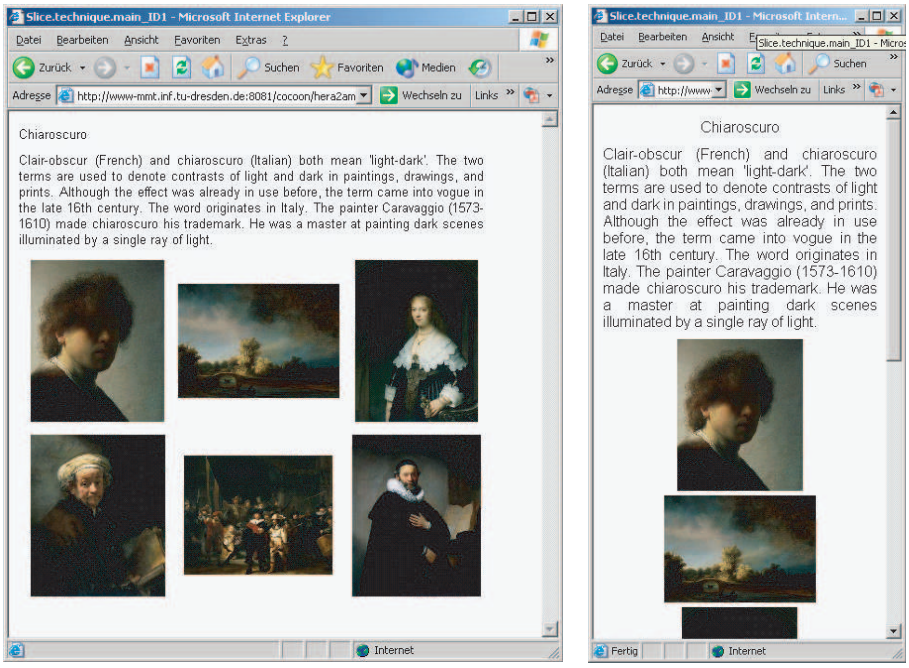

Figure 5.20: Presentation layer adaptivity

5.4 Summary and Realized Applications

This chapter dealt with the engineering process of component-based Web applications. Different application scenarios were briefly discussed, but the main focus was put on the structured development of data-driven adaptive Web presentations. It was shown how during the phases of design and implementation different aspects of adaptation can be dealt with. Furthermore, tools and mechanisms for authoring or generating component-based adaptive Web applications were explained and demonstrated. This section gives a summary of the proposed multi-stage development process and gives an overview of already realized component-based adaptive Web applications.

5.4.1 Summary of the Multi-stage Development and Document Generation Process

As a summary, Figure 5.21 recapitulates in a graphical way the different levels and possible activities involved in the development and publication process of component-based Web applications. It distinguishes between three main "levels": *design and modeling*, *component-based implementation*, and *document generation*. The numbered arrows represent selected Web engineering activities or processes. In the following they are discussed in more detail.

1. **Component Authoring:** The main focus of the activities described in this chapter lies on the development of adaptive Web applications from reusable implementation entities (document components) in a component-based way. Such a component-based Web document is schematically depicted on the left side of the second (middle) box in Figure 5.21. Though it could be created or edited by using a simple text or XML editor, the complexity of the component-based document format's underlying XML grammar (see Chapter 4) calls for visual authoring support. For this purpose the

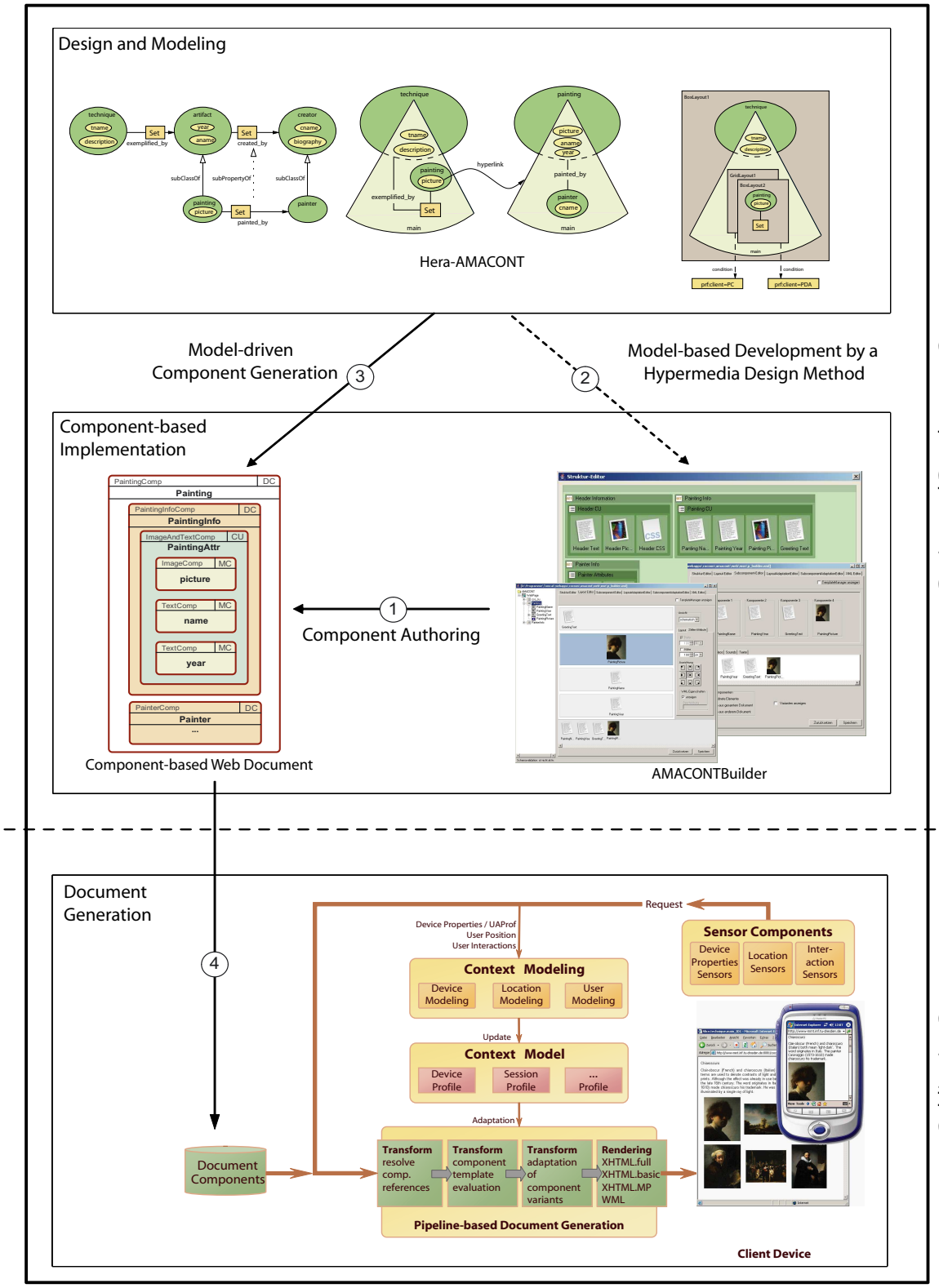

Figure 5.21: Overview of the multi-stage development process

authoring tool AMACONTBuilder (shown on the right side of the second box) was introduced. It offers a number of graphical editor modules for the implementation of adaptive Web applications by the visual creation, configuration, and interlinking of document components. In Figure 5.21 this activity of "Component Authoring" is depicted by the horizontal arrow in the second box (Nr. 1).

2. **Model-based Development by a Hypermedia Design Method:** As a format-specific tool aimed at the component-based implementation of adaptive Web applications, the AMACONTBuilder is not bound to a given process model or authoring workflow. Quite the opposite, it can be flexibly used in different development scenarios based on the requirements of the targeted application area. While for smaller Web presentations an ad-hoc approach is obviously suitable, the development of more complex adaptive Web applications necessitates to systematically take into account different application (and adaptation) concerns. As argued earlier, in this latter case component authors should proceed in a structured way, guided by an appropriate high-level model-based design of the hypermedia application.

   In Figure 5.21 this model-based component development process is depicted by the dashed arrow (Nr. 2) pointing from the level of "Design and Modeling" to the AMACONTBuilder. Considering the lines identified by a high-level model-based design methodology (in this case Hera-AMACONT) as a guideline, component authors implement adaptive Web applications by utilizing the appropriate modules of the AMACONTBuilder in a systematic way. The advantage of this approach is the usage of a graphical authoring tool that facilitates to manipulate component properties in detail. Furthermore, in a similar way it is possible to proceed according to the steps identified by another design method. Note that this approach is also typical for today's software engineering practice. Guided by a number of (mostly UML-based) models specifying the targeted software application, software developers utilize visual development platforms and selected implementation (programming) languages for the realization of the required functionality.

3. **Model-driven Component Generation:** As discussed above, the AMACONTBuilder is a flexible authoring tool that facilitates to implement component-based Web applications independent from a given design or process model. Still, the observation can be made that by exploiting the explicit semantics described in a high-level design model it might be also possible to add automation to the process of design and implementation. That is to say, the resulting development process is not only model-based but also *model-driven*. This process of "Model-driven Component Generation" is depicted by the arrow Nr. 3 and was illustrated by example of the Hera-AMACONT methodology in Section 5.3 of this chapter. Based on the RDF-based formalization of the PM, it was shown how a component-based Web application can be automatically generated based on a sequence of design models aimed at describing the Web application's semantic, navigation, and presentation behavior in a formal high-level way. The resulting Web application still contains all adaptation variants and can be thus later published for specific users, devices, and contexts.

   The main benefit of this approach is the specification of a Web application on a high-level of abstraction independent of its actual implementation. The required implementation-specific knowledge is integrated into the "model-to-component transformation process", i.e. the automatic mapping guarantees that the semantics of the design models is appropriately incorporated in the generated component-based implementation.

Moreover, the usage of Semantic Web technologies in the models also provides a number of facilities (to be investigated in the future), such as efficient model reuse, interoperability, model checking, and validation, etc.

On the other hand, the specification of an adaptive Web application in form of high-level design models does not allow to describe its implementation and presentation aspects as detailed as a "lower-level" implementation-oriented authoring tool. Thus, as also depicted in Figure 5.21, a combined approach is also possible: the model-driven generation of a component-based Web document (arrow Nr. 3) and its further "refinement" with an implementation-centric authoring tool (arrow Nr. 1).

4. **Document Generation:** While the activities mentioned above aimed at the creation or model-driven generation of component-based Web documents, the arrow Nr. 4 depicts the process of their publication to a specific Web output format. For this purpose the document generation architecture presented in Section 4.5 is utilized, that acts as a "player" of the component-based document format. The documents created (or generated) on the higher levels serve as the input of this architecture. It automatically generates a Web presentation from this input, based on available information on the current user as well as his entire usage context. The resulting presentations are delivered to the user's browser in an appropriate Web output format, such XHTML, cHTML, or WML.

   Note that this document generation process can be also considered as a specific kind of model-driven transformation. Starting from a platform-independent (and context-independent) description of a Web application based on the proposed concern-oriented component model, it generates a Web page in a platform-specific (i.e. device-specific) Web implementation format. Thus, the overall authoring and publication framework depicted in Figure 5.21 can be viewed as a multi-stage model-driven transformation process between the three abstraction levels of design and modeling, component-based implementation, and format-specific Web presentations. Yet, while the transformation process between the Hera-AMACONT models and their component-based implementation (arrow Nr. 3) is performed only once for each application, the transformation from this component-based implementation to a specific Web output format is executed for each user request.

5.4.2 Realized Applications

This chapter exemplified the design and implementation process of component-based adaptive Web presentations based on a rather small example. However, during the "evolution" of the work presented in this thesis a number of component-based Web presentations have been developed. They vary in size and complexity, address different application scenarios, and support different kinds of both static and dynamic adaptation. This section provides a representative overview of them.

Component-based MMT homepage prototype: As one of the first demonstrators selected pages of the Web site of the author's research group were realized in a device independent component-based way. Figure 5.22 depicts two versions of the research group's welcome page for desktop browsers and PDAs, respectively. The demonstrator provides mainly presentation adaptation based on the adjustment of the utilized layouts. Furthermore, the inserted media elements are also adapted (regarding their

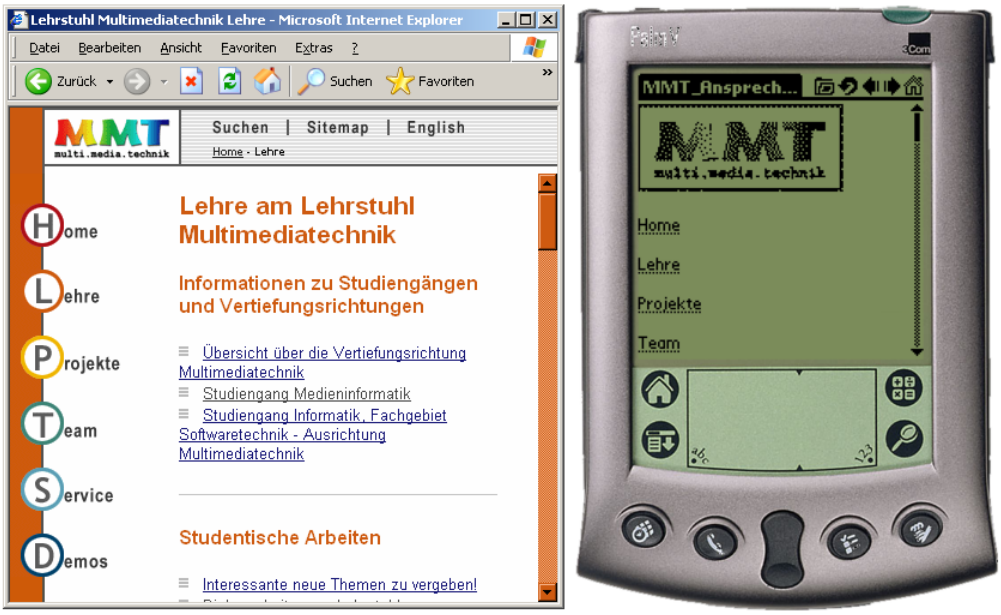

Figure 5.22: Component-based MMT homepage prototype

size, resolution, and other quality attributes) based on the capabilities of the appropriate end device. The component-based MMT homepage prototype was authored with conventional XML editors.

SoundNexus Prototype: The SoundNexus prototype is a data-driven Web presentation providing online information on music genres, bands (performers), and their albums. It was completely authored with the AMACONTBuilder to demonstrate its various editor modules and its support for creating component templates [Niederhausen 2006, Tietz 2006].

The prototype offers different kinds of content, navigation, and presentation adaptation. As an example, the list of albums shown to the user is generated dynamically, and is adjusted to his age, genre preferences, and personal voting. Furthermore, a TOP 10 list of the most popular albums (based on the votings of other users) is also provided. As also depicted in Figure 5.23, the presentation of this TOP 10 list makes use of the adaptation technique *link annotation*, i.e. the albums belonging to the current user's favorite genre are highlighted with a special icon.

Finally, the presentation of genres, bands, and albums is also adjusted to the device capabilities of the current user, i.e. different media elements with different modality (image vs. video) and media quality (large resolution image vs. small resolution image) can be used, respectively. For further information on the SoundNexus prototype the reader is referred to [Niederhausen 2006, Tietz 2006].

Model-driven Painting Gallery Prototype: This prototype demonstrates the automatic generation of a component-based adaptive Web presentation based on high-level (Hera-AMACONT based) model specifications, and is the actual implementation of the concepts described in Section 5.3. The user has the possibility to choose from a number of possible presentation model specifications and view the generated Web pages, accordingly.

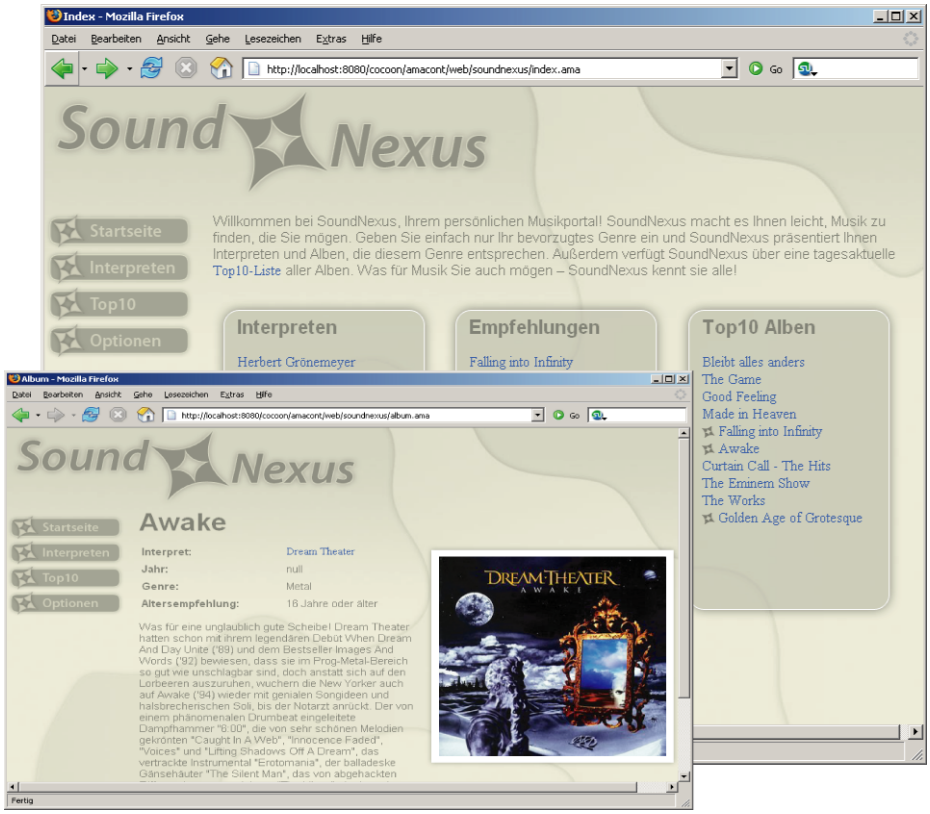

Figure 5.23: SoundNexus prototype

The prototype was presented in 2004 at the 4th International Conference of Web Engineering (ICWE04 [Fiala et al. 2004a]) and is available online at [@ICWE2004Demo]. Selected screenshots of the generated presentations were already presented in Figure 5.19 and Figure 5.20 of this chapter.

Adaptive Web Information System for presenting student works: In order to demonstrate the capabilities of the component-based document model by example of a larger Web application, an adaptive Web information system aimed at the presentation of students' works at the author's university was designed and developed [Starke 2005]. It allows students of the multimedia technology study program to upload multimedia material created in different classes and courses (e.g. pictures, video and audio material, flash presentations, etc.), as well as to navigate through this information in an adaptive way.

Besides student works, the application offers information on the current and past semesters, their courses, as well as the persons responsible for them. The application supports for presentation layer adaptation based on its users' end devices and layout preferences by adjusting the the quality (e.g. different image resolutions), the type (image vs. flash presentations), as well as the spatial arrangement of the included media elements. Furthermore, it also adapts the structure and the interconnection of pages based on security aspects (different versions internal vs. external visitors), the

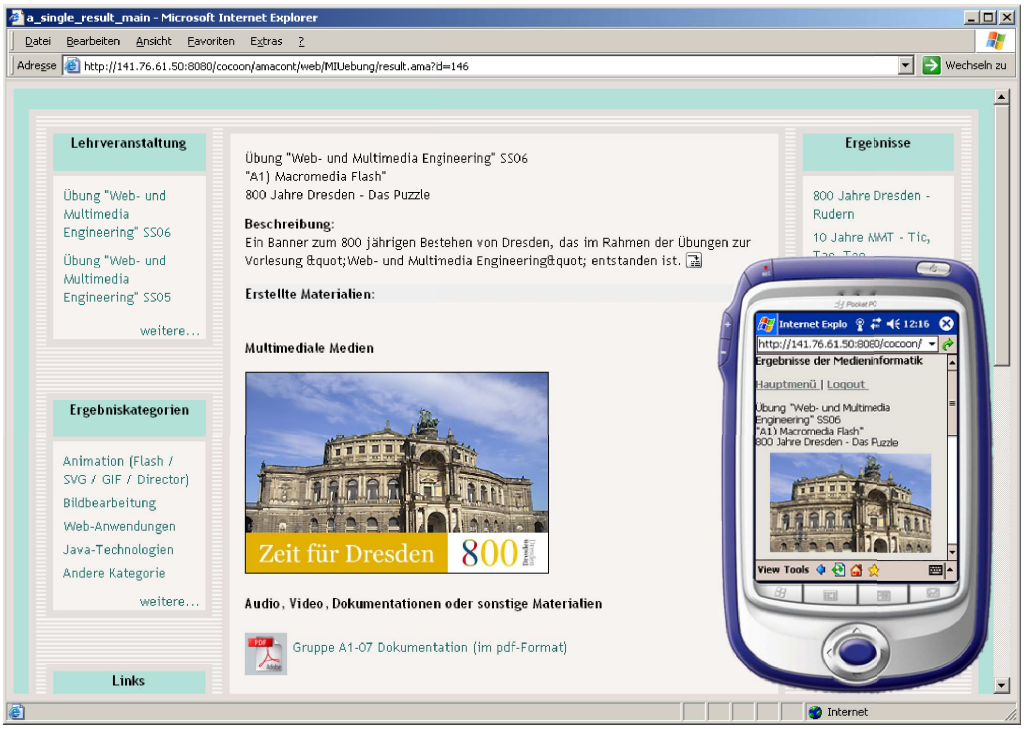

Figure 5.24: AWIS for presenting student works

visiting students' experience (their actual semester), etc. The application was successfully utilized at different courses at the author's research group. Figure 5.24 presents two screenshots of the application, one for desktop PCs and another one for PDAs. For more information on its design and realization the interested reader is referred to [Starke 2005, @kpss05].

Note that all mentioned prototype applications are available at the AMACONT project's homepage [@AMACONT].

Chapter 6

A Generic Transcoding Tool for Making Web Applications Adaptive

> *"This here's a re-search laboratory. Re-search means look again, don't it? Means they're looking for something they found once and it got away somehow, and now they got to re-search for it?"*[1]

6.1 Motivation and Introduction

In the preceding chapters of this thesis a component-based document format for adaptive dynamic Web documents was introduced. In combination with a structured authoring process and supported by a graphical authoring tool, it facilitates the efficient development of personalized ubiquitous Web presentations from reusable implementation artefacts. It was illustrated how different application aspects (concerning content, navigation, presentation, and their appropriate adaptation issues) can be systematically considered by guiding component developers through the phases of the overall Web engineering process. However, the resulting authoring framework assumes to create adaptive Web applications "from scratch", not providing sufficient support for developers (providers) who intend to *add adaptation* to an already existing Web-based system.

On the other hand, there already exists a number of formats, methodologies, and frameworks for Web application engineering. A detailed overview of the most important approaches was provided in Chapter 3. As discussed there, only some of them support (selected) issues of personalization and device dependency. Therefore, this chapter deals with the question how the lessons learned from engineering component-based adaptive Web presentations can be applied (i.e. generalized) for extending a broader range of existing Web applications by additional adaptation concerns.

In order to answer this question, it is important to investigate the way how existing Web Information Systems are typically implemented. As can be observed, they are generally based on a series of data transformations that convert some input content (in general XML data) to a hypermedia presentation in a particular implementation format, such as (X)HTML, cHTML, WML, X3D, etc. These data transformations are controlled by a specification (mostly in form of a specific XML-based document format) that dictates the application's semantic, navigational, and presentational behavior [Fiala and Houben 2005]. Such a typical pipeline-based (staged) Web presentation generation architecture is illustrated in Figure 6.1. The original content is subject to a number of transformations that subsequently lead to the desired hypermedia presentation.

Note that Web Information Systems supporting adaptation are realized in a similar way.

[1] Kurt Vonnegut, Jr.: Cat's Cradle, 1963

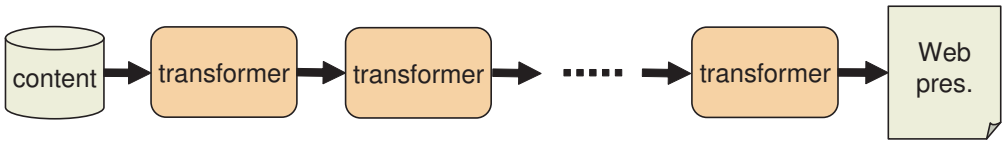

Figure 6.1: WIS implementation based on data transformations

The only difference to non-adaptive WISs is that they utilize adaptation-specific content transformation steps that are additionally parameterized by available external information describing the user's actual usage context. Usually, this information is referred to as a *user model* or a *context model*. It is typically used (referenced) by *adaptation conditions* that are attached to (i.e. contained by) fragments of the input content. A content transformation pipeline containing such an adaptation-specific transformation step is illustrated in Figure 6.2. Note, however, that there are also scenarios where several transformation steps utilize context information.

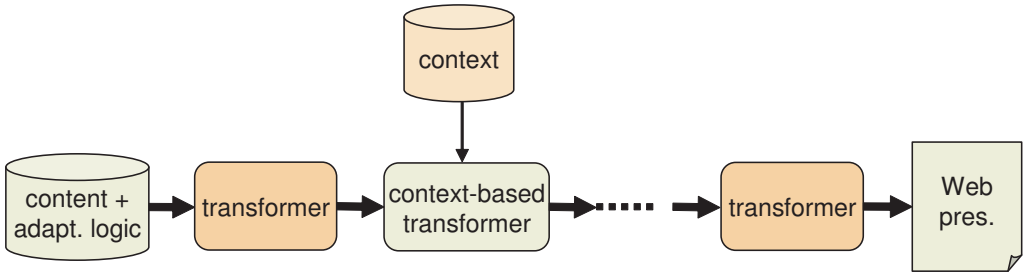

Figure 6.2: WIS implementation with adaptation

In general, most content transformations in a WIS implementation are very specific to the formats or models dictated by the underlying methodology or framework. Nevertheless, it can be recognized that adaptation-specific transformations have a lot in common. As identified by Brusilovsky's surveys [Brusilovsky 2001], they typically perform similar operations on (groups of) structured content fragments (e.g. document components, data units, slices, document nodes). Well-known adaptation operations on such structured content fragments are *conditional inclusion*, *selection*, *removal*, *insertion*, *sorting*, etc. (see Section 2.2.3). As also demonstrated by the component-based document format introduced in Chapter 4, such basic adaptation operations (transformations) can be used to realize a variety of adaptation concerns.

Given the similarity and the generic nature of such "adaptation-specific content transformations", the key observation can be made that major parts of them can be well separated from the rest of a Web application's hypermedia generation pipeline. What is more, this separate implementation of selected adaptation transformations (operations) also allows for "extracting" their configuration from the document formats describing the underlying Web application. As a consequence, it becomes possible to realize given adaptations based on *generic* transformer modules that can be appropriately controlled by an *external* configuration. Moreover, when both the implementation and appropriate configuration of adaptation operations can be separated from the original application, then it also becomes possible to *add adaptation* to an existing Web-based system.

A transformation scenario utilizing such a generic transformer module is depicted in Figure 6.3. Note that besides the information describing the current usage *context*, this trans-

former additionally takes an external *adaptation recipe* (i.e. configuration) into account, that dictates which adaptation operations it has to perform on (which selected parts of) its input content. That is to say, the specification of adaptations is not an inherent part of the input content anymore. Quite the opposite, it is "outsourced" to the generic transformer's configuration and addresses the content fragments (e.g. document components, data units, sections, etc.) to be adapted externally. Thus, the concept of document components (fragments) containing inherent adaptation descriptions can be generalized for a broader range of Web applications by the external assignment of adaptation descriptions to parts (fragments) of an arbitrary XML-based document format.

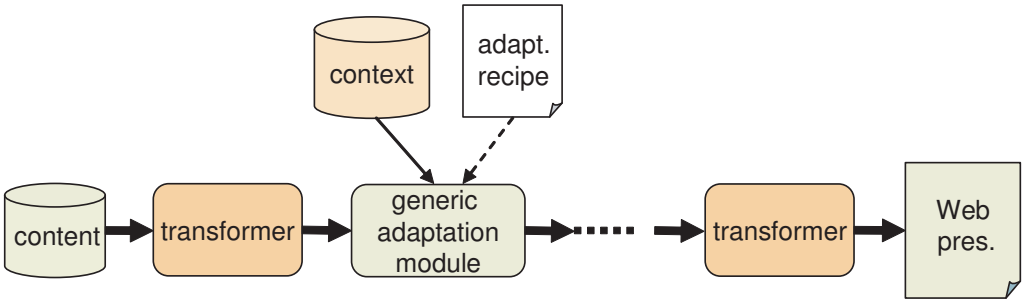

Figure 6.3: WIS implementation based on generic adaptation modules

To demonstrate this idea, this chapter introduces the Generic Adaptation Component (GAC [Fiala and Houben 2005]), a generic transcoding tool aiming at making existing Web applications adaptable and adaptive[2]. The provider of a Web Information System can use it as a stand-alone module, configure it, and integrate it into his Web site architecture. For the configuration of the GAC an RDF-based rule language is introduced, allowing to specify rules for both content adaptation and context data updates. Furthermore, a set of operations for implementing these rules will be provided.

Note that the separation of adaptation from the rest of the application might obviously result in a restricted adaptation support compared to adaptive Web applications that have been designed for adaptation from the beginning. However, it will be demonstrated that even this transformation-based "lightweight" adaptation extension can be efficiently utilized in many different application scenarios.

The rest of this chapter is structured as follows. After briefly discussing related approaches in Section 6.2, an overview of the architecture, the main functionality, and the most important application scenarios of the GAC is given (Section 6.3). Then, Section 6.4 describes central issues of the GAC's configuration in more detail, among them the requirements towards its input data, the adaptation context data it utilizes, and its RDF-based configuration language. To help the reader understand the main concepts, all these aspects are explained by a running example. The implementation details of the GAC based on the component-based document format's presentation generation architecture are introduced in Section 6.5. Finally, Section 6.6 gives a comparison of the transcoding-based adaptation approach described in this chapter and the component-based approach explained in the previous chapters, by discussing their main advantages and disadvantages.

[2]The GAC was designed and developed within the scope of a long-term collaboration between the author's research group (the AMACONT project [@AMACONT]) and the Hera research program [@HERA]. Note, however, that the GAC's basic concepts, its rule-based configuration language, and its implementation are a contribution of the author.

6.2 Existing Web Transcoding Solutions

Recently, a number of transcoding solutions for adapting Web applications have emerged. Most of them aim at adjusting HTML-based Web content to limited presentation capabilities, like those of small hand-held devices [Alam and Rahman 2003].

Many approaches utilize so-called transcoding heuristics [Bickmore et al. 1999] for the fully automatic re-authoring of Web pages. The most important characteristics of such heuristics is that they do not take into account the structure or semantics of a particular Web page, thus they provide transcoding operations that are applicable to (almost) arbitrary Web pages. The most widely used heuristics are first sentence elision, image reduction [Bickmore et al. 1999], video/audio transcoding [Smith et al. 1998], and outlining (i.e. the replacement of section headers with hyperlinks pointing to the corresponding text blocks [Hwang et al. 2002]). Furthermore, there are also approaches aimed at removing advertisements, link lists, or empty tables [Gupta et al. 2003], as well as automatically abbreviating common words [Gomes et al. 2001]. While being generally applicable, these approaches do not take into account the specific structure and the domain-specific semantics of the underlying Web content sufficiently. Furthermore, as a consequence of their heuristic nature, the result of the adaptation (and especially the quality or usability of the resulting Web pages) is often unpredictable [Hwang et al. 2003].

Barrett et al. [Barrett and Maglio 1999] define intermediaries as computational entities that operate on information as it flows along a stream, and introduce the Web Intermediaries (WBI) [Barrett et al. 1997] approach, a framework for manipulating Web information streams. Data manipulation functionality is implemented by autonomous agents that can be deployed on the server, the client, or as proxies. The approach supports four kinds of agents: monitors, editors, generators, and autonomous agents. Utilizing WBI, Hori et al. [Hori et al. 2000] present an annotation-based transcoding solution for accessing HTML documents from information appliances like PDAs, cell phones, and set-top boxes. RDF-based external annotations specifying content transformation rules such as *content alternative selection* or *page splitting hints* can be assigned to fragments of particular Web pages [Hori et al. 2002]. The main benefit of this approach is that both the structure and content of the input data can be taken into account. Furthermore, as the adaptation metadata is separated from the content itself, different application-specific adjustment scenarios are possible. However, even this approach is restricted to the transcoding of HTML content mainly based on device capabilities. There is no support for dynamic adaptation, nor for maintaining a broader range of contexts (e.g. personalized user profiles).

A similar solution based on the assignment of external transformation instructions to Web documents is provided by RDL/TT (Rule Description Language for Tree Transformations [Schaefer et al. 2002, Osterdiekhoff 2004]). Still, instead of declarative annotations, a Java-based imperative transcoding language is utilized. Again, this language focuses also primarily on the specifics of HTML-based Web documents.

Besides for device adaptation, transcoding techniques are also intensively used to make Web applications accessible for visually impaired users [Asakawa and Takagi 2000]. Again, some solutions are based on external annotations. As an example, we mention the Travel Ontology [Yesilada et al. 2004], allowing to (semi-)automatically transform Web pages to a form optimized for voice output. Aurora [Huang and Sundaresan 2000] pursues a more semantic approach and uses domain-specific schemas describing the functional semantics of Web objects to extract their content and automatically adapt it to different user requests.

Looking at related work on Web transcoding, one can see that existing approaches mainly

allow static adaptation (*adaptability*), i.e. the adjustment of Web pages to a static set of user or device parameters. Moreover, most solutions are restricted to the presentation layer of Web applications, aiming at transforming HTML pages to limited device capabilities or users' visual impairments. Still, we claim that transcoding could be used for a broader range of adaptation and personalization issues, especially for *adaptivity*, i.e. adaptation according to parameters that may change while the Web presentation is being accessed or browsed.

6.3 GAC: Generic Adaptation Component

6.3.1 GAC Overview

As mentioned above, the GAC is a generic transcoding tool aimed at adding adaptation to existing Web applications. Figure 6.4 shows how it is integrated into a typical hypermedia generation process: it processes XML-based *Web content* provided by some *Web application generator* and adjusts it to the preferences and properties of individual users and their clients. As a generic component, the GAC can perform different adaptations on its input, the recipe for which is specified by its *configuration*. This configuration consists of a set of *adaptation rules*, each dictating a different content adaptation aspect. To take (besides the input) the current usage context into account, adaptation rules can reference arbitrary parameters from the *adaptation context data*. Finally, in order to support adaptivity, the configuration also contains *update rules* allowing to manipulate this context data according to the user's navigation and interaction history.

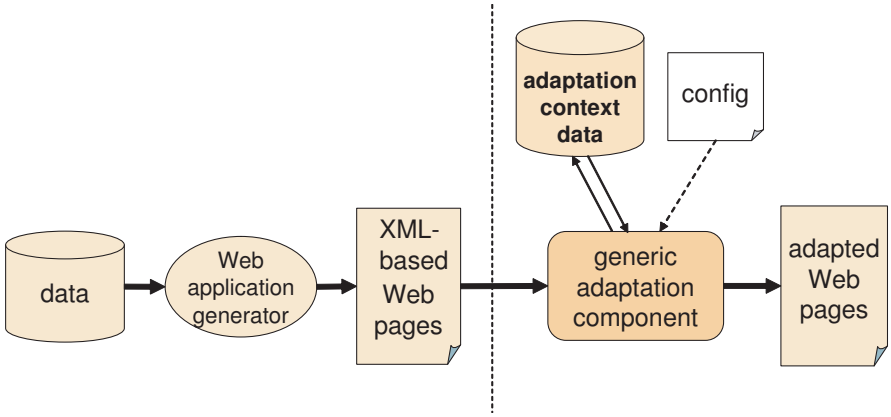

Figure 6.4: GAC abstract system overview

6.3.2 Possible Application Scenarios

As a generically applicable transcoding tool, the GAC does not make any assumptions (restrictions) to the preceding Web content generation process[3]. Quite the opposite, it can support a variety of application scenarios depending on how its XML-based input is created or generated. The following examples summarize a number of important GAC application areas.

[3]As will be described in more detail in Section 6.4.1, the only assumption is that the the original content generation process delivers data in XML format.

Transcoding static Web pages

Figure 6.5 shows a basic transcoding scenario where the "Web application generation process" acts as a traditional Web server delivering static XHTML pages. As an example, the GAC could adapt those pages to limited devices by filtering out large images, omitting videos, or eliding tags not interpretable on them. Furthermore, it could also perform user specific personalization tasks, such as changing font colors or removing information being unimportant for the user (e.g. decoration elements or links to forbidden sites).

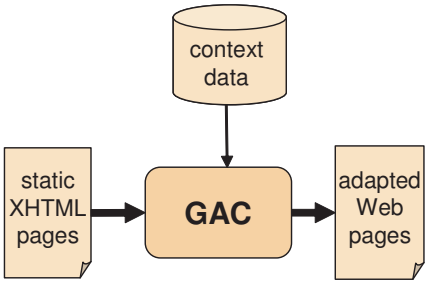

Figure 6.5: GAC scenario 1. - Transcoding static XHTML

Adaptive WIS Front-end

A more complex scenario is shown in Figure 6.6. In this case the GAC is used as the adaptive front-end of a more complex Web Information System. Based on its input data (which is typically retrieved from a data source, e.g. a database), this WIS delivers dynamically generated Web pages.

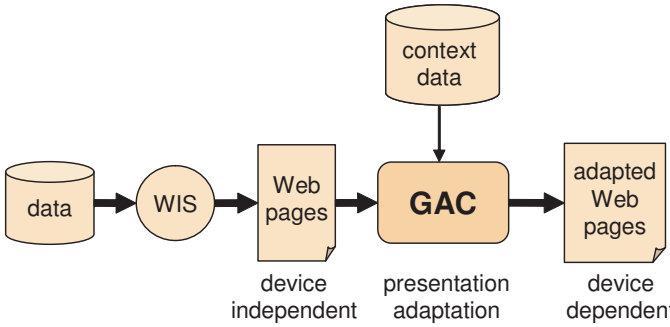

Figure 6.6: GAC scenario 2. - Adaptive WIS front-end

The WIS might be non-adaptive, or it might have already performed some content- or navigation-specific adaptations. Consequently, the role of the GAC could be to perform presentation-specific adaptation operations on those pages. This could include the adjustment of content elements to the media types and tag structures supported by a specific device or document format, the manipulation of the spatial adjustment of those content elements on the generated pages, or even the consideration of the current user's layout preferences (background images, link colors, etc.). However, the fact that the GAC operates "only" on the presentation level of the underlying Web application means that its adaptation capabilities are limited, respectively.

Transcoding with multiple GACs

While the above examples utilize only one GAC, it is possible to employ several independent GACs at different stages of the Web presentation generation process. As discussed in Section 5.3.2, a Web Information System can be efficiently realized with three layers, namely the semantic layer, the navigation layer, and the presentation layer, each responsible for its specific adaptation processes. Thus, as an extension of the preceding scenario, a non-adaptive WIS can be extended with a GAC each layer (see Figure 6.7). As a matter of course, each GAC is required to deliver data corresponding to the requirements of its successor layer.

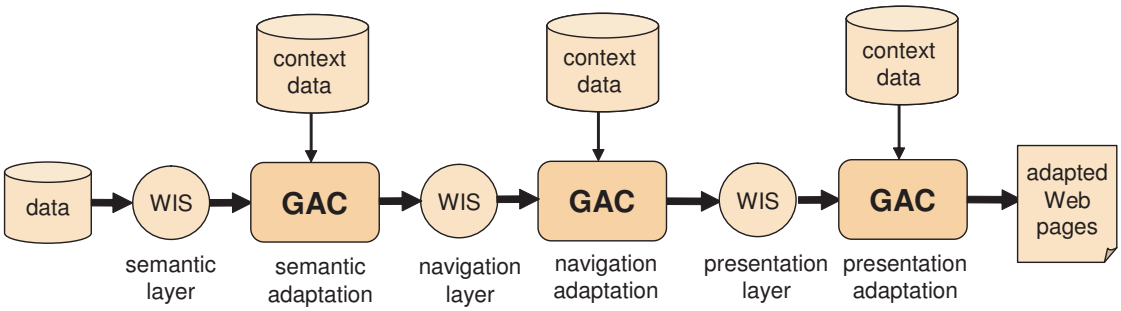

Figure 6.7: GAC scenario 3. - Adaptive WIS based on GAC pipeline

Separation of Adaptation Aspects with Multiple GACs

The GAC is a generic component aimed at performing different kinds of adaptations on its input data. Still, adaptation in a Web application is typically centered around a number of well separable independent adaptation concerns (or adaptation aspects). For instance, the navigational structure of a Web application might be adjusted according to a number of (possibly orthogonal) design concerns, such as device dependency, localization, personalization, or security. While all of these adaptations can be reduced to context-based data transformations, the provider of a Web application might need to handle them independently, i.e. by using a separate GAC for each of them.

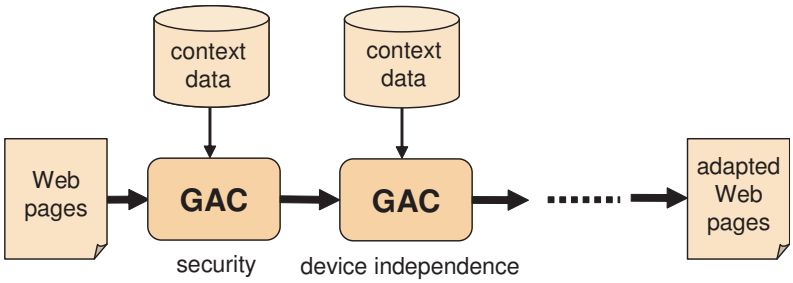

Figure 6.8: GAC scenario 4. - Separation of concerns with multiple GACs

Figure 6.8 illustrates such an adaptation scenario consisting of two GACs. While the first one performs adaptation operations supporting security issues (e.g. by hiding trustworthy content from users that are logged in as guests), the second one targets device independence (e.g. by filtering out media items being not suitable for a certain client device). Note that this separation of adaptation aspects allows providers to easily add (or remove) additional

adaptation concerns to an application without the need to change (reconfigure) or rewrite it completely. Furthermore, they can also easily reconfigure the priority of adaptation concerns by exchanging the order according to which the utilized GACs are switched in line. For more information on the advantages of this separation of concerns by using multiple GACs the reader is referred to [Casteleyn et al. 2006a, Casteleyn et al. 2006b].

Adaptivity Support

As mentioned in Section 6.2, most transcoding-based solutions provide adaptability, i.e. the adaptation based on a static user or device profile, not taking into account the user's browsing behavior. Still, modern AWISs with their increased interactivity require to support dynamic adaptation (adaptivity). As examples we mention the elision of information the (returning) user has already seen, the recommendation of links to pages the user might have become interested in, but also the dynamic reorganization of the WIS's presentation layer whenever he resizes his browser window.

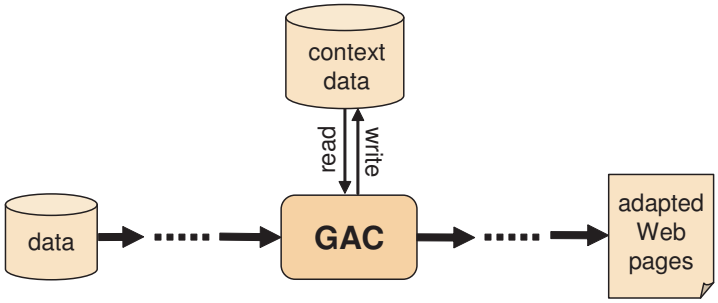

Figure 6.9: GAC scenario 5. - Support for adaptivity

To support adaptivity, the GAC has access to adaptation context data, and it can not only read but also dynamically update that context data by means of so-called *update rules* (see Figure 6.9). Consequently, the example scenarios mentioned above can be extended by even more sophisticated adaptation mechanisms. As a possible extension of the first scenario, the GAC can monitor the pages visited by users, maintain the knowledge they obtain when reading those pages, and use this knowledge to dynamically order links to related pages.

6.3.3 Running Example Overview

In order to help the reader understand the main concepts, the architecture, and the configuration of the GAC, the rest of this chapter will explain these details based on a small example application. This selected example is a dynamic Web Information System providing information about a research collaboration between the author's research group (the AMACONT project [@AMACONT]) and the Hera research program of the Vrije Universiteit Brussels [@HERA]. There are members working at the project, each characterized by a name, a CV, a picture, as well as some contact data (email address, phone number etc). They produce publications on their research efforts, which are described by a title, the name of the corresponding conference or journal, the year of publication, and an abstract.

The example Web application consists of dynamically generated Web pages in the XHTML format (see Figure 6.10). The starting page of the example application is the project homepage. It provides basic information on the project (title, description, budget information) as well as a dynamically generated link list consisting of its members' pictures as thumbnails.

By clicking on a thumbnail the user can navigate to the corresponding member's page that contains his name, contacts, CV, image, and a list of his publications. This list again leads to another page describing the corresponding publication in more detail.

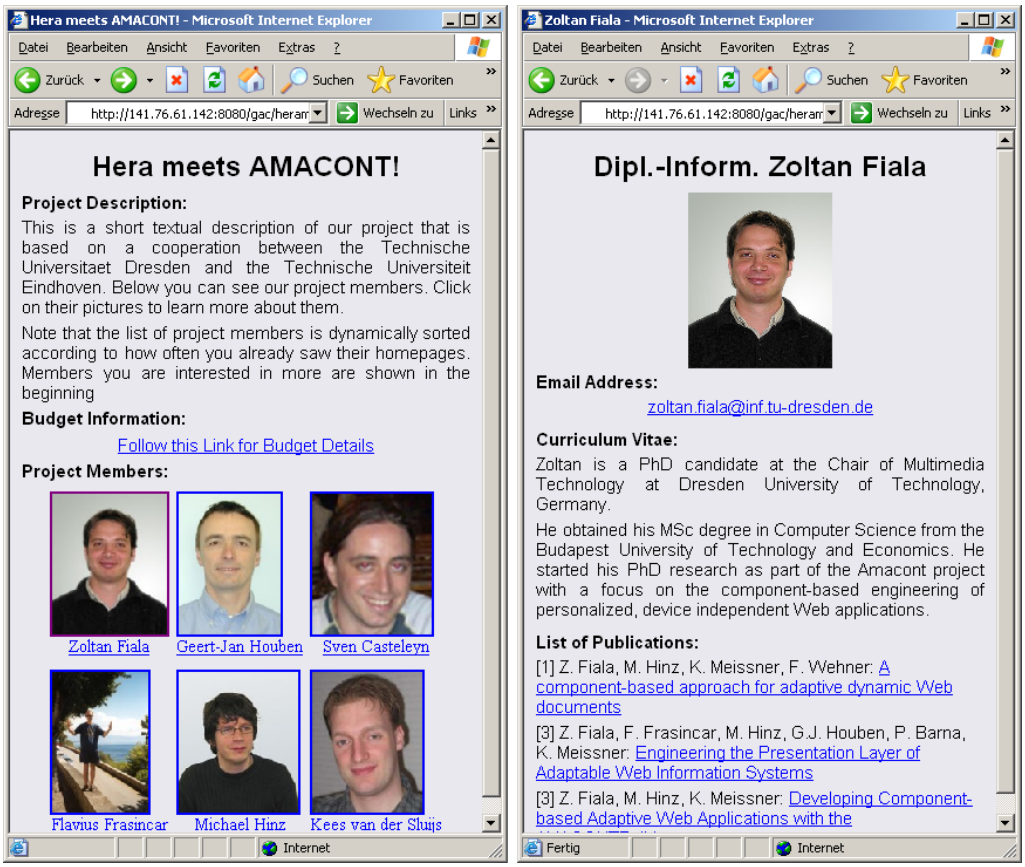

Figure 6.10: GAC running example overview

As this Web presentation does not take into account the user's preferences, nor the client's capabilities, the GAC will be used to add personalization and adaptation to it. The configuration and realization of the supported adaptations will be shown in the following.

6.4 GAC Configuration

As shown in Figure 6.4, the most crucial issues for understanding the overall architecture and functionality of the GAC are 1) the requirements towards the input content to be adapted, 2) the structure of the adaptation context data, and 3) the RDF-based rule language used to configure the corresponding adaptation operations. In accordance with the running example described above, this section explaines these issues in more detail.

6.4.1 Input Data Requirements

The GAC gets its input data from a Web presentation generation process, which can be e.g. a (part of a) legacy Web application. According to its configuration and the information describing the current usage context, it performs transformations on that input. Thus, the

transformations need access to the input content, i.e. a definition (of an interface) is required that states the structural elements to be encountered in it.

For the sake of generality, arbitrary XML-based Web content is allowed as input for the GAC. This enables the GAC to process a wide spectrum of content, both Web pages delivered in a standardized format ((X)HTML, cHTML or WML), as well as richly annotated XML data that abstracts from a specific output format and provides more information about the structure and semantics of its content. In general, the better structured and annotated the input data is, the more sophisticated adaptations can be specified.

As discussed above, the example application (to be adapted) used throughout this chapter delivers Web pages in XHTML. Furthermore, it is assumed that its designer put a focus on the separation of content and layout, and structured the generated presentation appropriately. Listing 6.1 shows the structure of a Web page presenting information about a project member.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ... >
2  <html>
3    ...
4    <body>
5      <div class="member" id="member_ID1">
6        ....
7
8        <div class="membername" id="membername_ID1">
9          <h1>Dipl-Inform. Zoltán Fiala</h1>
10       </div>
11       ...
12
13       <div class="membercv" id="membercv_ID1">
14         <p>Zoltán is a PhD student at Dresden University of Technology ...</p>
15       </div>
16       ...
17
18       <div class="publications" id="publications_ID1">
19         ....
20       </div>
21
22     </div>
23   ...
24   </body>
25 </html>
```

Listing 6.1: GAC input content example

As can be seen, the meaningful content elements (e.g. the member's name, CV, publications, etc.) to be presented are appropriately encapsulated by *div* elements and are also identified by a *class* attribute and a unique *id* attribute. As will be shown later, the presence of such content structuring tags facilitates to apply a number of content-specific adaptations. Note, however, that the usage of this structure in our example does not restrict the GAC's generality. First, it will be shown that GAC rules are independent of specific XML grammars. Second, most hypermedia document formats and WIS approaches utilize similar hierarchically ordered data containers to structure their Web content. As important analogies we mention WebML's data units [Ceri et al. 2000], Hera's slices [Frasincar et al. 2002], AHA!'s page fragments [De Bra et al. 2002], CHAMELEON's components [Wehner and Lorz 2001], HMDoc's document nodes [Westbomke and Dittrich 2002], or even the Section elements of the upcoming W3C standard XHTML 2.0 [Axelsson et al. 2004].

6.4.2 Adaptation Context Data

The adaptation operations executed by the GAC are parameterized by the *adaptation context data*. It contains up-to-date information on the user, his device, and entire usage context which the GAC has read and write access to. The structure of the GAC's adaptation context data is based on CC/PP, an RDF grammar for describing device capabilities and user preferences. As mentioned in Section 4.5.2, it represents context information based on a two-level hierarchy of *components* and their *attributes*, the latter of which are described as name-value pairs.

The GAC's configuration language (which will be described in more detail in Section 6.4.3) refers to context data parameters as variables of the form `$paramname`, where `paramname` is a literal consisting of alphanumeric characters. Moreover, it also allows for array-like context parameters in the form `$paramname[index]`, where the index of such an array is again an arbitrary literal of alphanumeric characters[4]. The usage of such array-like structures is important when handling context information which is somehow related to the underlying data (e.g. the number of times the user was been presented a given content element) and will be demonstrated in Section 6.4.3 by a number of examples.

As it will be shown later, the usage of CC/PP allows to "reuse" the context modeling framework of the modular document generation architecture presented in Section 4.5 for the GAC's implementation. An excerpt from the CC/PP-based context model of that architecture was already shown in Section 4.5.2. As mentioned there, it can be extended arbitrarily by the introduction of new profiles.

6.4.3 The Rule-based GAC Configuration Language

The GAC is controlled by its RDF-based configuration. It consists of a set of *rules* that specify the content units to be adjusted, the adaptations to be performed on them, and (in the case of adaptivity) the way the adaptation context data has to be updated. Rules are declarative, i.e. they describe *what* should be done, rather than *how*. This means, for example, that different implementations are possible for a rule specification. This is a main benefit compared to imperative approaches (e.g. RDL/TT [Schaefer et al. 2002]) that explicitly focus on a concrete implementation.

A graphical excerpt of (a part of) the RDF schema defining the GAC rule hierarchy is depicted in Figure 6.11. The top of this hierarchy is the abstract class *Rule*. A *Rule* is always bound to a *Condition*, i.e. it is activated if and only if that condition holds. A *Condition* is an arbitrary complex Boolean expression consisting of constants, parameters from the adaptation context data, as well as logical and arithmetic operations. Rules can be either *adaptation rules* or *update rules*. Whereas *adaptation rules* describe how the input data has to be transcoded, *update rules* aim at manipulating the adaptation context data. Along the lines of the example application, the following sections describe the corresponding rule types and their configuration options in more detail.

6.4.4 Adaptation Rules

Adaptation rules describe basic adaptation operations to be performed on specific parts or structures of the input content. As depicted in Figure 6.11, they all inherit from the abstract class *AdaptationRule*. They have a *selector* property that contains an XPath expression

[4] In order to stay conform with the RDF-based syntax of CC/PP, such parameters are serialized as RDF properties called "paramname_index".

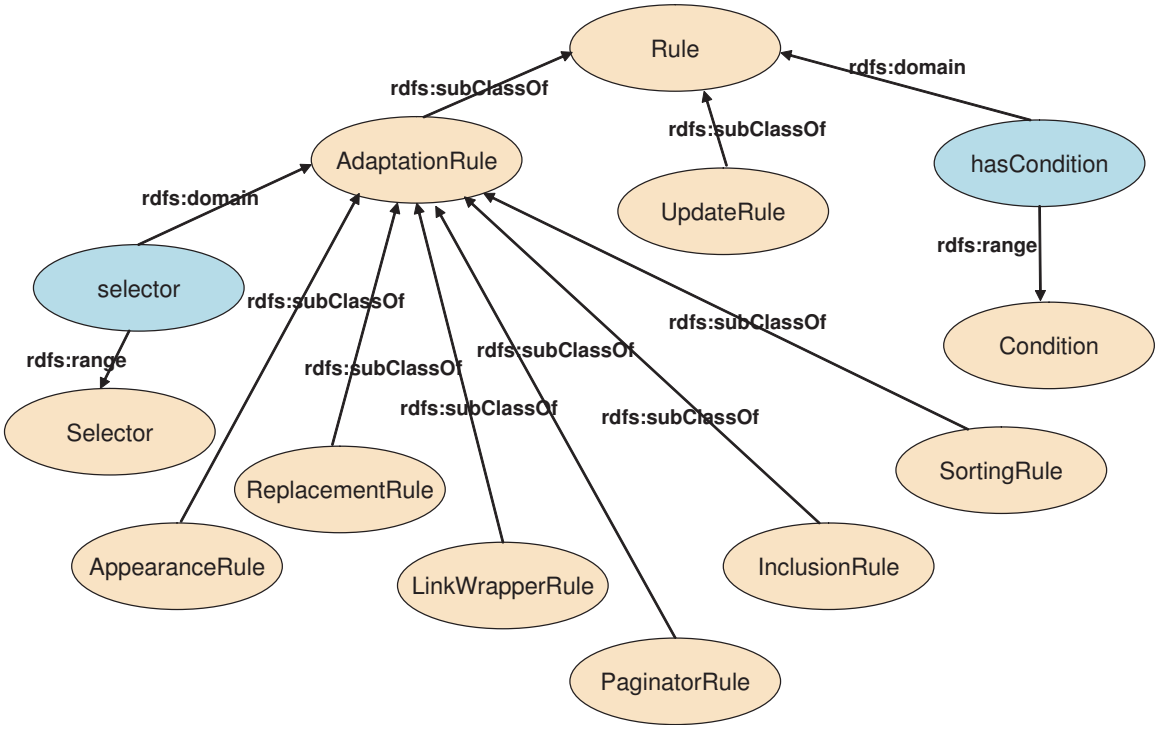

Figure 6.11: GAC rule schema excerpt

[Berglund et al. 2004] in order to unequivocally identify the parts of the XML input content to be adapted. Whenever there are several *adaptation rules* addressing the same part of the input content, they are ordered according their *priority* properties. The *priority* property of an *adaptation rule* is a non negative integer value. Its usage is optional, the default priority value is 0. *Adaptation rules* of a given GAC configuration are executed according to the descending order of their priorities, i.e. the rule with the highest priority property is processed first. *Adaptation rules* with the same priority are executed according to the order of their occurrence in the GAC's RDF-based configuration document.

Table 6.1 summarizes the properties used for parameterizing *adaptation rules*. It specifies their names, their meaning, their usage (i.e. whether they are required or optional), as well as their possible values.

Name	**Meaning**	**Usage**	**Values**
`selector`	Identifies the parts of the input content to be transcoded	required	XPath expression
`priority`	Rule priority	optional	Non negative integer

Table 6.1: Properties of an adaptation rule

In order to use *adaptation rules* (and the corresponding adaptation operations) in different application scenarios, a common set of generally applicable rule primitives were identified. Based on Brusilovsky's survey on basic methods and techniques for content and navigation adaptation [Brusilovsky 2001], as well as own results concerning presentation adapta-

tion [Fiala et al. 2004a], following rules have been selected and implemented[5]:

6.4.4.1 Appearance Rule

An *appearance rule* (Class *AppearanceRule*) realizes one of the most basic adaptation methods: the selected content is included in the output only if the associated condition is valid. Appearance rules can address arbitrary (sets of) XML elements, attributes, and text nodes. In the case when an element is selected, all of its descendant nodes are concerned, as well. An appearance rule has no additional parameters.

The two rule examples in Listing 6.2 address device dependency by adjusting the original Web content to the limited presentation capabilities of handheld devices (PDAs). While the first one omits all images from the input XHTML documents, the second one removes the lists of publications from them. The XML elements (fragments) to be elided are selected by the XPath expressions in the rules' *selector* properties (see lines 1 and 8). The conditions address parameters from the adaptation context data which are denoted with a $ sign, respectively.

```
1   <gac:AppearanceRule rdf:ID="hideImageRule" gac:selector="//img">
2     <gac:hasCondition>
3       <gac:Condition gac:when="($Device!='pda')"/>
4     </gac:hasCondition>
5   </gac:AppearanceRule>
6
7   <gac:AppearanceRule rdf:ID="hidePubListRule"
8                       gac:selector="//div[@class='publications']">
9     <gac:hasCondition>
10      <gac:Condition gac:when="($Device!='pda')"/>
11    </gac:hasCondition>
12  </gac:AppearanceRule>
```

Listing 6.2: Appearance rule example

While these appearance rules realize static adaptation, Listing 6.3 also defines one for adaptivity. The CV of a project member is shown only for users who visit that page for the first time (i.e. it is elided for returning users). The `Visited` variable from the adaptation context data is in this case an array which is parameterized by the `id` attribute of the currently selected XML element (in this case the element representing members' CVs). This way the condition is appropriately adjusted to each selected CV instance. As a matter of course, this example assumes that the user's visits to project members' homepages (and CVs) are appropriately tracked during his navigation through the Web application. The corresponding rule for updating the adaptation context data will be shown later in Section 6.4.5.

6.4.4.2 Element Filter Rule

While an appearance rule allows to include/exclude a whole XML fragment, in some cases it is meaningful to filter out only an element itself and retain its descendant elements (nodes). This is typically the case when processing XML elements aimed at formatting the underlying content, such as the *b* (bold), *i* (italics) or *u* (underline) tags of HTML. For this purpose the so-called *element filter rule* (Class *ElementFilterRule*) was developed. Its selector property

[5]Since these rules inherit from the abstract AdaptationRule, only their additional properties will be mentioned and specified in the rest of this chapter.


```
1     <gac:AppearanceRule rdf:ID="hideCVRule"
2                         gac:selector="//div[@class='membercv']">
3       <gac:hasCondition>
4         <gac:Condition gac:when="($Visited[@id]==0)"/>
5       </gac:hasCondition>
6     </gac:AppearanceRule>
```

Listing 6.3: Appearance rule example Nr. 2

can address arbitrary XML elements in the input content. Similar to appearance rules, it has no additional parameters.

The example shown in Listing 6.4 disables all hyperlinks pointing to the detailed description of publications (on the pages of project members) for users that are logged in as guests. It filters out all corresponding *a* elements but does not remove the link anchors (in this case the titles of the publications). Thus, in this particular scenario this rule implements the adaptive navigation technique *link disabling* (see Section 2.2.3).

```
1     <gac:ElementFilterRule rdf:ID="disablePubLinksRule"
2                         gac:selector="//div[@class='publications']//a">
3       <gac:hasCondition>
4         <gac:Condition gac:when="$Login=='Guest'"/>
5       </gac:hasCondition>
6     </gac:ElementFilterRule>
```

Listing 6.4: Element filter rule example

6.4.4.3 Inclusion Rules

An *inclusion rule* (Class *InclusionRule*) realizes the inverse mechanism, i.e. the insertion of external content into the processed Web document. If the associated condition is valid, XML data from a specific URI (specified by the additional *what* property of the *inclusion rule*) is included in the output document at the place determined by the rule's selector (see Table 6.2).

The data to be inserted has to be well-formed XML. Furthermore, the selector property of an inclusion rule has to address an element node. The optional *where* property defines whether the data to be included should be inserted as a preceding sibling, a following sibling, or as the first child element of the selected XML element. Its default value is *child*. Whenever the values *preceding* or *following* are used, the selected XML element may not be the input XML document's root element. Note that this generic mechanism is a powerful means of data insertion: the addressed URI can be e.g. the target URL of an HTTP request or even a complex query to a dynamic data source.

The example shown in Listing 6.5 includes an advertisement at the bottom of the project homepage for desktop PCs.

While the inclusion rule facilitates the insertion of an arbitrary XML fragment, the *attribute inclusion rule* (class AttributeInclusionRule) aims at inserting an XML attribute into an XML element. It has two additional properties that indicate the new attribute's name and value, respectively (see Table 6.3).

Name	Meaning	Usage	Values
what	URL pointing to the content to be included	required	String describing an URI
where	relative position to the selected element	optional	preceding|following|child

Table 6.2: Properties of an inclusion rule

```
1  <gac:InclusionRule rdf:ID="includeAdvertRule"
2                     gac:selector="//div[@class='project']/*[last()]">
3    <gac:hasCondition>
4      <gac:Condition gac:when="($Device=='desktop')"/>
5    </gac:hasCondition>
6    <gac:what>http://www-mmt.inf.tu-dresden.de/gacadvert</gac:what>
7    <gac:where>preceding</gac:where>
8  </gac:InclusionRule>
```

Listing 6.5: Inclusion rule example

6.4.4.4 Replacement Rules

A *replacement rule* (Class *ReplacementRule*) substitutes specific parts of the input content with an alternative value. Its selector property can address XML elements, attributes, or text nodes. The additional *with* parameter specifies the new value of the selected document part and is a string (optionally containing context data parameters). While in the case of XML elements their names are changed, attributes and text nodes get a new value.

As an example, the simple replacement rule shown in Listing 6.6 enlarges XHTML headers for users with visual impairments by exchanging H3 tags with H1 elements.

```
1    <gac:ReplacementRule rdf:ID="replaceHeaderRule"
2                         gac:selector="//H3">
3      <gac:hasCondition>
4        <gac:Condition gac:when="($visuallyimpaired=='yes')"/>
5      </gac:hasCondition>
6      <gac:with>H1</gac:with>
7    </gac:ReplacementRule>
```

Listing 6.6: Replacement rule example

While the replacement rule allows the manipulation of single XML tags, the *code replace-*

Name	Meaning	Usage	Values
name	name of the attribute to be included	required	String
value	value of the attribute to be inserted	required	String

Table 6.3: Properties of an attribute inclusion rule

Name	Meaning	Usage	Values
`with`	Indicates a new element name, attribute value or text node content	required	String

Table 6.4: Properties of a replacement rule

ment rule (Class *CodeReplacementRule*) is a slightly modified version of it that enables to replace larger XML code fragments. The rule's selector property addresses the starting XML element of the document fragment to be replaced. However, in this case the *with* property is not a simple literal, rather a URI addressing a "remote" code fragment (see Table 6.5).

Name	Meaning	Usage	Values
`with`	URI pointing to an XML fragment	required	String

Table 6.5: Properties of a code replacement rule

6.4.4.5 Link Wrapper Rule

Link wrapper rules (Class *LinkWrapperRule*) are used to manipulate the target URLs of hyperlinks found in the input content. Their main application is the modification of hyperlink targets encountered in the input XML documents. In this way users' clicks on the appropriate links can be redirected to a target defined by the GAC configurator. Furthermore, according to the adaptation context data the link wrapper rules can also add additional (personalized) request parameters to hyperlinks.

In order to identify the hyperlink references (URLs) to be manipulated the rule's selector property is used. The *toURL* property specifies the new URL (to which the link has to be redirected). Furthermore, a number of parameters in form of name/value pairs can be defined in order to attach arbitrary request parameters to the link (see Table 6.6). The optional *keepOldURL* parameter indicates that the original URL should be retained as a special request parameter of the new URL. In this case the *oldURLParamName* property determines the name of this special request parameter.

In the running example a link wrapper rule is used to realize a security-specific adaptation. Its goal is to deactivate a hyperlink that points from the project homepage to another page presenting information on the project's budget. For users that are logged in as guests, the appropriate rule redirects this link to the login page (see Listing 6.7).

```
1  <gac:LinkWrapperRule rdf:ID="loginRedirectRule"
2                       gac:selector="//a[@href='budget.html']">
3     <gac:hasCondition>
4       <gac:Condition gac:when="($LogIn=='Guest')"/>
5     </gac:hasCondition>
6     <gac:toUrl>http://www.gacexample.org/login.html</gac:toUrl>
7  </gac:LinkWrapperRule>
```

Listing 6.7: Link wrapper rule example

Name	Meaning	Usage	Values
`toUrl`	Name of the new URL	optional	String
`param`	Additional request parameter	optional	name/value pair
`keepOldURL`	Specifies whether the old URL should be retained as a specific request parameter	optional	true|false
`oldURLParamName`	Name of the request parameter containing the old URL	optional	String

Table 6.6: Properties of a link wrapper rule

Note that a link wrapper rule can not only be applied to hyperlink targets but to arbitrary XML attributes describing URLs, e.g. also to the *action* attributes of Web forms. The name link wrapper rule is used because the adjustment of hyperlinks is the rule's most often used application scenario.

6.4.4.6 Sorting Rule

Whereas the rules mentioned above address single content units (XML nodes), there are also rules adapting sets of content units, such as all child elements or all variants of a specific content unit. One of them is the *sorting rule* (Class *SortingRule*) aimed at ordering sets of XML elements according to one of their attributes. The elements to be sorted are addressed by the XPath expression specified by the rule's *selector* property. They are ordered based on the value of the attribute defined by the *sorting rule*'s additional *by* property. This can be either an XML attribute of the selected nodes, or a value of a context data parameter that is parameterized by such an attribute. The *order* attribute dictates whether the ordering is ascending or descending.

Name	Meaning	Usage	Values
`by`	Specifies the attribute according to which the sorting is performed	required	String
`order`	Specifies if the ordering is ascending or descending	required	asc|desc

Table 6.7: Properties of a sorting rule

In our running example the list of project members shown on the project homepage is sorted according to whether (and how often) the user already saw their homepages (see Listing 6.8). The attribute according to which the sorting has to be performed is defined by the adaptation context data parameter called *Visited*. It is an array that is parameterized by the unique identifiers (*id* attributes) of content elements. Members the user was already interested in are shown in the beginning of the list. Therefore, the rules order property has the value *desc*. As no condition is defined, this rule is always executed. Note that this is also

an example of adaptivity.

```
1  <gac:SortingRule rdf:ID="sortMemberRule"
2                   gac:selector="//div[@class='member']">
3    <gac:by>$Visited[@id]</gac:by>
4    <gac:order>desc</gac:order>
5  </gac:SortingRule>
```

Listing 6.8: Sorting rule example

6.4.4.7 Paginator Rule

A *paginator rule* (Class *Paginator Rule*) aims at dividing sets of XML elements into a number of smaller subsets, each containing only a predefined number of elements. It is typically applied to efficiently place content elements into a subsequent series of grouping elements (e.g. XML elements denoting container-like "grouping" structures such as pages, sections, slices or components), so that only a limited number of content elements is shown in a single group.

The rule's *selector* property addresses the XML elements, the subelements of which should be paginated. As shown in Table 6.8, these subelements are then grouped by grouping elements, the name of which is specified by the additional *group* property. The number of content units in each resulting group is determined by the *number* property. As a matter of course, the size of the last group to be created can be less than *number*, because it contains only the remaining elements.

Name	Meaning	Usage	Values
`group`	Name of the grouping element	required	String
`number`	Number of subelements in each resulting group	required	Integer

Table 6.8: Properties of a paginator rule

6.4.5 Update Rules

Unlike *adaptation rules*, *update rules* (Class *UpdateRule*) aim at updating the adaptation context data. They facilitate to change existing context parameters or to create new ones. Optionally, they can also have a *selector property*. In this case they are triggered for each selected XML node. Otherwise, they are activated only once (i.e. once each time the GAC processes an input document).

The action performed by an Update Rule is specified in its *do* property, a string describing a value assignment to an adaptation context data (ACD) parameter. The *phase* property determines whether the rule is executed before or after the transcoding process. While the *update rules* with the *phase* value *pre* are executed before adaptation rules, those with the *phase* value *post* are processed after them. That is to say, while the effects of an update rule with the *phase* value *pre* can already influence the actual adaptation transformations, the

effects of an update rule with the *phase* value *post* can be perceived only at the next page request. Since the usage of the *phase* property is optional, its default value is *pre*[6].

Name	Meaning	Usage	Values
`selector`	selects parts of the input XML content	required	String
`do`	Specifies the action to be performed	required	String
`phase`	Specifies if the update rules is performed before or after the transcoding process	optional	pre/post

Table 6.9: Properties of an update rule

Among the Adaptation Rules of our running example two rules supporting adaptivity were mentioned (see Listing 6.3 and Listing 6.8). Both require to keep track of the content elements already been visited by the user. This update mechanism can be easily supported by the following very simple Update Rule (Listing 6.9).

```
1  <gac:UpdateRule rdf:ID="trackVisitsRule"
2                  gac:selector="//div[@id]">
3    <gac:do>$Visited[@id]=true</gac:do>
4    <gac:phase>post</gac:phase>
5  </gac:UpdateRule>
```

Listing 6.9: Update rule example

The XPath expression in the rule's selector attribute identifies all content elements by addressing XML elements containing an `id` attribute. The value assignment described in the rule's *do* property tracks the fact that a content element was displayed by appropriately setting the *$Visited* variable. Note that this variable was already referred to in the adaptation rules shown in Listing 6.3 and Listing 6.8. As this rule is not associated with a condition it is always triggered. However, since its *phase* attribute has the value *post*, it is activated only after all other adaptation rules were performed. That is to say, its effect can be perceived only at the user's next page request when the appropriate adaptation rules are performed again.

To demonstrate the "interplay" of update rules and adaptation rules, Listing 6.10 illustrates a more complex adaptation strategy consisting of three rules. The first rule aims at counting a user's page visits in the example application by incrementing the NumberOfClicks variable for each requested page. Again, it has no condition associated, i.e. it is always triggered. If the value stored in the *$NumberOfClicks* variable exceeds the average number of page visits (counted for all user sessions) by a given percentage, the second rule classified the user as "interested in details". In this case the third rule (InclusionRule) includes additional information about the project at the project homepage.

6.5 Implementation Issues

In order to efficiently "reuse" the functionality of the pipeline-based document generation architecture introduced in Section 4.5, the GAC was implemented as one of its transformer

[6]Introduced by the AHAM reference model for adaptive hypermedia applications [De Bra et al. 1999], the phase attribute is widely used in systems supporting adaptivity (see Section 2.2.5).


```
1   <gac:UpdateRule rdf:ID="clickCounterRule">
2     <gac:do>$NumberOfClicks=$NumberOfClicks+1</gac:do>
3     <gac:phase>post</gac:phase>
4   </gac:UpdateRule>
5
6   <gac:UpdateRule rdf:ID="trackInterestRule">
7     <gac:hasCondition>
8       <gac:Condition gac:when="$NumberOfClicks>$AverageClicks*1.5"/>
9     </gac:hasCondition>
10    <gac:do>$InterestedInDetails='yes'</gac:do>
11    <gac:phase>post</gac:phase>
12  </gac:UpdateRule>
13
14  <gac:InclusionRule rdf:ID="includeDetailsRule"
15                     gac:selector="//div[@id='project']">
16    <gac:hasCondition>
17      <gac:Condition gac:when="$InterestedInDetails=='yes'"/>
18    </gac:hasCondition>
19    <gac:what>http://www.gac.org/AdditionalInformation/</gac:what>
20  </gac:InclusionRule>
```

Listing 6.10: Interplay of update rules and adaptation rules

modules. As illustrated in Figure 6.12, it can be used at arbitrary stages of the document generation pipeline to perform adaptation transformations on its incoming XML-based input. Each GAC transformer is configured by its RDF-based configuration document. Furthermore, each of them has read and write access to its adaptation context data (ACD), which is a part of the architecture's context model. As shown in Figure 6.12, these parts are typically disjoint. However, the current implementation allows the GACs to access arbitrary parameters from the overall context model.

Since the document generation architecture is based on the publication framework Cocoon, the GAC was developed as a custom Cocoon transformer [Ziegeler and Langham 2002] written in Java. Inheriting from Cocoon's AbstractDOMTransformer class, it performs the corresponding data transformations on the JDOM [@jdom] view of its input XML documents. In order to effectively realize adaptation and context data update rules, a Java class was implemented for each rule type introduced in Section 6.4. The corresponding implementations are optimized for performing the appropriate adaptation operations (elision, separation, inclusion, replacement, sorting, etc.) on the processed XML content.

At configuration time, the GAC processes its RDF-based configuration file and retrieves all rule definitions contained in it. For each retrieved rule it instantiates the corresponding Java rule class and sets its parameters, respectively. Sorted by their priority, the instantiated rules are registered by the *RuleManager*, a Java object maintaining a dynamic array of rule objects. This rule instantiation process is performed only once, i.e. at the time when the Web application is initialized.

At run-time the GAC is triggered by receiving XML content from its preceding data transformation components in the Cocoon pipeline. In this case the *RuleManager* is activated that triggers its registered rules one after another. Utilizing the XPath API, each rule determines the set of XML elements it is assigned to and evaluates whether the corresponding conditions hold. If they hold, the corresponding data transformations or context data updates are performed and the next rule object is invoked. After the last rule is triggered, the resulting XML document is sent to the next processing step in the Cocoon pipeline.

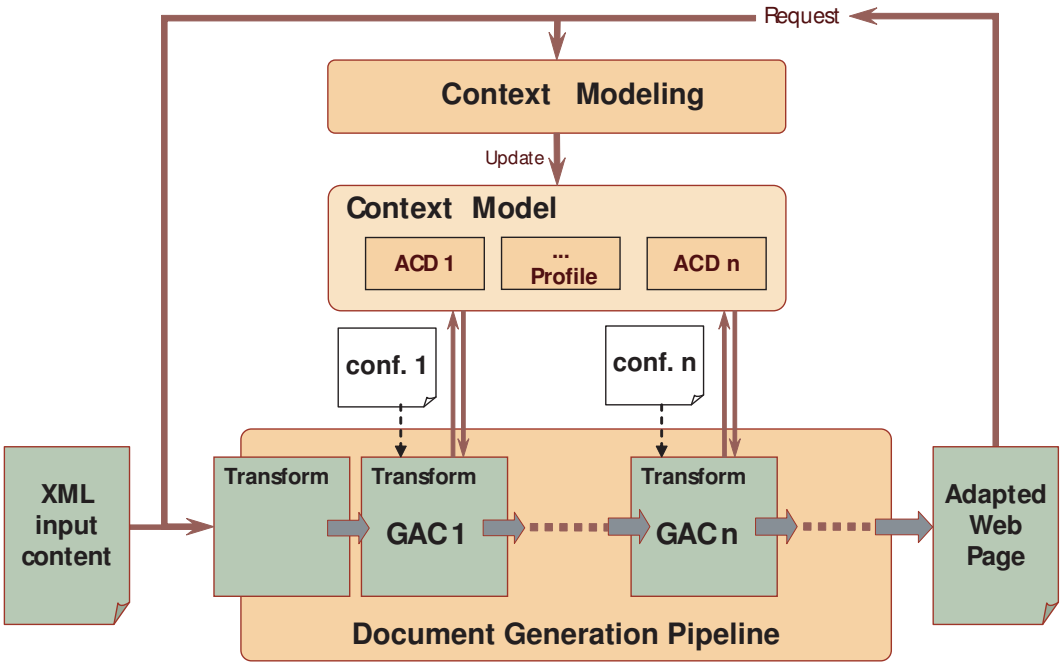

Figure 6.12: GAC implementation overview.

The context model (containing the GAC's adaptation context data) was realized based on Cocoon's so-called authentication context. It allows to store session and context information in form of arbitrary XML data structures that can be manipulated by DOM operations. Furthermore, in a later version, another implementation of the ACD repository based on the open source RDF database Sesame [Broekstra et al. 2002] was also realized . In this version the GAC implementation utilizes SeRQL (Sesame RDF Query language) for retrieving or updating this data. The usage of SeRQL allows for expressing more powerful queries (both select and update operations) on the ACD, as well as the integration of heterogeneous context data sources.

6.5.1 Running Example Implementation Configuration

While Figure 6.12 illustrated the general architecture of a GAC-based implementation, Figure 6.13 depicts the concrete GAC configuration of the running example used throughout this chapter. The input documents of the document generation pipeline are Web pages delivered in form of XHTML documents. They are subdued to two GAC transformers, each of which realizes a certain adaptation concern. For the sake of simplicity, in this scenario both GACs are configured to utilize the same adaptation context data repository.

Since the two GACs are switched immediately behind each other, note that it would be also possible to use only one GAC that is configured by all adaptation and update rules. Still, in order to support a better separation of concerns, the rules executed by each GAC are grouped according to a certain adaptation aspect. The first GAC transformer performs adaptation operations concerning device dependency. The second one performs all other adaptations that deal with user-specific personalization issues. Note that a main advantage of this separation of concerns with different GACs is the possibility to easily "plug-in" or remove a certain adaptation aspect from the entire application. Furthermore, it also nicely

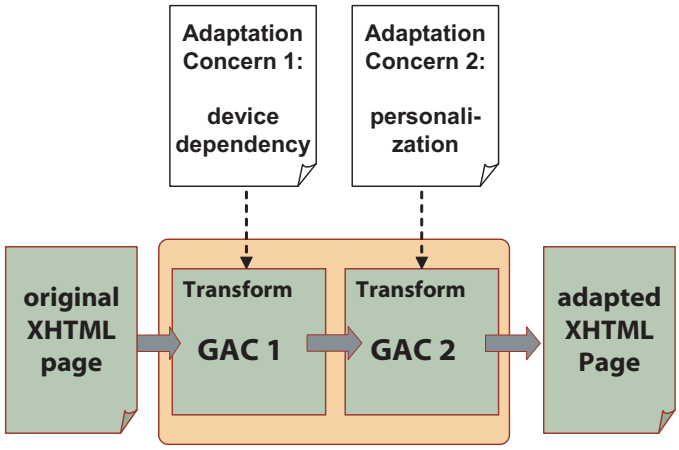

Figure 6.13: Running example implementation configuration.

corresponds to the preferred strategy of a Web designer who tries to specify independent application concerns (in this case adaptation issues) separately from each other. The usage of an implementation based on a series of GACs enables to step-by-step incorporate all these additional concerns into the original Web application at run-time.

Figure 6.14 depicts two screenshots from the "adapted versions" of the running example. The first one (on the left) shows the project overview page as it is presented on a desktop browser. Note that, according to the inclusion rule described in Listing 6.5, an "advertisement" of the GAC was inserted on its bottom. It indicates that "Adaptation on this site is powered by the GAC". Furthermore, the list of project members was also dynamically reorganized based on the user's visits to their personal homepages. The second screenshot (on the right) shows the Web page of a particular project member on a handheld device. Based on the adaptation rules addressing device dependency, both the image of the project member and the list of his publications was omitted.

6.5.2 Extensibility Issues

The GAC provides a repertoire of generic adaptation operations (rules) that are applicable on arbitrary XML input. Nevertheless, in some cases a designer might require further (e.g. more specific) adaptation rules in order to cope with a given transcoding scenario. The reason for this could be the requirement to specifically target the characteristics of a given XML format (or Web application) by the provision of a set of designated adaptation rules.

As a typical example we mention the well-known "table transform" [Hwang et al. 2003] transcoding operation. It is used for displaying large XHTML tables on handhelds either by unrolling them to a list, or by splitting them to a number of smaller tables (or subtables) with a configurable number of columns and/or rows. These operations can be reduced to a series of basic (generic) transformation operations (inclusion, omission, replacement, etc.). Yet, a provider might need a designated rule for them in order to 1) have a more "high-level" view on these operations or 2) to achieve a better performance by taking into account the specific characteristics of the given XML format (in this case XHTML) and putting all required functionality in one rule.

To realize such extensions two steps have to be performed. First, a new GAC adaptation rule (and its possibly parameters) have to be specified in the GAC rule schema. Second,

Figure 6.14: Running example screenshots

the appropriate adaptation operations to be performed on the input XML content have to be implemented. Since the extensible rule classes of the current GAC implementation allow to utilize DOM operations on the input content, programmers can easily add arbitrary functionality, either by programming it themselves or by importing existing DOM-based data transformation implementations. Furthermore, we note that the DOM (JDOM) libraries used for the GAC's implementation also allow programmers to apply any arbitrary existing XSLT stylesheet on the DOM tree of the actual document.

Besides adaptation rules, a Web developer using the GAC might also need further functionality for updating the adaptation context data. While the current GAC update rules are quite generic, we note that they are rather low-level and do not support for the expression of more specific context modeling (e.g. self-learning) algorithms. Again, the use of a Java-based implementation and the standardized CC/PP-based interface to the adaptation context data allow to easily incorporate existing context modeling components into the GAC's rule repertory.

6.6 Conclusion and Discussion

This chapter introduced the GAC, a generic transcoding tool for making XML-based Web applications adaptive. Based on the key observation that an Adaptive Web Information System is typically realized as a series of transformations, it was shown how it can *add* additional adaptation concerns to an existing Web application without the need to completely

redesign it. First, an overview of the GAC's main functionality was given, and a number of its possible application scenarios were demonstrated. Then, an RDF-based rule language for the specification of the GAC's exact functionality was presented. The different kinds of content adaptation and context data update rules were illustrated based on a running example. Finally, a prototypical implementation (developed by the author of this thesis) for realizing the GAC's main functionality was presented.

As could be demonstrated, the current GAC architecture provides the necessary functionality to incorporate different kinds of adaptation into XML-based Web applications. Still, the author is aware of the fact that, on top of this foundation, there is a need for concepts and visual GAC configuration tools that would allow Web developers to specify such additional adaptation concerns in a high-level systematic way. This issue of "GAC-based authoring" was not a focus of this work and has thus not been sufficiently addressed, yet. We note, however, that initial ideas in this direction will be discussed within the scope of future work ideas presented in Section 7.3.

Nevertheless, an interesting aspect to be discussed here is the relation of the GAC to the concern-oriented component model presented in Chapter 4. Even though the GAC was inspired by the adaptation functionality of that model and its pipeline-based document generation architecture, note that it pursues a complementary approach. Whereas a Web presentation built of document components contains adaptation definitions from the beginning in a component-based inherent way, in a Web transcoding scenario this adaptation is added to the underlying application afterwards in a rule-based manner. As a matter of course, both solutions have a number of advantages which mainly depend on the respective application scenario. The rest of this section is dedicated to the discussion of these differences in more detail.

Adaptation by Transcoding: Advantages

In general, the separate specification, storage, and implementation of "external" adaptation rules has the following advantages.

Adaptation support without content reauthoring: The providers responsible for adaptation do not need to manipulate or reauthor the input content in order to prepare it for adaptation. That is to say, XML documents from arbitrary content authors can be taken as input. Furthermore, the provider of the transcoding solution does not need to be granted write access to this original content.

Adaptation support for future Web applications: Adaptation rules may be specified also for Web pages (or in more general for Web content) that have not been even created, yet. As an example, a GAC configurator might create a rule dictating that all images in any incoming XHTML document have to be elided for handheld devices. As a matter of course, this rule is applicable for all kinds of dynamically created volatile XHTML input, too.

Flexible adaptation reconfiguration: As a consequence of the former advantage, the same input document can be used in different adaptation scenarios. It is merely the configuration of the transcoder that has to be altered in order to change the current adaptation policy. For instance, while a configuration $C1$ might be used to adjust XHTML documents to mobile devices with small displays, another configuration $C2$ could be used to transcode them for users with visual impairments. Since the corresponding adaptation specification are not intertwined with the content to be adapted,

it is easily possible to add new ones or remove existing ones without having to redesign the original application. Furthermore, the possibility to switch several transcoders in line also allows to easily change the order or priority of the applied adaptation concerns. Thus, instead of using "fixed adaptations" that are "hard-wired" to the original content, the configurator of the GAC can "adapt the adaptation" to the particularities of the given transcoding scenario.

Independent application and adaptation evolution: Due to the separate storage of transcoding rules from the original content, the introduction of a new GAC rule type or the modification (or extension) of an existing one does not require to change all corresponding input documents. Furthermore, a change to a given rule's implementation (e.g. for the sake of performance enhancements) affects only the "inner life" of the GAC, not necessitating to change the entire Web application to be adapted. That is to say, the specification and implementation of adaptation rules may evolve independently from the input documents.

Support for distributed adaptation operations: The flexible assignment of adaptation rules to documents' parts by XPath expressions allows to attach rules to multiple fragments of a document. Thus, a single adaptation rule be used to adjust different parts (components) of a Web page. This advantage is especially important because adaptation concerns (e.g. the omission of high quality pictures for devices with low presentation capabilities) are typically not pinpointed to a specific element of the input content, rather spread over several similar content elements (e.g. in this case all appropriate images) in a Web application. Consequently, such an adaptation rule addressing a number of content elements can be easily added, removed, or altered by changing only a small part of the separately stored and managed adaptation configuration.

Adaptation by Transcoding: Limitations

On the other hand, an adaptation scenario based on a transcoding solution utilizing external adaptation rules has also some limitations.

Adaptation by content filtering: The basic principle of the GAC is to perform context-dependent transformations on an already existing content stream. Thus, the adaptation operations supported by it are "restricted" to filtering and/or reorganizing this input content, not supporting to easily add new content alternatives. Even though *inclusion rules* allow to insert XML-fragments into the processed Web documents, the introduction of a new adaptation variant (e.g. a video representation of all products of an online shop for users with high bandwidth connections) typically requires a more thorough reengineering of an existing Web application.

Need for detailed knowledge of the input XML content: The efficiency of a transcoding solution significantly depends on how much the "configurator" of transcoding rules knows about the input content. Generally, the more he is familiar with the structure (e.g. the underlying data model or schema) of the input documents, the more powerful adaptations he can to express. Still, this configurator is often independent from the author(s) of the original application. Furthermore, modern Web applications are increasingly developed with high-level visual authoring tools that aim at hiding the rather low-level XML notation of the the underlying engineering approach. On the contrary, if a Web author (or designer) considers adaptation as an inherent issue of the

Web application to be created from the beginning, an appropriate process model can optimally support him.

Dependence on the semantic richness of the input content: Another important question is at which stage of the overall data transformation process a transcoding tool can be utilized. As stated above, its efficiency mainly depends on how well structured the input data is, especially how much metadata it contains. Increasingly, modern Web Information Systems utilize XML-based representations of the data they process. Starting from a well annotated data structure, they perform a number of transformations leading to a hypermedia presentation. Of course, if the data transformation pipeline cannot be "cut up" and the transcoder can only operate on top of the final presentation, then its adaptation capabilities are mostly restricted to presentation adaptation.

Dependence on the underlying architecture: As a consequence of the previous issue, the applicability of the GAC also depends on the overall architecture of the underlying Web application. That is to say, the question of where and how efficiently the GAC can be used is significantly influenced by the modularity and extensibility of that architecture. Furthermore, we mention that modern Web applications are typically developed and maintained by complex engineering frameworks, application servers, content management systems, etc. As a matter of course, a possible GAC-based extension has to be in "perfect harmony" with all these architecture components.

Low-level adaptation specification: The usage of a transcoding tool (e.g. the GAC) requires a rather low-level specification of adaptation transformations in terms of adaptation rules that operate on XML elements. Furthermore, since the GAC and its configuration language are by definition independent of a given XML grammar or methodology, it is also difficult to provide a generic graphical authoring tool for intuitively adding adaptation to any existing Web application in a high-level manner. Consequently, such a graphical tool has to be created separately (e.g. as an "add-on" or a "plug-in") for each specific authoring tool.

No inherent support for type safety: In principle, external transcoding rules can perform arbitrary transformations of the input content. Nevertheless, it might be the case that different adaptations (such as the omitting of a content element or XML tag) lead to invalid documents, i.e. to documents that already do not correspond to their original data schema. This might lead to problems when the affected documents are processed by further transformation steps, e.g. in order to be presented in a given output format. On the other hand, a component model with built-in support for adaptation permits only document transformations that assure type safety and validity.

Lacking support for component-based reuse: Separating adaptation rules from the underlying application prevents the efficient reuse of adaptable implementation artefacts in a component-wise, black-box-like manner. Reusing a part of the base application in another composition scenario also implies to extract its corresponding adaptation recipe from the original application's adaptation configuration and to "transfer" it to the new application's adaptation configuration in a possibly modified way. Thus, the detached management of application and adaptation code can lead to higher maintenance efforts.

Orphan adaptation rules: Since in a transcoding scenario the adaptation rules are fully detached from the input content, they also have to be maintained separately. However,

this separate storage of external transcoding rules (e.g. GAC rules) might lead to inconsistency problems if changes to the structure of the input documents (i.e. the documents to be adapted) are made. For instance, if the XPath expression used as the selector of a given GAC rule addresses e.g. the second subelement of a given XML element, then the inclusion of a new preceding sibling can lead to an unexpected transcoding effect. Especially, such inconsistencies arise if the person modifying a Web application is not familiar with its corresponding GAC-based adaptation configuration.

However, this problem can be reduced to the problem of so-called *orphan annotations* [Brush et al. 2001] (i.e. external annotations that can no longer be attached to a document because it was modified) known from Web annotation systems. While not being a central issue of this work, we note that there have been several approaches proposed that address this problem by utilizing so-called *robust annotation positioning* techniques. For more information on this topic the reader is referred to [Brush et al. 2001, Abe and Hori 2003].

To sum up, the separation of adaptation from the original application provides for more flexibility in terms of reconfiguration and evolution (adding, removing, or reordering adaptation aspects, inventing new adaptation rules, etc.), and the possibility to define adaptation conditions (operations) that are distributed over multiple content elements while using only one adaptation rule. On the other hand, a solution based on adaptive document components provides better reusability (in terms of adaptable content artefacts), consistency, validation support, as well as the possibility to use dedicated high-level design and authoring tools.

We remark furthermore, that a combined usage of both approaches is also conceivable. It is possible to design and implement an adaptive Web application based on reusable document components and, if demanded, flexibly add additional and rather "volatile" adaptation concerns (e.g. adaptations that are only needed in a specific deployment of the application) to it based on one or more GACs. In this case the GACs mainly serve as "customizers" that make the application runnable in a given scenario that was not foreseen at the time of its original design. Thus, while a component-based approach provides for "fixed" adaptations, GACs can be utilized for further "adapting these adaptations". Note that the modularity of the component-based document format's (staged) document generation architecture allows to plug-in GAC components at any arbitrary stage of the data transformation pipeline.

Chapter 7

Conclusion and Future Work

"It is never a mistake to say good-bye."[1]

This dissertation dealt with the design and development process of context-adaptive Web applications. It reviewed the state of the art in the fields of adaptive hypermedia and Web engineering, identified main shortcomings of existing solutions, and proposed a component-based approach for engineering adaptive Web sites. For the systematic realization of adaptive Web presentations from reusable components, a model-based authoring process was designed and constructively validated based on a prototypically realized visual component authoring tool, an automatic transformation facility for the model-driven generation of adaptive component structures, as well as a number of example applications. Finally, it was shown how the lessons learned from engineering component-based adaptive Web sites can be generalized in order to add adaptation to existing (not component-based) Web applications.

This final chapter reflects on the results of this dissertation. First, Section 7.1 recapitulates the work presented in the previous chapters. Then, Section 7.2 discusses the dissertation's main scientific contributions and achievements, but also mentions its limitations and boundaries. Finally, Section 7.3 presents possible directions of future work to be carried out on the foundations of this thesis.

7.1 Summary of the Chapters and their Contributions

Chapter 1 - Introduction

Chapter 1 introduced the background and motivation of this work. It pointed out that personalization and device independence are crucial issues of today's Web development, but also identified a number of problems and shortcomings regarding the design, development, and delivery of such adaptive Web applications. The vision of the work, the main problems to be solved, basic research theses, as well as the goals to be achieved were presented. Moreover, a short outline of the dissertation and the structure of its chapters was given.

Chapter 2 - Adaptive Hypermedia and Web-based Systems

The goal of Chapter 2 was to provide necessary background information for the reader on the field of adaptive hypermedia and Web-based applications. It stated main definitions and described the most significant methods, techniques, and application areas of hypermedia adaptation. Furthermore, the most important reference models aimed at identifying the

[1] Kurt Vonnegut, Jr.: The Books of Bokonon (from Cat's Cradle, 1963)

most common features of adaptive hypermedia and Web applications were also summarized in detail.

The bases of this chapter were fundamental works and existing surveys on the field of hypermedia, adaptation, user modeling, and context-awareness. Its contribution is a compact introduction to the field of adaptive and context-aware hypermedia systems and Web applications.

Chapter 3 - Development of Adaptive Web Applications: State of the Art

After summarizing the fundamentals of adaptive hypermedia, Chapter 3 analyzed existing approaches aimed at engineering adaptive Web applications. The focus of investigations was on two different aspects: 1) component-based and document-centric approaches aimed at the implementation of Web applications, 2) and model-based methodologies focusing on their structured design and development. From both fields the most prominent approaches were summarized. While examining related work, a special focus was put on the question of how adaptation and personalization are supported.

The basis of this chapter was a thorough examination of the recent years' scientific literature on adaptive hypermedia, Web engineering, and multimedia engineering. It pointed out the importance of declarative, component- and document-centric approaches, yet identified the lacking support for the efficient creation of adaptive multimedia Web presentations from reusable and configurable implementation entities.

Chapter 4 - A Concern-Oriented Component Model for Adaptive Web Applications

Chapter 4 presented a concern-oriented component model for component-based adaptive Web documents. First, basic requirements towards the model were discussed and the concept of declarative document components was introduced. Then, a component-based format for adaptive dynamic Web documents and its XML-based description language were presented. The different abstraction levels of document components, their support for adaptation, as well as the concept of document component templates are explained by examples. Furthermore, a modular architecture for the on-the-fly publishing and adaptation of component-based Web documents was described. Finally, the description of the model's selected benefits rounded off this chapter. The contributions of Chapter 4 are the:

- design of a component-oriented XML-based document model for dynamic adaptive Web documents;
- compact introduction and overview of the model's language constructs and its XML-based description language.

The main research results of Chapter 4 were published in a number of international publications: [Fiala et al. 2003b, Fiala et al. 2003a, Fiala and Meissner 2003, Fiala et al. 2003c, Hinz and Fiala 2005]

Chapter 5 - The Authoring Process and its Tool Support

After presenting the component-based document model, Chapter 5 dealt with the authoring process of component-based adaptive Web applications and its tool support. It discussed different application scenarios, but put a main focus on the development of data-driven adaptive

Web presentations from reusable document components. First, a possible structured development process by adopting and extending the Hera design methodology to the context of component-based Web engineering was presented. Second, the AMACONTBuilder, a modular authoring tool for the intuitive graphical authoring of component-based Web documents was introduced. Selected editor plugins of the AMACONTBuilder were presented from the point of view of component authors. Moreover, it was also shown how the overall process of design and implementation can be even automated, by automatically generating a component-based adaptive Web application from high-level design specifications in a model-driven way. Finally, the different authoring approaches were discussed, and example applications were described. The chapter's contributions are the:

- adoption of a structured design methodology for the development process of component-based adaptive Web applications;
- design and prototypical implementation of a visual authoring tool for component-based adaptive Web applications;
- extension of the Hera methodology by design and RDF(S)-based formalization of an adaptive presentation model;
- design and implementation of a solution for automatically translating high-level model specifications (design artefacts) to a component-based implementation supporting different aspects of static and dynamic adaptation;
- validation of the different authoring approaches by the development of a number of component-based adaptive Web applications.

The research results described in Chapter 5 were previously described in several publications: [Fiala et al. 2004a, Fiala et al. 2004b, Frasincar et al. 2004, Hinz and Fiala 2004, Fiala et al. 2005]

Chapter 6 - A Generic Transcoding Tool for Making Web Applications Adaptive

While the combination of the component-based document format with a structured design and authoring process provides an efficient framework for developing adaptive Web-based systems, it assumes to develop adaptive Web applications from scratch. Therefore, Chapter 6 dealt with the research question of how the lessons learned from the preceding chapters can be generalized for adding adaptation to a broader range of Web presentations. It was recognized that adaptive Web applications can be reduced to a series of data transformations and that major parts of the adaptation-specific transformations can be separated from the rest of the applications. Thus, a flexible transcoding tool called the Generic Adaptation Component (GAC) was introduced.

First, the GAC's main architecture and most important application scenarios were described. Then, its RDF-based configuration language was introduced in detail, allowing to define both content adaptation and context data update rules. To prove the concept's feasibility, it was illustrated how GACs can be configured to add adaptation to an existing Web application. As the contributions of Chapter 6 we mention the:

- design of mechanisms for separating adaptation-specific transformations from adaptive Web applications;

- design and implementation of the Generic Adaptation Component (GAC), a generic Web transcoding component for making existing XML-based Web applications adaptable and adaptive;
- specification of an RDF-based declarative language for the application-independent definition of adaptation and interaction processing rules;
- illustration of the GAC's functionality based on a running example.

The Generic Adaptation Component and its application in different transcoding scenarios were published in a number of international publications: [Fiala and Houben 2005, Houben et al. 2005, Casteleyn et al. 2006a, Casteleyn et al. 2006b].

7.2 Discussion

The research theses formulated in the introduction of the dissertation served as its main guidelines and motivation. Nevertheless, note that some of them cannot be proven directly and can thus be only evaluated as a result of extensive long-term studies. This work provides a sound foundation for such investigations. On the other hand, the research goals derived from those theses (see Section 1.2) can be concretely compared with the results of the dissertation. As also described in the previous summary of chapters, those goals could be successfully accomplished.

The main goal to develop adaptive Web applications from reusable, configurable, and adaptable implementation artefacts was achieved by the design and development of a declarative, document-centric component model. Its underlying XML-based description language supports the definition of document components that encapsulate both separate application aspects (content, structure, semantics, navigation, presentation) and their corresponding adaptation issues on different abstraction levels. Consequently, its expressivity and reusability goes far beyond the possibilities of conventional Web document formats. Based on a number of examples, it could be shown that the component model is applicable for implementing the most important hypermedia adaptation techniques. Furthermore, it was illustrated how document components can be automatically transformed to traditional Web document formats, adapted to the appropriate user and his usage context.

Second, the applicability of the proposed component model in the overall engineering process of adaptive Web applications was also successfully demonstrated. Its combination with the model-based Hera-AMACONT methodology provides significant research benefits: 1) the thorough separation of concerns and the reuse of artefacts at both design and implementation, 2) the systematic consideration of different adaptation aspects at each development step, and 3) the design-time support for presentation layer adaptation, an issue that has not been sufficiently addressed by existing methodologies, previously. With the AMACONTBuilder a flexible visual tool was introduced to facilitate different authoring scenarios independent of any one specific methodology. Furthermore, the RDF(S)-based formalization of the Hera-AMACONT presentation model enabled even the automatic, model-driven generation of a component-based implementation. The main concepts and tools of the overall multi-stage authoring and document generation process could be presented and thus be constructively validated by a number of developed adaptive Web applications.

Finally, the research goal of extending existing XML-based Web applications with (additional) adaptation concerns was also achieved. Aided by the Generic Adaptation Component, Web developers have the possibility to decouple selected adaptation operations from the rest

of a Web applications and thus to specify them at a later stage, i.e. after the Web site has already been deployed. Furthermore, as this specification of adaptation is not intertwined with the regular Web application, it allows easy re-use of adaptation configurations for different Web sites. Again, the developed concepts could be demonstrated by an implementation based on example applications.

7.2.1 Scientific Contributions

Section 7.1 already summarized the contribution of each chapter of the dissertation. As mentioned there, the most important results were also published in a number of international publications. The following list recapitulates the most important scientific contributions of the overall work.

- Development of a novel, document-centric component model for context-adaptive Web presentations with a rigorous separation of different application and adaptation concerns on multiple component levels. Thereby, extensive use of XML standards for the homogeneous description of component properties, composition, interlinking, and adaptation.
- Design of a structured, model-based authoring process for component-based adaptive Web applications. Therefore, adoption and extension of existing Web design methods to the context of component-based Web engineering. Provision of design-time support for presentation-layer adaptation in Web site modeling.
- Model-driven generation of a component-based implementation based on an RDF(S)-based high-level Web design specification. Thus, automatic combination of the advantages of model-based Web design methodologies (e.g. high-level specification, thorough separation of concerns, etc.) with the benefits of component-based implementation techniques (such as reusability, configurability, or self-adaptation).
- Provision of a mechanism for decoupling selected adaptation operations from the rest of a Web application. Design of a declarative rule-based language for the application-independent description of adaptation operations. Development of a generic tool for the addition of (both static and dynamic) adaptation concerns to existing XML-based Web applications.

7.2.2 Limitations and Boundaries

The following list summarizes limitations and boundaries of this dissertation. Some of these limitations concern the proposed approach itself. However, there are also some issues, the thorough elaboration of which would go beyond the time scope of the thesis, and thus should be tackled in form of future work.

Limitations of the component-based document format: The adaptation facilities provided by the component-based document model cover the most important adaptation techniques (see Section 4.6.4), yet assume the author to predefine all possible adaptation variants (for content, navigation, presentation) already at authoring time. A more dynamic and transparent specification or automatic computation of adaptation variants (e.g. based on rules or behavior constraints to be evaluated at run-time) are not supported, as this was not the focus of this thesis.

Limitations of the supported adaptations: The adaptation addressed in this work focuses mainly on personalization, i.e. the adjustment of Web applications to individual users, their client devices, and context. Adaptation based on the behavior of all users (e.g. by examining common browsing patterns over a longer time scale) are not supported yet, and would necessitate the introduction of a "global user model" and appropriate modeling mechanisms. Furthermore, the supported adaptations are considered to be performed on the server, client-side adaptation possibilities (e.g. on top of the emerging AJAX technology [Gamperl 2006]) should be investigated within the scope of future work.

Limitations of the utilized context model: The CC/PP-based context model is perfectly suited for describing client capabilities and user preferences, but its flat two-level hierarchy does not support for more complex model descriptions utilizing concept hierarchies and/or relationships. In order to use more sophisticated user modeling mechanisms (e.g. based on semantic or probabilistic inferences), a more complex context model and an appropriate manipulation language based on Semantic Web technologies should be used.

Limitations of the presented Web engineering process: The Web engineering process described in Chapter 5 covers the phases of designing and implementing component-based adaptive Web applications. Yet, it does not consider important Web engineering issues, such as the maintenance, continuous updating (i.e. content management), or testing adaptive Web sites. These topics imply interesting research questions and should thus be more thoroughly investigated within the scope of future work.

Limitations of the AMACONTBuilder: The editor modules of the AMACONTBuilder support the most important steps of the authoring process, yet do not cover all facilities provided by the component-based format, and should thus be further developed, respectively.

Limitations of the example applications: The approach presented in this dissertation was proven by a number of example applications (see Section 5.4.2). However, even the largest example application (the Web Information System aimed presenting students' works) is rather middle-scaled in comparison to today's Web sites both in size and the amount of its visitors. For a more thorough evaluation of the authoring process, its tool support, and the run-time performance of the resulting applications further experiments on larger-scaled Web applications would be needed.

Limitations of the GAC: Currently, the RDF-based adaptation and update rules configuring the Generic Adaptation Component have to be edited by hand. The creation of a visual GAC configuration tool, the systematic authoring process of GAC-based adaptation rules on top of an underlying Web application, as well as the automatic generation of rules based on higher-level design specifications are not supported yet, as this was not the focus of this work.

7.3 Future Work

The work presented in this thesis provides different possibilities for further work. Whereas some of them concern straightforward extensions of the presented approach and its tool support, there are also possibilities to combine the results of the thesis with other research

areas. As the most important and interesting issues (parts of which are already addressed by ongoing work) the following can be mentioned:

Extensions to the Component-based Document Model

A possible extension of the component-based document model is the integration of additional adaptation facilities (e.g. automatic pagination or link/component sorting) into its repertoire of built-in language constructs. Furthermore, the support of streaming-based media content on the level of media components and its adaptive delivery by the document generation architecture should be also investigated. Finally, the development of transformation stylesheets for converting component-based adaptive Web presentations to alternative multimedia and print formats (such as SVG, Flash, MPEG-4, or PDF) or even fat-client user interfaces (e.g. Java Swing) is also an interesting extension issue.

Extensions to the AMACONTBuilder

The AMACONTBuilder introduced in Section 5.2 offers a number of graphical editor modules to demonstrate the authoring process of component-based adaptive Web presentations. Still, it does not lay claim to be a mature development environment covering all the facilities provided by the document model. Especially, editor modules for the intuitive authoring of adaptable interaction elements (e.g. form elements or other client-side interaction components) and for the visual configuration of the different user and context modeling facilities should be developed. Furthermore, tools aimed at the graphical specification of context models would be also desirable. Moreover, a thorough evaluation of the existing tools' correctness and usability by involving a number of test authors should be performed.

Furthermore, while the AMACONTBuilder is a graphical authoring tool oriented at the specifics of the concern-oriented component model, there is a need for an integrated development suit that covers all phases of the overall Web engineering process described in Chapter 5 (also shown in Figure 5.21), as well as other Web engineerings tasks that were not considered in this dissertation, such as requirement engineering, maintenance, or the test of Web applications. The vision is a modular development suit that enables designers, developers, content creators, etc. to choose from a repertoir of modeling, implementation, and content management tools that best fit the requirements of a given application. The basis of such a development suit could be a plug-in-oriented framework (e.g. based on the OSGI standard [OSG 2005]), allowing to integrate and combine different "tool components".

Support for Collaborative, Interdisciplinary Web Authoring

Chapter 5 introduced the authoring process of adaptive Web applications from a Web engineer's (model designer or application developer) point of view. Still, the overall process of designing, developing, maintaining, and evolving Web applications involves a number of experts from different domains (Web developers, graphics and layout designers, usability experts, content editors, etc.), whose work should be appropriately supported and coordinated.

This interdisciplinary authoring approach requires facilities for defining different author roles and the assignment of specific authoring workflows to those roles. First steps in this direction have been already undertaken by a prediploma thesis supervised by the author [Niederhausen 2005b], aimed at the definition and conducted execution of self-defined authoring workflows. Still, the interplay and coordination of different authoring roles should be investigated more thoroughly within the scope of a larger development project.

Support for Collaborative Adaptive Web Applications

An important characteristics of today's World Wide Web is its evolution to a communication and cooperation medium. Web applications are increasingly used in collaborative scenarios, supporting both asynchronous (e.g. Web annotations) and synchronous ways (e.g. shared Web browsing) of communication[2]. As the participants of such collaborative scenarios use typically different mobile devices, there is a need for a proper combination of Web collaboration and Web adaptation techniques. Still, even though there exist a number of Web collaboration solutions both from industry [Lin 2003, Ulbricht 2006] and the academic field [Greenberg and Roseman 1996, Esenther 2002], there are only very few approaches (such as [Han et al. 2000]) that explicitly address adaptation and device independence.

The combination of Web collaboration and Web adaptation techniques implies the investigation of a number of challenging research issues, among them the sharing and synchronization of different user and context models between parallel Web sessions, the adaptation of Web content to the varying interaction capabilities of cooperating mobile devices, multi-device Web browsing, or even the device dependent visualization of group awareness in collaborative Web sessions. Since the work presented in this thesis provides generic and reusable facilities for developing adaptive Web applications, it appears to be ideal for combination with existing Web collaboration techniques. We note that the industry project VCS (Virtual Consulting Services [@VCSProject]) aims at adopting the research results of this thesis to the field of ubiquitous personalized co-browsing.

Authoring Support for Adaptive Rich Media Web Applications

A main trend of today's WWW is the emergence of so-called Rich Internet Applications (RIAs [Duhl 2003]). RIAs (also often referred to as *rich media applications*) embed audio, video, 3D, and other highly interactive multimedia content, and can be seen as the fusion of the interactive and multimedia user interface functionality of desktop applications with Web applications. Their development necessitates to combine methods and tools of both the Web engineering and multimedia engineering [Bailey et al. 2001, Sauer and Engels 1999] disciplines. Furthermore, the device independent delivery of RIAs makes adaptation (of content, navigation, presentation, modality, interaction) to a crucial issue. Still, existing work on adaptation engineering mainly focuses on traditional hypermedia and Web content, there are only a few approaches (e.g. [Preciado et al. 2005, Bozzon et al. 2006]) addressing the specifics of RIAs.

The flexibility of the component-based document format introduced in this thesis allows for the easy integration of rich media elements into component-based adaptive Web applications. First steps in this research direction have been already undertaken by combining its adaptation facilities with the component-based 3D user interface description language of the CONTIGRA project in [Dachselt et al. 2006]. Furthermore, the genericity of the GAC approach promises to easily address adaptation in a large number of XML-based Web applications including rich media content. For this purpose an appropriate extension of the GAC's rule-based adaptation configuration language appears to be a feasible solution.

[2]This evolution of the WWW to a "second generation" of collaborative services is also often denoted as Web 2.0 [O'Reilly 2006].

Extensions to the Generic Adaptation Component

Chapter 6 introduced the GAC as a generic transcoding tool aimed at adding adaptation to XML-based Web applications. While it illustrated its application in a number of server-side transcoding scenarios, its utilization as a client-side component appears to be also an interesting future research direction. A client-side GAC could act as a portable private adaptation component providing a personalized view on Web applications for its users. As possible use cases we mention the adaptive management of personal bookmarks and links, the maintenance of users' private comments (annotations) attached to Web page fragments, or even the "portable" realization of a Web application's adaptive presentation and adaptation layer on a mobile device. For this purpose an intuitive, "easy-to-use" GAC configuration user interface should be developed, allowing users to set it up for adjusting selected Web applications to their personal preferences.

Another possible research issue is the development of a graphical GAC configuration module that can be plugged into existing XML-based authoring tools to visually create GAC rules. A promising approach seems to be the application of the so-called "transformation by example" (or programming by demonstration) paradigm [Koyanagi et al. 2000], that enables authors to visually define transformations on fragments of an exemplary XML document and to generalize the resulting rules, so that they can be applied to other documents slightly different from the original one. A similar solution was already introduced in [Ono et al. 2002], enabling to automatically generate XSLT stylesheets based on the WYSIWYG editing of HTML documents. Its adoption for the automatic derivation (generation) of GAC transformation rules appears to be a straightforward solution.

Third, the specification and development of high-level, domain-specific GAC rules (as well as corresponding authoring support) is also an interesting extension possibility. Even though generally applicable to arbitrary XML content, note that the current GAC rules are rather low-level, i.e. the usage of domain-specific extensions would provide adaptation engineers a more high-level, application-dependent view on the given adaptation scenario. Such rules could be either automatically mapped to one or more "original" GAC rules, or, for the sake of better performance, provided with their own DOM-based implementation. A possible solution would be to bundle domain-specific (language-specific) GAC rules to so-called GAC profiles (e.g. HTML GAC profile, X3D GAC profile, etc.). Thus, developers of an adaptive Web application could easily specify, implement, publish, and exchange predefined "adaptation rule packages" dedicated to the current application domain.

Finally, the application of the GAC for model-level adaptations is also a possible research issue. Even though it was originally developed to adjust XML document instances at the level of hypermedia presentation generation, the trend to formalize (Web) design models in XML (and to generate applications based on model transformations) allows to apply it even for model adaptation. Thereby, a conceivable scenario is the combination of GACs both at model-level and instance level. While the former ones could realize static adaptation (i.e. adaptation that has to be performed only once and is thus executable at model level), the latter could implement dynamic adaptation (i.e. adaptation that has to be performed for each requested document instance separately).

Application of AOP Principles to Web Application Design

The GAC-based implementation architecture illustrated in Chapter 6 allows to easily incorporate additional (independent) adaptation concerns into a Web application. Still, whereas the GAC supports powerful adaptation operations on XML input at instance level, the com-

plexity of Web applications, and the typical distribution of adaptation throughout the application, necessitates the high-level specification of such adaptations at design level. Currently, adaptation is in most design methods specified in the form of conditions that are embedded (intertwined) in the relevant design models. Extending a design with a particular context-dependency concern therefore requires that the designer embeds for the relevant design elements the new adaptation condition(s) that result in the desired context-dependency. Though these conditions can occur at one specific place in the design (e.g. to remove a link between two concrete pages), it more frequently happens that they cannot be pinpointed to one particular element (e.g. to hide for privacy reasons all sensitive data) and need to be applied at distributed places in the design (model) [Casteleyn et al. 2006a].

A similar observation was made in the programming community, when considering different design concerns of a software application: some concerns cannot be localized to a particular class or module; instead they are inherently distributed over the whole application. Such a concern is called a *cross-cutting concern*. To cleanly separate the programming code addressing this concern from the regular application code, Aspect-Oriented Programming [Kiczales et al. 1997] was introduced. An *aspect* captures the functionality of a cross-cutting concern and can be applied at different parts of the application.

Therefore, a challenging research issue for future work is the application of principles of Aspect-Oriented Programming to Web design, allowing to separate a given Web application design from the specification of additional context-dependency design concerns. If one can easily add such functionality, it becomes possible to cleanly separate additional design aspects and describe them independently from the base application. Furthermore, by translating high-level design aspect descriptions to appropriate GAC rules, it becomes also possible to automatically generate a component-based implementation. First steps towards the application of aspect-oriented principles to Web design and their GAC-based implementation were already successfully undertaken and published in [Casteleyn et al. 2006b, Casteleyn et al. 2006a].

Bibliography

[Abdelnur et al. 1999] Alejandro Abdelnur, Elaine Chien, and Stefan Hepper. *JSR-000168 Portlet Specification (Final Release)*, `http://jcp.org/aboutJava/communityprocess/final/jsr168/`, 5 May 1999.

[Abe and Hori 2003] Mari Abe and Masahiro Hori. Robust Pointing by XPath Language: Authoring Support and Empirical Evaluation. In *2003 Symposium on Applications and the Internet (SAINT 2003), Orlando, FL*, January 2003.

[Abowd et al. 1999] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In *First International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe, Germany, September 27-29*, pages 304–307. Springer LNCS 1707, 1999.

[Abrams and Helms 2002] Marc Abrams and Jim Helms. User Interface Markup Language (UIML) Specification Draft Language Version 3.0. In *http://www.uiml.org/specs/docs/uiml30-revised-02-12-02.pdf*, February 2002.

[@AHA] AHA! - Adaptive Hypermedia for All! - project homepage. http://aha.win.tue.nl/. Date of access: 30th July 2006.

[Alam and Rahman 2003] Hassan Alam and Fuad Rahman. Web Document Manipulation for Small Screen Devices: A Review. In *Second International Workshop on Web Document Analysis (WDA2003), Edinburgh, UK*, 2003.

[Allamaraju and Brooks 2005] Subbu Allamaraju and Rex Brooks. *Web Services for Remote Portlets 1.0 Primer.* OASIS Committee Draft 1.01, `http://www.oasis-open.org/committees/download.php/11177`, 25 January 2005.

[@AMACONT] AMACONT research project homepage. http://www-mmt.inf.tu-dresden.de/english/Projekte/AMACONT/. Date of access: 30th July 2006.

[Ardissono et al. 1999] Liliana Ardissono, Luca Console, and Ilaria Torre. On the Application of Personalization Techniques to News Servers on the WWW. In *AI*IA 99:Advances in Artificial Intelligence, 6th Congress of the Italian Association for Artificial Intelligence, Bologna, Italy*, pages 261–272, 1999.

[Ardissono et al. 2002] Liliana Ardissono, Anna Goy, Giovanna Petrone, and Marino Segnan. Personalization in business-to-customer interaction. *Commun. ACM*, 45(5):52–53, 2002.

[Aroyo et al. 2003] Lora Aroyo, Paul De Bra, and Geert-Jan Houben. Embedding Information Retrieval in Adaptive Hypermedia: IR meets AHA! In *AH2003: Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems, Budapest, Hungary*, pages 63–76, 2003.

[Aroyo et al. 2005] Lora Aroyo, Lloyd Rutledge, Rogier Brussee, Paul De Bra, Peter Gorgels, Natasha Stahs, and Mettina Veenstra. Personalized Presentation and Navigation of Cultural Heritage Content. In *IEEE International Conference on Multimedia and Expo (ICME 2005), Amsterdam, The Netherlands*, pages 1589–1592, 2005.

[Asakawa and Takagi 2000] Chieko Asakawa and Hironobu Takagi. Annotation-based Transcoding for Nonvisual Web Access. In *4th International ACM/SIGCAPH Conference on Assistive Technologies (ASSETS'00), Arlington, Virginia*, pages 164–171, July 2000.

[Aßmann 2003] Uwe Aßmann. *Invasive Software Composition*. Springer-Verlag, 2003. ISBN: 3-540-44385-1.

[Aßmann 2005] Uwe Aßmann. Architectural Styles for Active Documents. *Science of Computer Programming*, 56(1-2):79–98, 2005.

[Axelsson et al. 2004] Jonny Axelsson, Beth Epperson, Masayasu Ishikawa, Shane McCarron, Ann Navarro, and Steven Pemberton. *XHTML 2.0*. W3C Working Draft, `http://www.w3.org/TR/xhtml2/`, July 2004.

[Bailey et al. 2001] B.P. Bailey, J.A. Konstan, and J.V. Carlis. DEMAIS: Designing Multimedia Applications with Interactive Storyboards. In *9th International Conference on Multimedia, Ottawa, Canada*. ACM Press, 2001.

[Barber and Badre 1998] Wendy Barber and Albert Badre. Culturability: The Merging of Culture and Usability. In *4th Conference on Human Factors and the Web, Basking Ridge, New Jersey, USA*, June 1998.

[Barrett and Maglio 1999] Rob Barrett and Paul P. Maglio. Intermediaries: An approach for manipulating information streams. *IBM Systems Journal*, 38(4):629–641, 1999.

[Barrett et al. 1997] Rob Barrett, Paul P. Maglio, and Kellem Daniel C. How to Personalize the Web. In *ACM Conference on Human Factors in Computing Systems (CHI'97), Atlanta*, pages 75–82, 1997.

[Belotti et al. 2005] Rudi Belotti, Corsin Decurtins, Michael Grossniklaus, Moira C. Norrie, and Alexios Palinginis. Interplay of Content and Context. *Journal of Web Engineering*, 4(1):57–78, 2005.

[Berglund et al. 2004] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jerome Simeon. *XML Path Language (XPath) 2.0*. W3C Working Draft, `http://www.w3.org/TR/xslt20/`, October 2004.

[Berners-Lee et al. 1992] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2):74–82, 1992.

[Berners-Lee et al. 2001] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001(May), 2001.

[Bickmore et al. 1999] Timothy W. Bickmore, Andreas Girgensohn, and Joseph W. Sullivan. Web Page Filtering and Reauthoring for Mobile Users. *Computer Journal*, 42(6):534–546, 1999.

[Bos et al. 2006] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. *Cascading Style Sheets, level 2 revision 1 CSS 2.1 Specification.* W3C Working Draft, http://www.w3.org/TR/CSS21/, 11 April 2006.

[Boyle and Encarnacion 1994] Craig Boyle and Antonio O. Encarnacion. Metadoc: An Adaptive Hypertext Reading System. *User Modeling and User-Adapted Interaction*, 4(1):1–19, 1994.

[Bozzon et al. 2006] Alessandro Bozzon, Piero Fraternali, Sara Comai, and Giovanni Toffetti Carughi. Conceptual Modeling and Code Generation for Rich Internet Applications. In *6th International Conference on Web Engineering (ICWE2006)*, 2006.

[Brickley and Guha 2003] Dan Brickley and R.V. Guha. RDF *Vocabulary Description Language 1.0:* RDF *Schema.* W3C Working Draft, http://www.w3.org/TR/rdf-schema/, 10 October 2003.

[Broekstra et al. 2002] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International Semantic Web Conference 2002*, pages 54–68, 2002.

[Brush et al. 2001] A. J. Bernheim Brush, David Bargeron, Anoop Gupta, and Jonathan J. Cadiz. Robust Annotation Positioning in Digital Documents. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 285–292. ACM Press, April 2001.

[Brusilovsky and Cooper 2002] Peter Brusilovsky and David W. Cooper. Domain, task, and user models for an adaptive hypermedia performance support system. In *International Conference on Intelligent User Interfaces (IUI 2002), San Francisco, California, USA*, pages 23–30, 2002.

[Brusilovsky et al. 1996] Peter Brusilovsky, Elmar W. Schwarz, and Gerhard Weber. A tool for developing adaptive electronic textbooks on WWW. In *World Conference on Web Society, WebNet'96, San Fransisco, CA*, pages 64–69, 1996.

[Brusilovsky 1996] Peter Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User Adapted Interaction*, 6(2-3):87–129, 1996.

[Brusilovsky 2001] Peter Brusilovsky. Adaptive Hypermedia. *User Modeling and User Adapted Interaction*, 11(1-2):87–110, 2001.

[Brusilovsky 2004] Peter Brusilovsky. Adaptive Educational Hypermedia: From generation to generation. In *4th Hellenic Conference on Information and Communication Technologies in Education, Athens, Greece*, pages 19–33, 2004.

[Bulterman et al. 2005] Dick Bulterman, Guido Grassel, Jack Jansen, Antti Koivisto, Nabil Layaida, Thierry Michel, Sjoerd Mullender, and Daniel Zucker. *Synchronized Multimedia Integration Language (SMIL 2.1).* W3C Recommendation, http://www.w3.org/TR/SMIL2/, December 2005.

[Burke 2002] Robin Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User Adapted Interaction*, 12(4):331–370, 2002.

[Bush 1945] Vannevar Bush. As We May Think. *The Atlantic Monthly*, 176 (1):101–108, 1945.

[Butler 2003] Mark H. Butler. DELI, an open-source CC/PP and UAProf servlet API. In *http://sourceforge.net/projects/delicon/*, November 2003.

[Casteleyn and De Troyer 2002] Sven Casteleyn and Olga De Troyer. Exploiting Link Types during the Web Site Design Process to Enhance Usability of Web Sites. In *Second International Workshop on Web-Oriented Software Technology (IWWOST 2002)*, 2002.

[Casteleyn et al. 2003] Sven Casteleyn, Olge De Troyer, and Saar Brockmans. Design Time Support for Adaptive Behavior in Web Sites. In *18th ACM Symposium on Applied Computing*, pages 1222–1228. ACM Press, March 2003.

[Casteleyn et al. 2006a] Sven Casteleyn, Zoltán Fiala, Geert-Jan Houben, and Kees van der Sluijs. Considering Additional Adaptation Concerns in the Design of Web Applications. In *Adaptive Hypermedia and Adaptive Web-Based Systems 2006 (AH2006)*. Springer, June 2006.

[Casteleyn et al. 2006b] Sven Casteleyn, Zoltán Fiala, Geert-Jan Houben, and Kees van der Sluijs. From Adaptation Engineering Towards Aspect-Oriented Context-Dependency. In *Fifteenth International Conference on the World Wide Web (WWW2006), Poster session*. ACM, May 2006.

[Casteleyn 2005] Sven Casteleyn. *Designer Specified Self Re-organizing Websites*. PhD thesis, Vrije Universiteit Brussel, September 2005.

[Ceri et al. 2000] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In *9th International Conference on the World Wide Web (WWW9), Amsterdam*, May 2000.

[Ceri et al. 1999] Stefano Ceri, Piero Fraternalie, and Stefano Paraboschi. Data-Driven One-to-One Web Site Generation for Data Intensive Applications. In *25th International Conference on Very Large Data Bases*, pages 615–626. Morgan Kaufman, 1999.

[Ceri et al. 2003a] Stefano Ceri, Florian Daniel, and Maristella Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *WISE - MMIS'03 Workshop (Mobile Multi-channel Information Systems)*, pages 225–233, 2003.

[Ceri et al. 2003b] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2003. ISBN: 1-558-60843-5.

[@CHAMELEON] CHAMELEON project homepage. http://www-mmt.inf.tu-dresden.de/english/Projekte/CHAMELEON/. Date of access: 30th July 2006.

[Chen 1975] Peter Pin-Shan Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1975.

[Chevchenko 2003] Elena Chevchenko. Entwicklung von Werkzeugmodulen für die Bearbeitung von Kurskomponenten und -strukturen in TeachML-Dokumenten. Master's thesis, Technische Universität Dresden, 2003.

[Cheverst et al. 2000] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In *CHI 2000 Conference on Human Factors in Computing Systems, The Hague, The Netherlands*, pages 17–24, 2000.

[Chisholm and Vanderheiden 1999] Wendy Chisholm and Gregg Vanderheiden. *Web Content Accessibility Guidelines 1.0.* W3C Recommendation, `http://www.w3.org/TR/WAI-WEBCONTENT`, 5 May 1999.

[@Consensus] Consensus project homepage. http://www.consensus-online.org. Date of access: 23th September 2006.

[Conallen 2000] Jim Conallen. *Building Web Applications with UML.* Addison-Wesley, 2000. ISBN: 2-016-1577-0.

[Costagliola et al. 2002] Gennaro Costagliola, Filomena Ferrucci, and Rita Francese. Web engineering: Models and methodologies for the design of hypermedia applications. *Handbook of Software Engineering & Knowledge Engineering, Emerging Technologies, World Scientific*, pages 181–199, 2002.

[Cowan and de Lu 1995] Donald D. Cowan and Carlos José Pereira de Lu. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. *IEEE Transactions on Software Engineering*, 21(3):229–243, 1995.

[Cranor et al. 2002] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification.* W3C Recommendation, `http://www.w3.org/TR/P3P/`, 16 April 2002.

[Dachselt and Rukzio 2003] Raimund Dachselt and Enrico Rukzio. Behavior3D: an XML-based framework for 3D graphics behavior. In *Eighth International Conference on 3D Web Technology,Web3D 2003, Saint Malo, France*, pages 101–112, March 2003.

[Dachselt et al. 2002] Raimund Dachselt, Michael Hinz, and Klaus Meißner. CONTIGRA: an XML-based architecture for component-oriented 3D applications. In *7th International Conference on 3D Web Technology (Web3D 2002)*, pages 155–163, Februar 2002.

[Dachselt et al. 2006] Raimund Dachselt, Michael Hinz, and Stefan Pietschmann. Using the Amacont Architecture for Flexible Adaptation of 3D Web Applications. In *11th International Conference on 3D Web Technology (Web3D 2006), Columbia, Maryland, USA*, pages 75–84, April 2006.

[Dachselt 2004] Raimund Dachselt. *Eine deklarative Komponentenarchitektur und Interaktionsbausteine für dreidimensionale multimediale Anwendungen.* Dissertation. Der Andere Verlag, 2004. ISBN: 3-89959-271-9.

[Dale et al. 1998] Robert Dale, Jon Oberlander, Maria Milosavljevic, and Alistair Knott. Integrating natural language generation and hypertext to produce dynamic documents. *Interacting with Computers*, 11(2):109–135, 1998.

[Dart 1999] Susan Dart. Change Management: Containing the Web Crisis. In *ICS Workshop on Web Engineering*, 1999.

[Davis and Huttenlocher 1995] Jim Davis and Dan Huttenlocher. *Conote System Overview.* http://www.cs.cornell.edu/home/dph/annotation/annotations.html, `http://www.cs.cornell.edu/home/dph/annotation/annotations.html`, December 1995.

[Dayal 1988] Umeshwar Dayal. Active Database management systems. In *Third International Conference on Data and Knowledge Bases*, pages 150–169. Morgan Kaufmann, 1988.

[Díaz et al. 1995] Alicia Díaz, Tomás Isakowitz, Vanesa Maiorana, and Gabriel Gilabert. RMC: a tool to design WWW applications. In *Fourth International World Wide Web Conference (WWW4)*, 1995.

[De Bra and Ruiter 2001] Paul De Bra and J.P. Ruiter. AHA! Adaptive Hypermedia for All. In *WebNet2001, World Conference on the WWW and Internet*, pages 262–268, October 2001.

[De Bra et al. 1999] Paul De Bra, Geert-Jan Houben, and Hongjing Wu. AHAM: A Dexter-Based Reference Model for Adaptive Hypermedia. In *10th ACM Conference on Hypertext and Hypermedia (HYPERTEXT '99), Darmstadt, Germany*, pages 147–156. ACM, February 1999.

[De Bra et al. 2002] Paul De Bra, Ad Aerts, David Smits, and Natalia Stash. AHA! Version 2.0, More Adaptation Flexibility for Authors. In *AACE ELearn'2002 conference*, pages 240–246, 2002.

[De Bra et al. 2004] Paul De Bra, Lora Aroyo, and Vadim Chepegin. The Next Big Thing: Adaptive Web-based Systems. *Journal of Digital Information*, 5(1), 2004.

[De Troyer and Casteleyn 2004] Olga De Troyer and Sven Casteleyn. Designing Localized Web Sites. In *5th International Conference on Web Information Systems Engineering (WISE2004), Brisbane, Australia*, pages 547–558, 2004.

[De Troyer 2001] Olga De Troyer. Audience-driven Web Design. In *Information Modeling in the New Millennium*, pages 442–462. IDEA GroupPublishing, 2001.

[Denoue and Vignollet 2000] Laurent Denoue and Laurence Vignollet. An annotation tool for Web browsers and its applications to information retrieval. In *6th Conference on Content-Based Multimedia Information Access (RIAO2000)*, 2000.

[Denoue 1999] Laurent Denoue. Adding metadata to improve retrieval: Yet Another Web Annotation System. Technical report, University of Savoie, 1999.

[DeRose et al. 2002] Steven DeRose, Ron Daniel, Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. *XML Pointer Language (Xpointer).* W3C Working Draft, `http://www.w3.org/TR/xptr/`, August 2002.

[Deshpande et al. 2002] Yogesh Deshpande, San Murugesan, Athula Ginige, Steve Hansen, Daniel Schwabe, Martin Gaedke, and Bebo White. Web Engineering. *Journal of Web Engineering*, 1(1):3–17, 2002.

[Dey 2001] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, February 2001.

[Dolog et al. 2003] Peter Dolog, Rita Gavriloaie, Wolfgang Nejdl, and Jan Brase. Integrating Adaptive Hypermedia Techniques and Open RDF-based Environments. In *Twelfth International World Wide Web Conference (WWW2003), Alternate Paper Tracks*, 2003.

[Dubinko 2004] Micah Dubinko. *XForms Essentials.* O'Reilly & Associates, 2004. ISBN: 0-596-00369-2.

[Duhl 2003] J. Duhl. Rich Internet Applications. In *IDC white papers, http://www.idc.com*, 2003.

[Esenther 2002] Alan W. Esenther. Instant Co-Browsing: Lightweight Real-time Collaborative Web Browsing. In *The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA*, 2002.

[Evers and Day 1997] Vanessa Evers and Donald L. Day. The Role of Culture in Interface Acceptance. In *IFIP TC13 Interantional Conference on Human-Computer Interaction (INTERACT '97), Sydney, Australia*, pages 260–267, 1997.

[Fallside and Walmsley 2004] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition.* W3C Recommendation, `http://www.w3.org/TR/xmlschema-0/`, October 2004.

[Ferraiolo and Jackson 2003] Jon Ferraiolo and Dean Jackson. *Scalable Vector Graphics (SVG) 1.1 Specification.* W3C Recommendation, `http://www.w3.org/TR/SVG11/`, January 2003.

[Fiala and Houben 2005] Zoltán Fiala and Geert-Jan Houben. A Generic Transcoding Tool for Making Web Applications Adaptive. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, pages 15–20. FEUP, June 2005.

[Fiala and Meissner 2003] Zoltán Fiala and Klaus Meissner. Annotating Virtual Web Documents with DynamicMarks. In *Workshop XML Technologien für das Semantic Web (XSW 2003), Berliner XML Tage*, pages 67–77, October 2003.

[Fiala et al. 2003a] Zoltán Fiala, Michael Hinz, Klaus Meißner, and Frank Wehner. A Component-based Approach for Adaptive Dynamic Web Documents. *Journal of Web Engineering, Rinton Press*, 2(1&2):058–073, September 2003.

[Fiala et al. 2003b] Zoltán Fiala, Michael Hinz, Klaus Meissner, and Frank Wehner. A Component-based Approach for Adaptive Dynamic Web Documents. In *Twelfth International Conference on the World Wide Web (WWW2003), Poster session.* ACM, May 2003.

[Fiala et al. 2003c] Zoltán Fiala, Michael Hinz, and Frank Wehner. An XML-based Component Architecture for Personalized Adaptive Web Applications. In *Workshop Personalisierung mittels XML-Technologien, Berliner XML Tage*, pages 370–378, October 2003.

[Fiala et al. 2004a] Zoltán Fiala, Flavius Frasincar, Michael Hinz, Geert-Jan Houben, Peter Barna, and Klaus Meissner. Engineering the Presentation Layer of Adaptable Web Information Systems. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, pages 459–472. Springer LNCS 3140, July 2004.

[Fiala et al. 2004b] Zoltán Fiala, Michael Hinz, Geert-Jan Houben, and Flavius Frasincar. Design and Implementation of Component-based Adaptive Web Presentations. In *19th Symposium on Applied Computing (SAC2004), Nicosia, Cyprus*, pages 1698–1704. ACM Press, March 2004.

[Fiala et al. 2005] Zoltán Fiala, Michael Hinz, and Klaus Meissner. Developing Component-based Adaptive Web Applications with the AMACONTBuilder. In *7th IEEE International Symposium on Web Site Evolution (WSE2005), Budapest, Hungary*, pages 39–45, September 2005.

[Fiala 2001] Zoltán Fiala. Web Content Management Systeme. Master's thesis, Budapest University of Technology and Technische Universität Dresden, June 2001.

[Finin 1989] Timothy W. Finin. GUMS - A general user modeling shell. In *User Modeling and User Adapted Interaction*, pages 411–430. Springer-Verlag, Berlin, 1989.

[Fink et al. 1998] Josef Fink, Alfred Kobsa, and Andreas Nill. Adaptable and adaptive information provision for all users, including disabled and elderly people. *The New Review of Hypermedia and Multimedia*, 16(4):163–188, 1998.

[Francisco-Revilla and Shipman 2000] Luis Francisco-Revilla and Frank M. Shipman. Adaptive medical information delivery combining user, task and situation models. In *2000 International Conference on Intelligent User Interfaces (IUI2000), New Orleans, USA*, pages 94–97, 2000.

[Frasincar et al. 2001] Flavius Frasincar, Geert-Jan Houben, and Richard Vdovjak. An RMM-based methodology for hypermedia presentation design. In *Advances in Databases and Information Systems, 5th East European Conference, ADBIS 2001, Vilnius, Lithuania*, pages 323–337, September 2001.

[Frasincar et al. 2002] Flavius Frasincar, Geert-Jan Houben, and Richard Vdovjak. Specification Framework for Engineering Adaptive Web Applications. In *WWW11, The Eleventh International Conference on the World Wide Web*, 2002.

[Frasincar et al. 2004] Flavius Frasincar, Geert-Jan Houben, Peter Barna, and Zoltán Fiala. Adaptation and Reuse in Web Information Systems. In *ITCC2004, International Conference on Information Technology*, pages 387–291. IEEE Computer Society, April 2004.

[Frasincar et al. 2005] Flavius Frasincar, Geert-Jan Houben, and Peter Barna. Hera Presentation Generator. In *14th International Conference on the World Wide Web (WWW2005), Poster Session*, pages 952–953, May 2005.

[Frasincar 2005] Flavius Frasincar. *Hypermedia Presentation Generation for Semantic Web Information Systems*. PhD thesis, Technische Universiteit Eindhoven, June 2005.

[Fraternali 1999] Piero Fraternali. Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[Fu et al. 2000] Xiaobin Fu, Jay Budzik, and Kristian J. Hammond. Mining Navigation History for Recommendation. In *2000 International Conference on Intelligent User Interfaces, New Orleans, LA*, pages 106–112, 2000.

[Furuta and Stotts 1989] Richard Furuta and P. David Stotts. Separating Hypertext Content from Structure in Trellis. In *UK Hypertext*, pages 205–213, 1989.

[Gaedke et al. 2000] Martin Gaedke, Christian Segor, and Hans-Werner Gellersen. WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. In ACM *Symposium on Applied Computing (SAC2000)*, pages 748–755, March 2000.

[Gaedke et al. 2003] Martin Gaedke, Martin Nussbaumer, Oliver Jung, and Markus Dieckmann. Implementierungstechnologien für Web-Anwendungen. In *Web Engineering: Systematische Entwicklung von Web-Anwendungen*, pages 133–160. 2003.

[Gamperl 2006] Johannes Gamperl. *Ajax - Web 2.0 in der Praxis*. Galileo Computing, 2006. ISBN: 3-89842-764-1.

[Garzotto et al. 1993] Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM - A model-based approach to hypermedia application design. *ACM Transactions on Information Systems*, 11(1):1–26, 1993.

[Gellersen et al. 1997] Hans-Werner Gellersen, Robert Wicke, and Martin Gaedke. WebComposition: an object-oriented support for system for the Web engineering lifecycle. *Computer Networks and ISDN Systems*, 29(8-13):1429–1437, 1997.

[Ghanem and Aref 2004] Thanaa M. Ghanem and Walid G. Aref. Databases Deepen the Web. *IEEE Computer*, 37(1):116–117, 2004.

[Gomes et al. 2001] Pedro Gomes, Sergio Tostao, Daniel Goncalves, and Joaquim Jorge. Web Clipping: Compression Heuristics for Displaying Text on a PDA. In *MobileHCI'01, Lille, France*, 2001.

[Gómez et al. 2001] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual Modeling of Device-Independent Web Applications. In *IEEE Multimedia Special Issue on Web Engineering*, pages 26–39. IEEE Computer Society Press, 2001.

[Goodman 1987] Danny Goodman. *The Complete HyperCard Handbook*. Bantam Books, 1987. ISBN: 0-966-55142-7.

[Graef and Gaedke 2000] Guntram Graef and Martin Gaedke. Construction of Adaptive Web-Applications from Reusable Components. In *First International Conference on Electronic Commerce and Web Technologies (EC-Web 2000)*, pages 1–12, September 2000.

[Greenberg and Roseman 1996] Saul Greenberg and Mark Roseman. GroupWeb: a WWW browser as real time groupware. In *ACM SIGCHI'96 Conference on Human Factors in Computing System, Vancouver, Canada*, pages 271–272. ACM Press, 1996.

[Grønbæk and Trigg 1996] Kaj Grønbæk and Randall H. Trigg. Towards a Dexter-based Model for Open Hypermedia: Unifying embedded references and link objects, 1996.

[Grønbæk et al. 1999] Kaj Grønbæk, Lennert Sloth, and Peter Ørbæk. Webvise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the WWW. *Computer Networks*, 31(11-16):1331–1345, 1999.

[Grünbacher 2003] Paul Grünbacher. Requirements Engineering für Web-Anwendungen. In *Web Engineering: Systematische Entwicklung von Web-Anwendungen*, pages 29–48. 2003.

[Gupta et al. 2003] Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. DOM-based Content Extraction of HTML Documents. In *Twelfth International Conference on the World Wide Web (WWW2003), Budapest, Hungary*, pages 207–214. ACM Press, May 2003.

[Halasz and Schwartz 1994] Frank G. Halasz and Mayer D. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.

[Halasz 1987] Frank G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. In *ACM Hypertext '87 Conference*, pages 345–365, 1987.

[Halpin 2001] Terry A. Halpin. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers, 2001. ISBN: 1-55860-672-6.

[Han et al. 2000] Richard Han, Veronique Perret, and Mahmoud Naghshineh. WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. In *ACM 2000 Conference on Computer Supported Cooperative Work (CSCW'00), Philadelphia, PA.*, pages 221–230, 2000.

[Hardman et al. 1994] Lynda Hardman, Dick C.A. Bulterman, and Guido van Rossum. The Amsterdam hypermedia model: adding time and context to the Dexter model. *Communications of the ACM*, 37(2):50–62, 1994.

[Hendrickx et al. 2005] Filip Hendrickx, Tom Beckers, Nico Oorts, and Rik Van De Walle. An Integrated Approach for Device Independent Publication of Complex Multimedia Documents. In *IMSA2005*, 2005.

[Henze and Nejdl 2001] Nicola Henze and Wolfgang Nejdl. Adaptation in open corpus hypermedia. *International Journal of Artificial Intelligence in Education*, 12(4):325–351, 2001.

[Hepper 2004] Stefan Hepper. The Java Portlet Specification (Die Java Portlet Spezifikation). *it - Information Technology*, 46(5):233–244, 2004.

[@HERA] The Hera software project. http://wwwis.win.tue.nl/~hera/. Date of access: 30th July 2006.

[Hicks and Tochtermann 2001] David L. Hicks and Klaus Tochtermann. Personal Digital Libraries and Knowledge Management. *Journal of Universal Computer Science (JUCS)*, 7(7):550–565, 2001.

[Hinz and Fiala 2004] Michael Hinz and Zoltán Fiala. AMACONT: A System Architecture for Adaptive Multimedia Web Applications. In *Workshop XML Technologien für das Semantic Web (XSW 2004), Berliner XML Tage*, October 2004.

[Hinz and Fiala 2005] Michael Hinz and Zoltán Fiala. Context Modeling for Device- and Location-Aware Mobile Web Applications. In *3rd International Conference on Pervasive Computing (Pervasive 2005), Workshop: PERMID 2005, München*, pages 204–215, May 8-13 2005.

[Hinz et al. 2004] Michael Hinz, Zoltán Fiala, and Frank Wehner. Personalization-Based Optimization of Web Interfaces for Mobile Devices. In *6th International Symposium on Mobile Human-Computer Interaction - Mobile HCI 2004, Glasgow, UK*, pages 204–215. Springer LNCS 3160, September 13-16 2004.

[Hinz et al. 2006] Michael Hinz, Stefan Pietschmann, and Zoltán Fiala. A Framework for Context Modeling in Adaptive Web Applications. In *IADIS International Conference WWW/Internet 2006, Murcia Spain*, October 2006.

[Hoffmann and Dachselt 2003] Heiko Hoffmann and Raimund Dachselt. An Independent Declarative 3D Audio Format on the Basis of XML. In *2003 International Conference on Auditory Display, Boston, MA, USA*, July 2003.

[Hohl et al. 1996] Hubertus Hohl, Heinz-Dieter Böcker, and Rul Gunzenhäuser. Hypadapter: An Adaptive Hypertext System for Exploratory Learning and Programming. *User Modeling and User-Adapted Interaction*, 6(2-3):131–156, 1996.

[Hoja 2005] Mark Hoja. Integration von Interaktionselementen in komponentenbasierte adaptive Web-Anwendungen. Prediploma thesis, Technische Universität Dresden, April 2005.

[Hölldobler 2001] Tanja Hölldobler. *Temporäre Benutzermodellierung für multimediale Produkt-präsentationen im World Wide Web.* Peter Lang, 2001. ISBN: 3-631-38343-6.

[Hori et al. 2000] Masahiro Hori, Goh Kondoh, Kouichi Ono, Shin-Ichi Hirose, and Sandeep Singhal. Annotation-Based Web Content Transcoding. In *9th International World Wide Web Conference, Amsterdam, The Netherlands*, 2000.

[Hori et al. 2002] Masahiro Hori, Kouichi Ono, Teruo Koyanagi, and Mari Abe. Annotation by Transformation for the Automatic Generation of Content Customization Metadata. In *International Conference on Pervasive Computing, Pervasive 2002, Zurich, Switzerland*, pages 267–281, 2002.

[Hothi and Hall 1998] Jatinder Hothi and Wendy Hall. An Evaluation of Adapted Hypermedia Techniques Using Static User Modelling. In *Second Adaptive Hypertext and Hypermedia Workshop at the Ninth ACM International Hypertext Conference (Hypertext'98), Pittsburgh, PA*, pages 45–50, 1998.

[Houben et al. 2005] Geert-Jan Houben, Zoltán Fiala, Kees ven der Sluijs, and Michael Hinz. Building Self-managing Web Information Systems from Reusable Components. In *First International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA'05)*, pages 53–67. FEUP, June 2005.

[Houben 2004] Geert-Jan Houben. Challenges in Adaptive Web Information Systems: Don't Forget the Link! In *ICWE Workshops*, pages 3–11, 2004.

[Huang and Sundaresan 2000] Anita W. Huang and Neel Sundaresan. A Semantic Transcoding System to Adapt Web Services for Users with Disabilities. In *4th International ACM/SIGCAPH Conference on Assistive Technologies (ASSETS'00), Arlington, Virginia*, pages 156–163, 2000.

[Hwang et al. 2002] Yonghyun Hwang, Eunkyong Seo, and Jihong Kim. WebAlchemist: A Structure-Aware Web Transcoding System for Mobile Devices. In *Mobile Search Workshop, Honolulu, Hawaii*, 2002.

[Hwang et al. 2003] Yonghyun Hwang, Jihong Kim, and Eunkyong Seo. Structure-Aware Web Transcoding for Mobile Devices. *IEEE Internet Computing*, 7(5):14–21, 2003.

[@ICWE2004Demo] Adaptive Painting Gallery prototype. http://www-mmt.inf.tu-dresden.de/fiala. Date of access: 30th July 2006.

[@ImageMagick] ImageMagick project homepage. http://www.imagemagick.org/. Date of access: 30th July 2006.

[@imarkup] iMarkup product homepage. http://www.imarkup.com/. Date of access: 30th July 2006.

[Isakowitz et al. 1995] Tomás Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, 1995.

[Isakowitz et al. 1998] Tomás Isakowitz, Michael Bieber, and Fabio Vitali. Web Information Systems - Introduction. *Communications of the ACM*, 41(7):78–80, 1998.

[ISO 2002] ISO/IEC FDIS 21000-2. *MPEG-21 - Part 2: Digital Item Declaration*, `http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm`, 2002.

[Jacyntho et al. 2002] Mark D. Jacyntho, Daniel Schwabe, and Gustavo Rossi. A Software Architecture for Structuring Complex Web Applications. *Journal of Web Engineering, Rinton Press*, 1(1):037–060, 2002.

[@jdom] JDOM Java Document Model project homepage. http://www.jdom.org/. Date of access: 30th July 2006.

[Jin et al. 2001] Yuhui Jin, Stefan Decker, and Gio Wiederhold. OntoWebber: Model-Driven Ontology-Based Web Site Management. In *The first Semantic Web Working Symposium (SWWS'01), Stanford University, California, USA*, pages 529–547, 2001.

[Jörding 1999] Tanja Jörding. Temporary User Modeling for Adaptive Product Presentations in the Web. In *Seventh International Conference on User Modeling (UM99), Banff, Canada*, June 1999.

[Kappel et al. 2004] Gerti Kappel, Birgit Pröll, Siegfried Reich, and Werner Retschitzegger, editors. *Web Engineering - Systematische Entwicklung von Web-Anwendungen.* dpunkt.verlag GmbH, 2004. ISBN: 3-898-64234-8.

[Kappel et al. 2006] Gerti Kappel, Birgit Pröll, Siegfried Reich, and Werner Retschitzegger, editors. *Web Engineering. The Discipline of Systematic Development of Web Applications.* John Wiley and Sons Ltd., 2006. ISBN: 0-470-01554-3.

[Kay 2004] Michael Kay. *XSL Transformations (XSLT) Version 2.0.* W3C Working Draft, `http://www.w3.org/TR/xslt20/`, November 2004.

[Kiczales et al. 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *11th European Conference on Object Oriented Programming (ECOOP'97), Jyväskylä, Finland*, pages 220–242, 1997.

[Klapsing and Neumann 2000] Reinhold Klapsing and Gustaf Neumann. Applying the Resource Description Framework to Web Engineering. In *First International Conference on Electronic Commerce and Web Technologies (ECWeb 2000)*, pages 229–238, 2000.

[Klyne et al. 2003] 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.* W3C Working Draft, `http://www.w3.org/TR/CCPP-struct-vocab/`, 2003.

[Kobsa et al. 2001] Alfred Kobsa, Jürgen Koenemann, and Wolfgang Pohl. Personalised hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16(2):111–155, 2001.

[Koch and Wirsing 2002] Nora Koch and Martin Wirsing. The Munich Reference Model for Adaptive Hypermedia Applications. In *Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2002)*, pages 213–222, 2002.

[Koch et al. 2001] Nora Koch, Andreas Kraus, and Ralf Hennicker. The Authoring Process of the UML-based Web Engineering Approach. In *First International Workshop on Web-Oriented Software Technology (IWWOST2001)*, 2001.

[Koch 2001] Nora Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process.* PhD thesis, Ludwig-Maximilians-Universität München, 2001.

[Koyanagi et al. 2000] Teruo Koyanagi, Kouichi Ono, and Masahiro Hori. Demonstrational interface for XSLT stylesheet generation. *Markup Languages: Theory and Practice*, 2(2):133–152, 2000.

[@kpss05] Komplexpraktikum Multimediatechnik I - Adaptive Webseiten zur Präsentation multimedialer studentischer Arbeiten des Studiengangs Medieninformatik. http://www-mmt.inf.tu-dresden.de/kp_ss05/. Date of access: 30th July 2006.

[Küpper 2005] Axel Küpper. *Location-based Services: Fundamentals and Operation.* John Wiley & Sons Ltd., 2005. ISBN: 0-470-09231-9.

[Lam 2001] Wing Lam. Testing E-Commerce Systems: A Practical Guide. *IT Professional*, 3(2):19–27, 2001.

[Lei et al. 2005] Yuangui Lei, Enrico Motta, and John Domingue. OntoWeaver: an Ontology-based Approach to the Design of Data-intensive Web Sites. *Journal of Web Engineering*, 4(3):244–262, 2005.

[Lie 2005] Håkon Wium Lie. *Cascading Style Sheets.* PhD thesis, University of Oslo, Faculty of Mathematics and Natural Sciences, 2005.

[Lin 2003] J.K. Lin. An Insider's Guide to Today's Cobrowsing. In *Whitepaper of PageShare Technologies, Inc.*, 2003.

[Linden et al. 2003] Greg Linden, Brent Smith, and Jeremy York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[Lowe 2003] David Lowe. Web system requirements: an overview. *Requirements Engineering*, 8(2):102–113, 2003.

[Lyman et al. 2003] Peter Lyman, Hal R. Varian, Kirsten Swearingen, Peter Charles, Nathan Good, Laheem Lamar Jordan, and Joyojeet Pal. *How much information? 2003 Executive Summary.* School of Information Management and Systems at the University of California at Berkeley, `http://www.sims.berkeley.edu/research/projects/how-much-info-2003/`, 2003.

[Maedche et al. 2003] Alexander Maedche, Steffen Staab, Nenad Stojanovic, Rudi Studer, and York Sure. SEmantic portAL: The SEAL Approach. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar]*, pages 317–359, 2003.

[Manjunath et al. 2002] B.S. Manjunath, Philippe Salembier, and Thomas Sikora. *Introduction to MPEG-7.* John Wiley & Sons, 2002. ISBN: 0-471-48678-7.

[Marinilli et al. 1999] Mauro Marinilli, Alessandro Micarelli, and Filippo Sciarrone. A Case-based Approach to Adaptive Information Filtering for the WWW. In *Second Workshop on Adaptive Systems and User Modeling on the World Wide Web, Toronto and Banff, Canada*, pages 81–87, 1999.

[Meißner et al. 2001] Klaus Meißner, Simone Röttger, and Frank Wehner. Dynamische Visualisierung modularer, XML-basierter Kursdokumente. In *11. Arbeitstreffen der GI-Fachgruppe 1.1.5/7.0.1 'Intelligente Lehr-/Lernsysteme', Dortmund, Germany*, 2001.

[Milosavljevic 1997] Maria Milosavljevic. Augmenting the User's Knowledge via Comparison. In *Sixth International Conference on User Modeling (UM97), Vienna, New York*, pages 119–130. Springer Wien New York, 1997.

[Müller et al. 2005] Christof Müller, Katharina Kiegler, Simon Biemer, Moritz Weeger, Martin Schmal, Henri Holjewilken, Carsten Judick, Kerstin Werner, Matthias Niederhausen, and Clark Helwig. *Werkzeuge zur Erstellung adaptiver komponentenhafter Web-Anwendungen.* Dokumentation des Komplexpraktikums I. KP WS0304, TU Dresden, Lehrstuhl Multimediatechnik, 2005.

[Murugesan and Deshpande 2001] San Murugesan and Yogesh Deshpande, editors. *Web Engineering - Managing the Diversity and Complexity of Web Application Development.* Springer Verlag, LNCS Vol. 2016, 2001. ISBN: 3-540-42130-0.

[Murugesan et al. 2001] San Murugesan, Yogesh Deshpande, Steve Hansen, and Athula Ginige. Web Engineering: A New Discipline for Development of Web-Based Systems. In *Web Engineering, Software Engineering and Web Application Development*, pages 3–13. Springer Verlag, LNCS Vol. 2016, 2001.

[Nelson 1965] Theodor Holm Nelson. A File Structure for The Complex, The Changing and the Indeterminate. *20th National Conference of the Association for Computing Machinery*, pages 84–100, 1965.

[Nelson 1987] Theodor H. Nelson. All for One and One for All. In *ACM Hypertext'87 Conference, Chapel Hill, North Carolina, USA*, 1987.

[Niederhausen 2005a] Matthias Niederhausen. *Erstellung eines Adaptation Editors.* AMACONTBuilder Documentation, TU Dresden, Lehrstuhl Multimediatechnik, 2005.

[Niederhausen 2005b] Matthias Niederhausen. Realisierung komplexer Arbeitsabläufe in einem Autorenwerkzeug für komponentenbasierte adaptive Web-Anwendungen. Prediploma thesis, Technische Universität Dresden, November 2005.

[Niederhausen 2006] Matthias Niederhausen. Konzeption und Realisierung eines Hyperlink-Editors für den AMACONTBuilder. Master's thesis, Technische Universität Dresden, May 2006.

[Nielsen 1995] Jakob Nielsen. *Multimedia and Hypertext: The Internet and Beyond.* Morgan Kaufmann, 1995. ISBN: 0-125-18408-5.

[Nielsen 2002] Jakob Nielsen. *Kids' corner: Website usability for children.* Jakob Nielsen's Alertbox at Useit.com., `http://www.useit.com/alertbox/20020414.html`, April 14, 2002.

[Oberlander et al. 1998] Jon Oberlander, Mick O'Donnell, Chris Mellish, and Alistair Knott. Conversation in the museum: experiments in dynamic hypermedia with the intelligent labelling explorer. *The New Review of Hypermedia and Multimedia*, 4:11–32, 1998.

[Ono et al. 2002] Kouichi Ono, Teruo Koyanagi, Mari Abe, and Masahiro Hori. XSLT stylesheet generation by example with WYSIWYG editing. In *2002 Symposium on Applications and the Internet (SAINT 2002), Nara City, Japan*, pages 150 – 159, 2002.

[Oorts et al. 2005] Nico Oorts, Filip Hendrickx, Tom Beckers, and Rik Van De Walle. Multichannel publication of interactive media content for Web Information Systems. In *Fifth International Conference on Web Engineering (ICWE2005), Sydney*, 2005.

[O'Reilly 2006] Tim O'Reilly. *What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software*, `http://www.oreillynet.com/`, 2006.

[Oreizy et al. 1999] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[OSG 2005] OSGi alliance. *About the OSGi Service Platform, Technical Whitepaper Revision 4.1*, `http://www.osgi.org/`, 11 November 2005.

[Osterdiekhoff 2004] Brigitte Osterdiekhoff. Transcoding von Webinhalten. *Informatik Spektrum*, 27(5):448–452, 2004.

[Ovsiannikov et al. 2000] Ilia A. Ovsiannikov, Michael A. Arbib, and Thomas H. McNeill. Annotation Technology. *International Journal of Human-Computer Studies*, 50(4):329–362, 2000.

[Pastor et al. 2003] Oscar Pastor, Joan Fons, and Vicente Pelechano. OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. In *International Workshop on Web Oriented Software Technology (IWWOST)*, pages 65–70, 2003.

[Patel-Schneider et al. 2004] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation, `http://www.w3.org/TR/owl-semantics/`, 10 February 2004.

[Paterno and Mancini 1999] Fabio Paterno and Cristiano Mancini. Designing Web Interfaces Adaptable to Different Types of Use. In *Workshop Museums and the Web, New Orleans, LA, USA*, 1999.

[Paterno et al. 1997] Fabio Paterno, Cristiano Mancini, and Silvia Meniconi. ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models. In *INTERACT 97*, pages 362–366. Chapman & Hall, 1997.

[Petrelli et al. 1999] Daniella Petrelli, Elena Not, Marcello Sarini, Oliviero Stock, Carlo Strapparava, and Massimo Zancanaro. HyperAudio: Location Awareness + Adaptivity. In *CHI'99, Conference on Human Factors in Computing Systems*, pages 21–22, 1999.

[Pospischil et al. 2002] Günther Pospischil, Martina Umlauft, and Elke Michlmayr. Designing LoL@, a Mobile Tourist Guide for UMTS. In *Mobile HCI*, pages 140–154, 2002.

[Preciado et al. 2005] Juan Carlos Preciado, Marino Linaje Trigueros, F. Sanchez, and Sara Comai. Necessity of Methodologies to model Rich Internet Applications. In *7th IEEE International Symposium on Web Site Evolution (WSE2005), Budapest, Hungary*, pages 7–13, September 2005.

[Puerta and Eisenstein 2002] Angel Puerta and Jacob Eisenstein. XIML: a common representation for interaction data. In *2002 Conference on Intelliegent User Interfaces (IUI2002)*, pages 216–217, 2002.

[Röscheisen et al. 1995] Martin Röscheisen, Christian Morgensen, and Terry Winograd. Interaction Design for Shared World-Wide Web Annotations. In *CHI 1995*, pages 328–329, May 1995.

[Rossi et al. 2001] Gustavo Rossi, Daniel Schwabe, and R.M. Guimaraes. Designing Personalized Web Applications. In *WWW10, The Tenth International Conference on the World Wide Web, Hong Kong*, pages 275–284, 2001.

[Sauer and Engels 1999] S. Sauer and G. Engels. Extending UML for Modeling of Multimedia Applications. In *IEEE Symposium on Visual Languages, Tokyo, Japan*, page 88. ACM Press, 1999.

[Schaefer et al. 2002] Robbie Schaefer, Andreas Dangberg, and Wolfgang Mueller. RDL/TT-A Description Language for Profile-Dependent Transcoding of XML Documents. In *ITEA Workshop on Virtual Home Environments (VHE), Paderborn, Germany*, February 2002.

[Schilit et al. 1994] Bill Schilit, Norman Adams, , and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California*, pages 85–90. IEEE Computer Society Press, 1994.

[Schmidt et al. 1999] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced Interaction in Context. In *First International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe, Germany*, pages 89–101, 1999.

[Schwabe and de Moura 2003] Daniel Schwabe and Sabrina Silva de Moura. Interface Development for Hypermedia Applications in the Semantic Web. In *WebMedia and LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, pages 106–113. IEEE Computer Society, 2003.

[Schwabe et al. 1996] Daniel Schwabe, Gustavo Rossi, and Simone D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Hypertext '96, The Seventh ACM Conference on Hypertext, Washington DC*, pages 116–128. ACM, March 1996.

[Schwabe et al. 1999] Daniel Schwabe, Rita de Almeida Pontes, and Isabela Moura. OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW. *SigWEB Newsletter*, 8(2), 1999.

[Schwinger and Koch 2003] Wieland Schwinger and Nora Koch. Modellierung von Web-Anwendungen. In *Web Engineering: Systematische Entwicklung von Web-Anwendungen*, pages 49–76. 2003.

[Segor and Gaedke 2000] Christian Segor and Martin Gaedke. Crossing the Gap - From Design to Implementation in Web-Application Development. In *Proceedings of Information Resources Management Association International Conference, Anchorage, Alaska, USA*, 2000.

[Shen 1996] Wei-Min Shen. An Efficient Algorithm for Incremental Learning of Decision Lists. Technical Report USC-ISI-96-012, Information Sciences Institute, University of Southern California, 1996.

[Shneiderman and Kearsley 1989] Ben Shneiderman and Greg Kearsley. *Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information.* Addison Wesley, 1989. ISBN: 0-201-15171-5.

[Shneiderman 1987] Ben Shneiderman. User Interface Design for the Hyperties Electronic Encyclopedia. In *Hypertext 1987: Chapel Hill, North Carolina, USA*, pages 189–194. ACM, 1987.

[Smith et al. 1998] John R. Smith, Rakesh Mohan, and Chung-Sheng Li. Transcoding Internet Content for Heterogeneous Client Devices. In *IEEE International Symposium on Circuits and Systems (ISCAS '98)*, volume 3, pages 599–602, 1998.

[Smyth et al. 2002] Barry Smyth, Keith Bradley, and Rachael Rafter. Personalization techniques for online recruitment services. *Communications of the ACM*, 45(5):39–40, 2002.

[Spivey 1989] J. Michael Spivey. *The Z Notation.* Prentice-Hall International, Hertfordshire, England, 1989. ISBN: 0-139-78529-9.

[Starke 2005] Susett Starke. Konzeption eines adaptiven Web-Informationssystems zur Präsentation multimedialer Studienergebnisse. Prediploma thesis, Technische Universität Dresden, July 2005.

[Steindl et al. 2003] Christoph Steindl, Rudolf Ramler, and Josef Altmann. Testen von Web-Anwendungen. In *Web Engineering: Systematische Entwicklung von Web-Anwendungen*, pages 161–186. 2003.

[Swoboda and Wadge 2000] Paul Swoboda and William W. Wadge. Vmake, ISE and IRCS: General tools for the intensionalization of software systems. In *Intensional Programming II, World-Scientific*, 2000.

[Szyperski 1998] Clemens Szyperski. *Component Software, Beyond Object-Oriented Programming.* Addison-Wesley, 1998. ISBN: 0-201-17888-5.

[Thalheim and Düsterhöft 2001] Bernhard Thalheim and Antje Düsterhöft. SiteLang: Conceptual Modeling of Internet Sites. In *20th International Conference on Conceptual Modeling (ER 2001), Yokohama, Japan*, pages 179–192, 2001.

[Tietz 2006] Vincent Tietz. Entwicklung template-basierter adaptiver Web-Anwendungen mit dem AMACONTBuilder. Prediploma thesis, Technische Universität Dresden, June 2006.

[Tochtermann and Dittrich 1996] Klaus Tochtermann and Gisbert Dittrich. The Dortmund Family of Hypermedia Models - Concepts and their Application. *Journal of Universal Computer Science*, 2(1):34–55, 1996.

[Tsalgatidou and Veijalainen 2000] Aphrodite Tsalgatidou and Jari Veijalainen. Mobile Electronic Commerce: Emerging Issues. In *First International Conference on Electronic Commerce and Web Technologies (EC-Web 2000), London, UK*, pages 477–486, 2000.

[Ulbricht 2006] Dirk Ulbricht. Web-basierte Kollaboration mobiler Endgeräte. Prediploma thesis, Technische Universität Dresden, April 2006.

[Ungar and Smith 1987] David Ungar and Randall B. Smith. Self: The Power of Simplicity. In *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87), Orlando, Florida*, pages 227–242, 1987.

[@VCSProject] Virtual Collaboration Services (VCS) project homepage. http://www-mmt.inf.tu-dresden.de/Projekte/. Date of access: 30th July 2006.

[Vdovjak et al. 2003] Richard Vdovjak, Flavius Frasincar, Geert-Jan Houben, and Peter Barna. Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering, Rinton Press*, 2(1&2):003–026, 2003.

[Wadge and Schraefel 2001] William W. Wadge and Monica M.C. Schraefel. A Complementary Approach for Adaptive and Adaptable Hypermedia: Intensional Hypertext. In *Hypermedia: Openness, Structural Awareness and Adaptivity - International Workshop OHS-7, Aarhus, Denmark*, pages 327–334, 2001.

[Wadge et al. 1998] William W. Wadge, Gord Brown, Monica M.C. Schraefel, and Taner Yildrim. Intensional HTML. In *4th International Workshop on Principles of Digital Document Processing (PODDP'98), Saint Malo, France*, pages 128–139, 1998.

[Wadge 2000] William W. Wadge. Intensional Markup Language. In *Third International Workshop on Distributed Communities on the Web (DCW2000), Quebec City, Canada*, pages 82–89, 2000.

[@WebRatio] WebRatio product homepage. http://www.webratio.com/. Date of access: 30th July 2006.

[Wehner and Lorz 2001] Frank Wehner and Alexander Lorz. Developing Modular and Adaptable Courseware Using TeachML. In *ED-MEDIA, World Conference on Educational Multimedia, Hypermedia and Telecommunications, Tampere, Finland*, 2001.

[Westbomke and Dittrich 2002] Jörg Westbomke and Gisbert Dittrich. Towards an XML-based Implementation of Structured Hypermedia Documents. *Journal of Universal Computer Science*, 8(10):944–956, 2002.

[Westbomke 2001] Jörg Westbomke. XML-*basierte Implementierung struturierter Hypertextdokumente*. Dissertation. Shaker Verlag, Aachen, 2001. ISBN: 3-826-59986-1.

[Wir 2001] Wireless Application Group, WAP Forum. *User Agent Profile Specification*, 2001.

[Wu 2001] Hongjing Wu. *Reference Architecture for Adaptive Hypermedia Applications*. PhD thesis, Technische Universiteit Eindhoven, 2001.

[@XMLSpy] XMLSpy product homepage. http://www.altova.com/. Date of access: 30th July 2006.

[Yankelovich et al. 1988] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, 21(1):81–96, 1988.

[Yee 1998] Ka-Ping Yee. *CritLink: Better Hyperlinks for the WWW*, `http://crit.org/~ping/ht98.html`, 1998.

[Yesilada et al. 2004] Yeliz Yesilada, Simon Harper, Carole Goble, and Robert Stevens. Screen Readers Cannot See - Ontology Based Semantic Annotation for Visually Impaired Web Travellers. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, pages 445–458. Springer LNCS 3140, 2004.

[Ziegeler and Langham 2002] Carsten Ziegeler and Matthew Langham. *Cocoon: Building XML Applications*. Sams Publishing, 2002. ISBN: 0-735-71235-2.

[Ziegert et al. 2004] Thomas Ziegert, Markus Lauff, and Lutz Heuser. Device Independent Web Applications – The Author Once - Display Everywhere Approach. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, pages 244–255. Springer LNCS 3140, July 2004.

List of Publications

[Fiala et al. 2003a] Zoltán Fiala, Michael Hinz, Klaus Meissner, and Frank Wehner. A Component-based Approach for Adaptive Dynamic Web Documents. In *Twelfth International Conference on the World Wide Web (WWW2003), Poster session*. ACM, May 2003.

[Fiala et al. 2003b] Zoltán Fiala, Michael Hinz, Klaus Meißner, and Frank Wehner. A Component-based Approach for Adaptive Dynamic Web Documents. *Journal of Web Engineering, Rinton Press*, 2(1&2):058–073, September 2003.

[Fiala and Meissner 2003] Zoltán Fiala and Klaus Meissner. Annotating Virtual Web Documents with DynamicMarks. In *Workshop XML Technologien für das Semantic Web (XSW 2003), Berliner XML Tage*, pages 67–77, October 2003.

[Fiala et al. 2003] Zoltán Fiala, Michael Hinz, and Frank Wehner. An XML-based Component Architecture for Personalized Adaptive Web Applications. In *Workshop Personalisierung mittels XML-Technologien, Berliner XML Tage*, pages 370–378, October 2003.

[Fiala et al. 2004] Zoltán Fiala, Michael Hinz, Geert-Jan Houben, and Flavius Frasincar. Design and Implementation of Component-based Adaptive Web Presentations. In *19th Symposium on Applied Computing (SAC2004), Nicosia, Cyprus*, pages 1698–1704. ACM Press, March 2004.

[Frasincar et al. 2004] Flavius Frasincar, Geert-Jan Houben, Peter Barna, and Zoltán Fiala. Adaptation and Reuse in Web Information Systems. In *ITCC2004, International Conference on Information Technology*, pages 387–291. IEEE Computer Society, April 2004.

[Fiala et al. 2004] Zoltán Fiala, Flavius Frasincar, Michael Hinz, Geert-Jan Houben, Peter Barna, and Klaus Meissner. Engineering the Presentation Layer of Adaptable Web Information Systems. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, pages 459–472. Springer LNCS 3140, July 2004.

[Hinz et al. 2004] Michael Hinz, Zoltán Fiala, and Frank Wehner. Personalization-Based Optimization of Web Interfaces for Mobile Devices. In *6th International Symposium on Mobile Human-Computer Interaction - Mobile HCI 2004, Glasgow, UK*, pages 204–215. Springer LNCS 3160, September 13-16 2004.

[Hinz and Fiala 2004] Michael Hinz and Zoltán Fiala. AMACONT: A System Architecture for Adaptive Multimedia Web Applications. In *Workshop XML Technologien für das Semantic Web (XSW 2004), Berliner XML Tage*, October 2004.

[Hinz and Fiala 2005] Michael Hinz and Zoltán Fiala. Context Modeling for Device- and Location-Aware Mobile Web Applications. In *3rd International Conference on Pervasive*

Computing (Pervasive 2005), Workshop: PERMID 2005, München, pages 204–215, May 8-13 2005.

[Fiala and Houben 2005] Zoltán Fiala and Geert-Jan Houben. A Generic Transcoding Tool for Making Web Applications Adaptive. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, pages 15–20. FEUP, June 2005.

[Houben et al. 2005] Geert-Jan Houben, Zoltán Fiala, Kees ven der Sluijs, and Michael Hinz. Building Self-managing Web Information Systems from Reusable Components. In *First International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA'05)*, pages 53–67. FEUP, June 2005.

[Hinz and Fiala 2005] Michael Hinz and Zoltán Fiala. Distribution and Synchronization of Context Modeling Mechanisms between Servers and Clients on the Web. In *Eigth International Conference on Wireless Personal Multimedia Communication (WPMC'05), Aalborg, Denmark*, September 2005.

[Fiala et al. 2005] Zoltán Fiala, Michael Hinz, and Klaus Meissner. Developing Component-based Adaptive Web Applications with the AMACONTBuilder. In *7th IEEE International Symposium on Web Site Evolution (WSE2005), Budapest, Hungary*, pages 39–45, September 2005.

[Casteleyn et al. 2006a] Sven Casteleyn, Zoltán Fiala, Geert-Jan Houben, and Kees van der Sluijs. Considering Additional Adaptation Concerns in the Design of Web Applications. In *Adaptive Hypermedia and Adaptive Web-Based Systems 2006 (AH2006)*. Springer, June 2006.

[Casteleyn et al. 2006b] Sven Casteleyn, Zoltán Fiala, Geert-Jan Houben, and Kees van der Sluijs. From Adaptation Engineering Towards Aspect-Oriented Context-Dependency. In *Fifteenth International Conference on the World Wide Web (WWW2006), Poster session*. ACM, May 2006.

[Hinz et al. 2006] Michael Hinz, Stefan Pietschmann, and Zoltán Fiala. A Framework for Context Modeling in Adaptive Web Applications. In *IADIS International Conference WWW/Internet 2006, Murcia Spain*, October 2006.

List of Abbreviations

ADV Abstract Data View

AHAM Adaptive Hypermedia Application Model

AHS Adaptive Hypermedia System

API Application Programming Interface

AWIS Adaptive Web Information System

cHTML Compact HTML

DOM Document Object Model

DTD Document Type Definition

GAC Generic Adaptation Component

HTML HyperText Markup Language

JSP Java Server Pages

MPEG Motion Pictures Expert Group

OOHDM Object-Oriented Hypermedia Design Model

OWL Web Ontology Language

RDFS Resource Description Framework Schema

RDF Resource Description Framework

RMM Relationship Management Methodology

SeRQL Sesame RDF Query Language

SMIL Synchronized Multimedia Integration Language

SQL Structured Query Language

UML Unified Modeling Language

W3C World Wide Web Consortium

WCML WebComposition Markup Language

WebML Web Modeling Language

WIS Web Information System

WML Wireless Markup Language

WSDM Web Site Design Method

WWW World Wide Web

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

XPath XML Path Language

XSL(T) Extensible Stylesheet Language (Transformations)

Index

X

Curriculum Vitae

Zoltán Fiala was born on 19th March 1977 in Budapest, Hungary. After completing his pre-university education, he studied computer science at the Budapest University of Technology and Economics. Within the scope of a scholarship from Deutsche Telekom, he spent the last year of his studies at Dresden University of Technology (Technische Universität Dresden - TUD), Germany. He graduated with a diploma thesis on Web Content Management Systems, supervised by Prof. Dr.-Ing. Klaus Meißner.

After graduation, he worked as a PhD student at the Graduiertenkolleg "Werkzeuge zum effektiven Einsatz paralleler und verteilter Rechnersysteme" at the Department of Computer Science of Dresden University of Technology. His research was carried out within the scope of the AMACONT project, with a special accent on the authoring process of component-based, personalized, ubiquitous Web applications. Since September 2004 he has been working as a research assistant at the Multimedia Technology Group of Dresden University of Technology.