Processing Structured Hypermedia — A Matter of Style

Jacco van Ossenbruggen

SIKS Dissertation Series No 2001-5
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

ISBN 90 6196 502 0

VRIJE UNIVERSITEIT

Processing Structured Hypermedia — A Matter of Style

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen \ Wiskunde en Informatica
op dinsdag 10 april 2001 om 15.45 uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Jacco Ronald van Ossenbruggen

geboren te Hoorn

Promotor: prof.dr. J.C. van Vliet
Copromotor: dr. A. Eliëns

Contents

Preface

I came across the terms "hypertext" and "hypermedia" relatively late. It was in 1991, during the discussion of an example for the Object-Oriented Programming course at the Vrije Universiteit (VU) in Amsterdam, given by Anton Eliëns. If I only had known then what an important role hypermedia, object-oriented programming and Anton would play in both my Master's and PhD thesis...

A year later, my girlfriend Annemarie and I planned to do our final term project together. Our common interests made us look for a subject that combined music and computer science. Although I was a musical *tabula rasa*, this could be compensated by my willingness to learn and Annemarie's strong background in music. It was again Anton who suggested that we should try to extend a hypermedia system (developed during another final term project) with music as a new media type. In those days, I wondered whether "hypermusic" was a topic with sufficient scientific significance for a Master's thesis. And naturally, I thought Anton (who turned out to also have a great interest in music) was only joking when he replied: "You could even do a PhD thesis on this subject." I really enjoyed doing that final term project, even though Annemarie decided to quit our project due to lack of time.

These were the early days of the Web and Mosaic, and after arriving at the university, I usually spent the first half hour checking out all the new pages that were added to the Web. Sometimes, I was lucky and there were two, or even three, new HTTP servers set up in a single day! I was one of the students that Anton managed to convince to write a paper about their Master's thesis research, and I was — still being a student — pleasantly surprised when my paper was accepted at the ACM's European Conference on Hypermedia Technology (ECHT'94).

Additionally, the paper had enlarged my chances to become a PhD student at the VU, and that gave me the opportunity to write this thesis. Due to another conference, in Dublin, Ireland, Anton could not be present at my graduation ceremony. It was in Dublin however, that Anton talked with Dick Bulterman and Lynda Hardman of CWI (the National Research Institute for Mathematics and Computer Science in the Netherlands), and where they agreed upon me coming to CWI for a six months period. I hardly could have wished for a better way to start my PhD research: after visiting ECHT'94, I started to work at CWI's group headed by Dick Bulterman, a group firmly rooted in both hypertext and multimedia research. The many discussions I had with Lynda made me aware of the intricacies of time-based hypermedia in general, and of the Amsterdam Hypermedia Model in particular. Lynda also introduced me to the (former) PhD students of MMUIS, and I hope to enjoy many more of their informal meetings after this thesis has been completed. Sjoerd Mullender and Jack Jansen taught me the inner details of their CMIFed hypermedia authoring environment and the programming language in which CMIFed was implemented, Python. This knowledge turned out to be very valuable for my later work at CWI.

I returned to the VU after these six months, but would still frequently visit CWI. At the VU, we had to decide on which particular hypermedia document model we would base our prototype implementations, and how to integrate that model with the model underlying the World Wide Web. Due to the wide variety of models that were available, most of them not "Web-aware", I wanted to postpone the decision for as long as possible. In this situation, I came across several articles from authors with a background in electronic publishing, advocating that different applications will always need different document models, and explaining how to design systems that could be easily adapted to new document models. Since these systems were often based on SGML-technology, I started to learn SGML, and realized soon that an SGML-based hypermedia system would allow both easy experimentation with different hypermedia document models, and full integration with the Web[1]. Bastiaan Schönhage, still a Master's student in those days, managed to control the large amount of (SGML) complexity involved and to transform my initial proof-of-concept into a fully operational SGML-based Web browser for his final term project. Anton, Bastiaan, Martijn van Welie, Sebastiaan Megens (two other Master's students) and I, greatly enjoyed our joint effort to integrate the many musical components we had developed into our SGML-based Web browser.

My knowledge of SGML (and more specifically its time-based hypermedia extension HyTime), and CWI's wish to have a more standardized interchange format as an alternative to CMIF's native multimedia file format, was one of the reasons for developing a mapping from CMIF to HyTime, and to implement an extension to the CMIFed environment to perform this translation automatically. The initial prototypes of my mapping and translation software were later significantly extended and improved by Lloyd Rutledge, HyTime expert *par excellence*. This formed the basis for the further cooperation with Lloyd, and I hope the results of our many discussions and publications are appropriately reflected in this thesis. Thanks, Lloyd!

I would also like to thank Dick Bulterman, Sjoerd Mullender, Jack Jansen and Lynda Hardman. Dick gave me the opportunity to work within his group at CWI and granted me total freedom in choosing my research topics. Nevertheless, he has continuously encouraged me to participate in all the activities of his group that were relevant for my research projects. The ability to participate in the W3C working group that developed SMIL is only one example of the many opportunities Dick has given me. With their long experience in hypermedia authoring, Sjoerd and Jack provided me with many insights, and their implementation of CMIFed's SMIL support forms an indispensable basis for the work on the Berlage environment reported in the third part of the thesis. After

[1] In those days, I was not aware of the role SGML had already played in the history of our software engineering group at the VU. An earlier project had even resulted in one of the world's first SGML parsers, developed by Jos Warmer, Sylvia van Egmond and Hans van Vliet.

moving from the university to CWI, Lynda gave me the opportunity to finish my thesis during office hours — providing helpful comments along the way. Lynda, many thanks for being so patient.

At the VU, I would like to thank my promotor Hans van Vliet for his support and patience. I really enjoyed the lively company of Martijn van Welie, Bastiaan Schönhage, Anton Eliëns and especially Frank Niessink, who I was lucky to have as my roommate for about three years.

Finally, I would like to thank Annemarie for her continuous support in my personal life and for not allowing this thesis to have any negative impact on our relation.

Jacco van Ossenbruggen, Amsterdam 2001

Part I

The Hypermedia Research Agenda

Chapter 1

Introduction

Mankind has a long history in recording information, which goes back to the days far before we were able to read or write. For thousands of years, the form of the documents in which we recorded this information was necessarily the same as the form in which the document was presented to the reader. Even after the introduction of the computer, which had the potential for a more flexible handling of documents, both forms remained remarkably similar. One of the main goals of the early digital text processing systems was to obtain the same quality prints as were produced by traditional printing techniques. Another goal was to provide "what you see is what you get" interfaces to allow authors to write documents the way they used to do before the introduction of the computer. The potential flexibility of the computer was mainly used to provide better support for the authoring and production process, not for the distribution and presentation process. Due to the superior quality of printed paper as compared to computer displays, the final version of a document was still disseminated to its audience in the traditional way: on paper.

While the difference in quality between printed paper and high-end computer displays is slowly becoming less significant, recent trends have introduced new functionality which make it both worthwhile to present documents on a computer display, and to abstract from the presentation of a document in order to be able to generate different versions of a document from the same source.

First, the methods and techniques to define machine-supported relations among different locations in a text — known as *hypertext* — have rapidly outgrown the research labs in which they were studied in the early sixties. Hypertext systems have become commercially available, and these systems offer the user an associative, point-and-click interface to a set of documents. Hypertexts through which users navigate by following (pre-defined) relationships allow for new, non-linear ways of reading documents on-line. On the other hand, the introduction of hypertext required the development of new document models reflecting the non-linear structure of the text, new authoring paradigms and new ways of proofreading (e.g. to detect dangling links).

Second, the success of the PC and its ever increasing computational power and storage capacity has boosted the use of media other than text and graphical material.

Presentation on a computer display instead of paper allows inclusion of audio, video, animations and even executable code in a document, providing authors with new opportunities, especially in commercial and educational applications. These *multimedia* documents also require new models, since their spatial layout can no longer be derived from a linear text flow, and the document model should support alignment and synchronization of different media within the temporal dimension.

Third, the *World Wide Web* is using the basic principles of hypertext (and, to a lesser extent, multimedia) to access resources available on the Internet. While the Web, being an extremely simple hypertext system, was initially frowned upon by many hypertext researchers, the ability to link to any document on the Internet made it far more useful and successful than the more traditional hypertext systems. Today, "the Web" is for many of its users synonymous to the term hypertext; it is even used to refer to the Internet as a whole.

Combined, these three trends made electronic documents in general, and hypermedia documents in particular, more than just an item on the research agenda of universities and other research centers. The massive adoption of Web-related technology, both on the Internet and the Intranet, makes *document engineering* — the design, development, testing and maintenance of electronic documents — as important as traditional software engineering. Central themes in document engineering include the need to deliver versions of a document on different media and on different platforms, the need to deliver versions with different style properties and sometimes even different content and the need to keep up with the rapidly changing technology while ensuring longevity of the documents concerned. This thesis is about the models and technology developed to satisfy these needs.

Various disciplines have influenced the design of today's hypermedia models and hypermedia systems, and all these disciplines have a different historical background, different objectives and different research agendas. The concept of hypermedia itself, for instance, is often described as a combination of hypertext and multimedia, two disciplines with different historic backgrounds and research agendas. In the nineties, the World Wide Web became a primary subject of research — with its own research community and associated series of international conferences. On the other hand, the markup languages on the Web are all based on SGML, a more than ten year old standard that has it roots in the structured documents and electronic publishing research that started in the early sixties.

All these research communities tend to have different perspectives on electronic documents which are relevant for hypermedia documents. The next section briefly sketches the backgrounds of these disciplines and the research issues involved.

1.1 Alternative perspectives on hypermedia research

Hypermedia research is, in general, about modeling information, and, in particular, about modeling relations among pieces of information. This makes hypermedia research a broad and multi-disciplinary research field. The hypermedia literature describes a broad range of topics, with contributions from authors originating from several research communities. Figure 1.1 depicts four disciplines which have all had a major influence on the topic of this thesis. Below, we give a short characterization of each discipline, and its relationship with the main topics of the thesis.

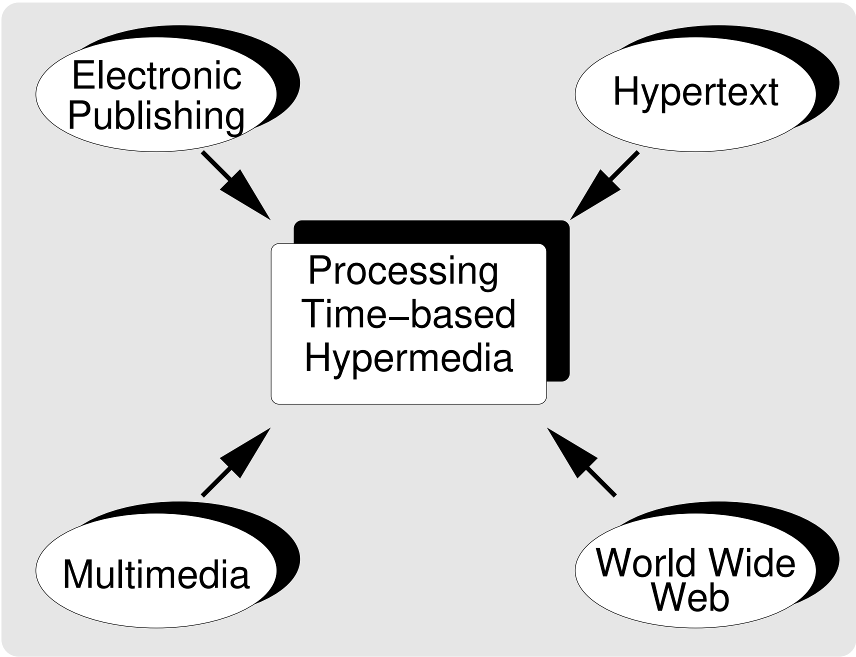

Figure 1.1: Research fields influencing time-based hypermedia processing.

Electronic Publishing

Since the early eighties, several conferences on electronic publishing have been organized, and in 1988 the first issue of Electronic Publishing — Origination, Dissemination and Design (EPodd) appeared. Articles written for this forum approach document processing often from a software engineering perspective, stressing the issues for industrial strength document processing systems. Typical subjects include document reuse, interoperability, internationalization, and standardization ("the SGML and ODA school") and high quality typography and pagination algorithms ("the TEX school"). While a number of articles discuss hypertext and multimedia issues, a large part of the electronic publishing literature is focused on linear and text-based documents.

The concept of a *structured document* plays a central role throughout this thesis, and it originates from this community. Structured documents deploy *structured markup* to make the inherent structure of the document's content explicit, leaving the characteristics of the presentation implicit. The following chapter gives a more extensive treatment of the notion of structured documents.

Hypertext

Hypertext has a remarkably long history, which goes back as far as 1945, the year Vannevar Bush published his landmark article "As We May Think" [53]. Other visionaries who have had a strong influence include Doug Engelbart, who developed the first working hypertext system in the early sixties, and Ted Nelson, who coined the term "hypertext" a few years later. The hypertext community is also heavily influenced by a group of researchers who, after attending a NIST workshop in 1988, developed the model which has been published as the Dexter Hypertext Reference Model. Typical topics in the hypertext literature include variants on the node/link hypertext model (composite nodes, typed links, etc.), user interface and navigation problems ("lost in hyperspace"), interoperability among different hypertext systems, and the role of hypertext in fiction and poetry. The main international conference on hypertext is organized on an annual basis, alternating between the USA and Europe.

Research carried out by the hypertext community has strongly influenced this thesis. The Dexter Hypertext Reference Model plays an important role in Part II, and research on hypertext interoperability will be discussed in Part III. But more generally, hypertext has fundamentally changed the perception of what a document exactly is. It has enriched the concept of a document by blurring the distinction between the active author and passive reader. A hypertext encourages readers to actively select parts from the material contained in the document, and to read these parts in the order they find most appropriate. Hypertext has further blurred the distinction between author and reader by supporting readers to make annotations and share these annotations with other users. Another important aspect of a hypertext document is that it makes its relationships with other documents explicit. Hypertext explicitly places a document in the context of a much larger collection of related documents. It is this enriched notion of documents which we will use throughout the rest of the thesis.

Multimedia

Until the early nineties, multimedia topics have not received much interest in the academic world (the few exceptions include the MIT Media Lab). The success of the (multimedia) PC more or less marked the start of the ACM and IEEE Multimedia conference series, and several new journals explicitly devoted to multimedia systems (including IEEE Multimedia and ACM Multimedia Systems). Nowadays, a huge number of conferences, books and journals discuss multimedia related topics. Typical topics include high bandwidth networks, quality of service management, audio and video compression, scheduling and synchronization, multimedia databases, and multimedia presentation and authoring systems. Special conferences and journals are devoted to related subjects such as computer graphics and visualization, which we consider as research fields in their own right.

The need for synchronization facilities makes time an important dimension in all multimedia systems. The lack of the notion of a temporal dimension in many text-based models is a frequently recurring issue in this thesis. Furthermore, the notion of time has strongly influenced the semantics of the term "hypermedia" as used throughout this thesis. While many researchers within the hypertext community tend to use the terms "hypertext" and "hypermedia" synonymously, we regard support for time-based media as an essential feature distinguishing hypermedia from hypertext systems. In cases where this distinction is particularly important, we use the term "time-based hypermedia" to stress the difference.

World Wide Web

The World Wide Web originates from a project initiated by Tim Berners-Lee at the European Laboratory for Particle Physics (CERN) in 1989. Although the project started originally to allow information sharing between research teams working on various (High Energy Physics) laboratories in the world, after the introduction of NCSA's Mosaic, the first commonly available browser with a graphical user interface, the Web has become the most important Internet application in the world.

Berners-Lee demonstrated a prototype version of his system at the Hypertext conference in 1991, but the Web community and Hypertext community have been remarkably separated research groups (as an example, The 6th World Wide Web conference (in the USA) and the 8th Hypertext conference (in Europe) were scheduled simultaneously). This has started to change only recently. The Web has now become a topic of significant importance on the Hypertext and Electronic Publishing conferences. Additionally, ACM's special interest group on hypertext changed is name from SIGLINK to SIGWEB in order to attract members from both the hypertext and the Web community. A major part of the Web community is cooperating within the World Wide Web Consortium (W3C). Although W3C is an industrial consortium, several research institutes are also a member.

While many research communities have research agendas that are different from the agenda of the Web, now that the Web protocols are maturing, there is a strong trend towards convergence. The Web is rapidly assimilating techniques developed by other research areas. Still, many issues remain to be solved. Much of the research described in this thesis can be characterized as an attempt to integrate electronic publishing, hypertext and multimedia technology into the Web's infrastructure.

Other relevant disciplines

While the four disciplines mentioned above have had the most direct impact on the topic of this thesis, many other research communities have also influenced the hypermedia research arena. Since hypermedia systems provide storage, retrieval and presentation of complex information, there is a close relation between hypermedia research and database research. Hypertext researchers have always used database technology. Several hypertext systems were built by adding associative links and navigation-based interfaces to existing databases. Sometimes, hypermedia models were inspired by database models.

On the other hand, hypertext and multimedia systems have influenced the design of database interfaces, and nowadays several database systems support hyperlinking, or even provide a "query by navigation" interface. The need to provide more sophisticated Web-based access mechanisms to existing databases will probably result in further integration of the database and hypermedia world. Due to the document-oriented nature of our research, databases only play a minor role in this thesis. We occasionally use existing database terminology to illustrate hypermedia concepts, and briefly discuss the use of document-oriented techniques in data-oriented database applications.

Research in areas other than databases, such as in distributed systems, computer networks, computer graphics, visualization and virtual reality, has also played an important role in the design of hypermedia systems. Again, given the document-oriented nature of our research, these research areas are beyond the scope of this thesis.

1.2 Scope and contributions of the thesis

This thesis combines the perspectives and research agendas of the relevant research communities by applying the use of structured documents to the domain of time-based hypermedia. We show that the main goals of the different research communities can indeed be realized by using structured hypermedia documents.

Structured documents (as developed within the electronic publishing community) can support the structuring mechanisms for complex linking (required by the hypertext community) and temporal and spatial alignment (required by the multimedia community). Given the history of the Web, and the more recent developments based on XML, structured documents are also a logical choice for realizing time-based hypermedia on the World Wide Web.

Structured documents were initially introduced as a means of providing long-term storage of documents in a platform and application neutral way, and to be able to generate different printed versions from the same source. Later, this approach was (inconspicuously) introduced on the Web by means of HTML, the Web's hypertext document format, and, more recently, the CSS style sheet format which is used to describe the style of HTML documents. The development of XML will further promote the use of structured documents on the Web.

Despite the popularity of hypermedia and the advantages of structured documents, there has been a relatively limited amount of research addressing the use of structured documents to store and present time-based hypermedia documents. Therefore, the research question of this thesis is how structured documents can be used to improve the design, development, maintenance and presentation of time-based hypermedia documents on the Web. To answer this question, we explore the following issues:

1. the use of structured documents for time-based hypermedia,
2. a formal (reference) model for time-based hypermedia, and
3. software support for processing structured time-based hypermedia documents.

The requirements for structured hypermedia document processing are discussed, in particular the areas where the underlying document models differ from their text-based counterparts. These areas include different spatial layout models, temporal synchronization and scheduling primitives, advanced hyperlinking mechanisms and the need for more run-time control over the document presentation.

Since many of the advantages of structured documents relate to longevity, interoperability and platform-independence, standardization is an important issue. We give an in-depth treatment of the use of SGML, XML and related standards to encode and present hypermedia documents. Since the solutions provided by these standards are merely on a syntactical level, we discuss the methods and software architectures needed to attach the operational semantics to the standardized syntactical constructs. We exemplify the discussion by describing two concrete software architectures, the DejaVu framework and the the Berlage environment.

Contributions of the thesis

The tangible results of the research described include:

- The initial design and implementation of the extensible DejaVu Web browser described in Chapter 8. Innovative aspects of the browser include the use of SGML-based structured documents for time-based hypermedia and the script-based style sheet mechanism to associate SGML markup with the multimedia functionality implemented by the software components of the DejaVu framework.

- The initial design and implementation of the HyTime-based document transformations of the Berlage environment described in Chapter 9. While other research groups had developed HyTime-based systems before (see discussion on page 206), this was the first practical use of the HyTime standard to encode a fully-fledged, time-based hypermedia document model.

- The formalization of the Amsterdam Hypermedia Model (AHM) described in Chapter 5. Discussing this formalization with Lynda Hardman, the main author of the AHM, also helped to refine the AHM as reflected in Hardman's PhD thesis "Modelling and Authoring Hypermedia Documents" [121]. A preliminary version of Chapter 5 was included as an appendix of that thesis.

- The research described in this thesis also provided me with the necessary background to be able to play a significant role in the development of SMIL [231] by participating in the W3C working group on synchronized multimedia (SYMM [111]).

The insight gained during the development of the DejaVu framework, the Berlage environment, the formalization of the AHM and the work on SMIL provided the basis for the major conceptual contribution of the thesis: the systematic overview of the research on structured text, hypertext, multimedia, hypermedia and Web-based document processing given in Chapters 2 and 3. While these chapters provide an introduction to the remainder of the thesis, their main goal is to identify the fundamental incompatibilities of the different approaches and the remaining open issues that need to be solved before these approaches can be reconciled.

Outline of the thesis

Part I of the thesis provides an overview of the research on text, hypertext, multimedia, hypermedia and Web document processing. In addition, it discusses the relationships between these traditionally separate research fields. Part II discusses hypermedia reference models from a more formal perspective. It also explores the research issues which still need to be resolved for a fully satisfactory hypermedia reference model. Finally, Part III describes the requirements of a software architecture supporting the processing of structured hypermedia documents, using two prototype architectures as an example.

The following chapter takes a closer look at the history of electronic documents. It discusses the most influential models in the field, as developed by the various research communities described in this chapter. In particular, it discusses

- structured documents, which have their roots in the electronic publishing community,
- hypertext and multimedia documents, rooted in the hypertext and multimedia community, respectively, and
- hypermedia documents, which are often characterized by their strong roots in either hypertext or multimedia.

The different document models discussed typically reflect the research agendas defined in this chapter. Additionally, they have strongly influenced some more recent developments on the World Wide Web, which will be discussed in Chapter 3.

Chapter 2

From Structured Text to Structured Hypermedia

In this chapter, we give an overview of the basic principles underlying the hypermedia document models that have been developed by the research communities discussed in the previous chapter, and explain the main assumptions underlying the research described in the thesis. The structure of this chapter is as follows. The first section gives an introduction to one of the basic models of electronic publishing: the multiple delivery publishing model. The following four sections point out how this model is (or could be) applied to respectively: structured text, hypertext, multimedia and hypermedia documents. Chapter 3 discusses these issues in the context of the World Wide Web.

2.1 Introduction

This section discusses the basic terminology and modeling principles which are needed to understand document models and formats in general. It forms the basis for the following sections, in which we describe the current state of the art in structured text, hypertext, multimedia and hypermedia document research. Because these sections are more focused than this general introduction, they also contain the bibliographic references that are relevant to the topics that are introduced in this section.

First, we explain the basic terminology and design dimensions that are used in many discussions related to document processing. Second, we describe the central model underlying most of the discussions in this thesis: the multiple delivery publishing model, and discuss the advantages and drawbacks associated with this model of document processing.

2.1.1 Terminology

By exploring some of the design dimensions underlying many document models, the following paragraphs give an informal introduction to the concepts and terminology that are used throughout this thesis.

Documents versus applications

Traditionally, electronic publishing applications (e.g. typesetters, word processors, desktop publishing packages) are characterized by a strict distinction between the applications themselves and the documents they process: the documents contain the static data upon which the applications operate. To discriminate documents from arbitrary (digital) data, we define a document as follows:

> A *document* is a self-contained unit of information, intended to be communicated to one or more human interpreters.

This definition implicitly includes both electronic and paper documents; it also includes documents which contain different media types. It explicitly excludes, however, data which is intended to be used solely for computer-to-computer communication.

We refer to the information that needs to be communicated to the human interpreter as the document's *content*. The definition above does not restrict *how* the content is to be communicated to the user, because this is considered to be the responsibility of the application.

Traditionally, electronic documents had sequentially ordered, static content, and could intuitively be regarded as the electronic versions of their paper counterparts. The distinction between such documents and the associated applications was typically easy to make. With the introduction of hypertext and multimedia, however, many documents no longer have a printed, paper counterpart, and documents have become more dynamic and interactive. Our intuitive notions of what an electronic document should represent (i.e. the electronic equivalent of a static paper document) can no longer be applied to hypertext or multimedia documents. For these areas, a distinction between document and applications has proved to be much harder to maintain. In some early hypertext and multimedia applications, the role of the document has even completely disappeared: in these systems the document content has become an integral part of the application software. While current hypermedia applications usually make the distinction between document and application, this distinction is typically not as strict as in more traditional text-based document processing systems.

When designing hypermedia systems, there is typically a trade off between a strict separation and a tight integration of the document and the application. The former promotes reuse of the application for different documents, and longevity of the documents by isolating them from the application's implementation. The latter allows for a wider variety of interactivity and dynamic content by supporting a mix of content data with procedural, often application-specific elements.

Content versus markup

Apart from the distinction between document and application, one can discriminate different types of information within the document. The fact that the content of a document is intended for human interpretation, does not imply that documents cannot contain any other type of information. Typically, they do contain additional information, information that helps the application to communicate the document's content effectively to the user. For now, we refer to this type of information as *markup*. The boundary between markup and content is a fuzzy one, because a processing application may present information that was originally authored as markup in such a way that the end-user cannot discriminate it from the traditional content. This applies especially to *metadata*, i.e. data which contains information *about* the document and its content. For example, a document designer could decide to add versioning information to the markup of a document in order to help processing applications discriminate between alternative versions of that document. An application designer however, may decide that this information is also relevant to the end-user, and include the versioning information in the printed version of the document. As long as it is machine-readable, metadata can generally be stored within a document as either part of the content or part of the markup. The choice for one of the two alternatives is usually just a matter of taste (see Appendix B for an example).

The properties of the markup of a document depend on the type of applications that are used for processing the document (and, in an ideal world, on the applications that are going to be used in the future). Often, when new hardware or software is introduced to process a set of documents, the markup of the documents need to change as well. Sometimes, the new environment requires documents to use markup that is similar to the markup that is currently in use. In these cases, the conversion process can be (partially) automated. However, if 90% of the markup can be converted automatically, this still means that all documents need to be processed manually to correct the remaining 10%. In practice, the situation is often even worse. As new environments often add new functionality, this functionality may require markup to provide new information that could not be derived from the original document, or may require markup based on different abstractions than those used in the previous environment. In such cases, automatic conversion is generally not feasible.

In almost all cases the conversion of large document sets is an expensive, time consuming and error-prone process. Choosing the right markup scheme for a particular application is therefore an extremely critical design decision. As stated above, an important factor in the decision process is the type of applications that are used to process the document. In this respect, it is useful to discriminate between *layout-driven* and *content-driven* applications.

Layout-driven versus content-driven applications

In layout-driven applications, the layout and content of a document are tightly coupled, and there is generally no need to produce multiple versions with alternative layouts. Typical examples of layout-driven applications include the cover page of a glossy magazine or advertisements with a large amount of graphical material. Because of the tight integration of content and layout, these documents can be effectively authored directly in terms of the final presentation.

In contrast, content-driven applications focus on the content, which often needs to be presented in several ways, for example by using different layouts. For content-driven applications it is useful to separate content from presentation information, because this separation allows reuse of the same content in alternative presentations.

The distinction between layout-driven and content-driven applications does not provide a strict classification; the two types of applications should be considered as the extremes of a continuum. Still, the distinction is an important one, because applications at the content-driven side of the continuum typically require a different type of markup than applications at the layout-driven side.

Visual versus structured markup

Layout-driven applications need markup that effectively describes the visual appearance of the document's content. Such markup is commonly referred to as *visual markup*. While visual markup can be defined declaratively, this type of markup often directly employs the command language of the application's presentation system. Therefore, visual markup is sometimes referred to as *procedural markup*. The associated languages are known as visual or procedural markup languages, or in the case of text-oriented systems, *page description languages*.

In contrast, content-driven applications need structures that describe a document independently from the document's presentation. Such a structure is referred to as the *logical structure* of the document. *Structured documents* are documents that use *structured markup* to describe this logical structure. Languages that are designed to markup structured documents are generally known as *structured markup languages*, but the terms *generic markup languages* and *declarative markup languages* are also used.

To be able to present a structured document, content-driven applications need a specification that defines how the logical structure is to be presented. While this specification is sometimes implicit, and built into a specific application, more generic applications allow this specification to be defined explicitly. Such an explicit specification is usually referred to as the document's *style sheet*. Style sheets will be discussed later in this section.

Procedural versus declarative models

For text-oriented applications, the distinction between procedural and declarative models is often identical to the distinction between visual and structured markup (as suggested by the alternative terminology described above). There is, however, another dimension in the procedural versus declarative debate, which has become more important in the on-line, interactive display of hypertext and multimedia documents. Such documents often employ *scripting* to realize behavior which is not covered by the underlying (often declarative) document model. Scripting technology is especially used to obtain dynamic and interactive behavior. Today's scripting languages are very expressive, and can be effectively employed for layout-driven applications. For content-driven applications, however, the use of scripts can have major disadvantages caused by insufficient

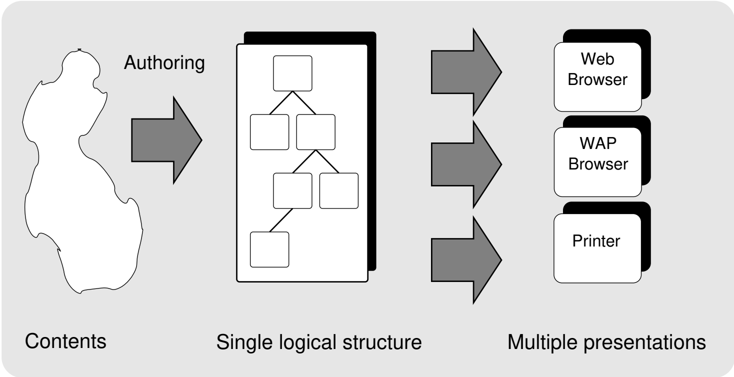

Figure 2.1: In the multiple delivery publishing model, multiple presentations can be generated from a single source.

platform, application and presentation independence. Additionally, it is hard to analyze and manipulate a document's behavior automatically if its behavior is defined by scripts. For content-driven applications, declarative models are preferred over procedural models to describe properties of documents that are often used and well-understood. This does not mean that declarative models do not need to support scripting: scripts are still required as an escape mechanism for behavior that is not covered by the declarative model, either because it is rarely used or not yet sufficiently understood to be able to develop adequate declarative solutions.

2.1.2 The multiple delivery publishing model

As stated above, content-driven applications are characterized by the need to support the generation of different presentations from a single source, as depicted in Figure 2.1. This model is often referred to as the *single source, multiple delivery publishing model*. The model implies that a document can no longer be authored in terms of a single presentation. Instead, the document has to be defined by concepts that abstract from the various presentations. Note that this is comparable with a computer program written in a high-level programming language that abstracts from the target microprocessor's instruction set. The advantages of the abstractions used in structured documents are also similar to those used in computer programs. Given that the document structure sufficiently abstracts from the presentation details, and given the existence of suitable transformations generating the target presentations, the document can be used on different output devices and output media. (This is similar to a computer program that runs on different platforms, given the existence of suitable compilers or interpreters.)

Additionally, when compared to the concepts used in the presentation format, the concepts describing the document structure are supposed to better reflect the concepts used in the problem domain. (This is similar to the concepts of a high-level programming language which, when compared to the instruction set of the target microprocessor, are closer to the concepts that are used in the problem domain.) In both cases, the choice for a particular programming or markup language depends on whether the language provides the right set of abstractions for the application at hand, and the quality of the tools that are available to translate that language to the languages of the target platforms. In the case of computer programs, this translation is typically the task of a compiler or interpreter. In the case of structured documents, *style sheets* play an important role.

Style sheets

The definition of a style sheet varies from application to application. In many applications, style sheets are, as their name suggests, strictly defining the parameters that determine the "style" of a document. That is, style sheets specify typical style parameters such as font types and font sizes, margins, use of color etc., but do not change the semantics of the content of the document.

In the realm of structured documents, however, style sheets are often assigned a much broader meaning. A typical application processing structured documents needs to deal with documents which contain hardly any explicit presentation information. Different documents may employ different logical structures, and which structures are going to be used is generally not known at the time the application is developed. Such a generic application needs more than "just" style information: it needs a specification which completely defines how a given logical structure is to be presented on a given target output medium. Ingredients of this specification will not only include style parameters as described above, but also rules that explicitly map logical structures to presentation-oriented structures, filters that reorder, select or hide information in the final presentation, or rules for adding new material to the presentation (e.g. rules that govern the generation of a table of contents or index, and rules that specify how to dynamically include material that was not available at authoring time).

These specifications may indeed have a significant impact on the semantics of the document's content as perceived by the end-user, and referring to such specifications as "style sheets" may thus seem counterintuitive. Though alternative names have been proposed, including "action sheets", "behavior sheets" and "transformation sheets", these terms are not common usage (yet). Throughout this thesis, we use a broader definition of style sheet as it is commonly used in a structured document context:

> A *style sheet* is a (machine readable) specification that maps one or more logically structured documents to a particular presentation structure.

As depicted in Figure 2.2 on the facing page, style sheets are frequently applied to a set of documents which share a similar structure. In this way, both documents and style sheets are subject to reuse: the content and structure of a single document can be reused in multiple presentations by applying different style sheets, and the style information can be reused to produce consistent presentations by applying the same style sheet to multiple documents.

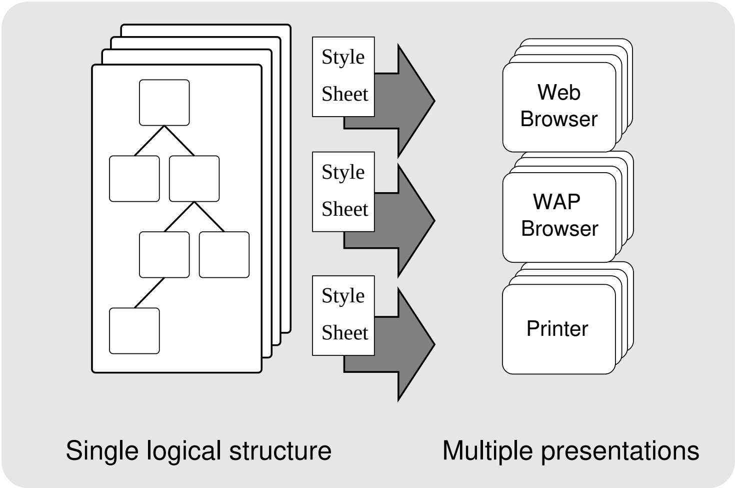

Figure 2.2: A style sheet maps the structure of one or more documents onto a specific type of presentation.

Although the term “style sheet” is used less frequently in areas other than text processing, some of these areas use a model that is similar to the multiple delivery publishing model. In adaptive hypertext, for example, techniques have been developed to automatically adapt the presentation to the specific task and knowledge of an individual user. In this case, the focus in the source document is not on abstracting from the differences in style and layout, but on abstracting from differences between individual end-users. In networked multimedia systems, quality of service (QoS) negotiation is used to design techniques that effectively present multimedia documents in an environment with limited network or playback facilities. In both cases, the models and techniques used share many similarities with the multiple delivery publishing model and will be discussed in more detail in the following sections of this chapter.

Advantages of the multiple delivery publishing model

One of the reasons why content-driven applications and structured markup have become popular in the publishing industry, is that they fit more naturally in their traditional production chain. It allows (and sometimes even forces) authors to focus on the content and structure of the document, while the actual appearance of the document in print is determined by a trained designer or typesetter. Additionally, structured documents have been used in other industries where technical documents needed to be reused in different contexts. More recently, the advantages of separating the structure and presentation of documents have been recognized by a much wider audience. Support for style sheets for example, has become a standard feature of many word processors and Web browsers.

As stated above, decoupling structure from presentation not only allows reuse of the basic document structure by applying different style sheets to a single document, but also promotes reuse of the style information by applying a single style sheet to multiple documents. This simplifies maintaining a consistent style across a potentially large set of documents and reduces the effort required to maintain both the documents and the style information. In distributed environments such as the Web, the reduced redundancy also decreases the down-load times and amount of network bandwidth needed to retrieve these documents [174]. In general, the advantages of structured documents can be described in terms of *longevity*, *reusability* and *tailorability*.

- **Longevity** — When compared to other electronic document formats, structured documents are usually less sensitive to changes in their environment. This is an important issue when longevity is essential. In domains where documents outlive the systems used for their preparation, formatting or archiving, electronic documents frequently cause problems similar to legacy software. In financial domains, for instance, government regulations often require archiving of financial documents for a period which exceeds the typical 4-5 years lifespan of a document processing system. In the aircraft industry, technical manuals for a specific type of plane need to be maintained and remain accessible, at least for the period in which the plane is in operation (typically more than 30 years).

  In such domains the amount of information stored in electronic documents is too large and too diverse to allow for a practical conversion of all data when (part of) the processing system is replaced or upgraded. As discussed above, converting all documents to the new format tends to be very expensive, time consuming and error prone. Additionally, because structural information is often coded implicitly into the documents, it may very well lead to loss of essential information.

  Even when the software and hardware environment is stable, the use of structured documents may have advantages in terms of document maintenance. Their high level of abstraction often simplifies modification of the document structure, and these modifications can be automatically reflected in the various target presentations. Changes to the visual appearance are also easier to make, because presentation information is localized (for instance, by specification in the style sheet), and can be manipulated independently from the actual content.

- **Reusability** — The platform and layout independence of structured documents increases the chance that (fragments of) a document can be reused in a different context. Structured markup is often used in systems that support on-the-fly generation of different versions of a document by assembling independent fragments stored in a database system. Besides being independently specified from the output device (e.g. a specific printer), structured documents can also be specified independently from the output *medium*. This allows automatic formatting of the same material for traditional paper printout, on-line presentation, CD-ROM application, etc. Medium independence also enhances the accessibility of documents because it simplifies the automatic generation of braille and speech synthesized versions of a document in order to make information available to, for instance, visually impaired users. Reuse of information by other systems is also facilitated by platform and medium independence, which is important in open systems where interoperability in a heterogeneous environment needs to be ensured.

- **Tailorability** — Documents that use visual markup are typically suited for only one process: presentation (either by sending the document to a printer or displaying on a screen). Because structured markup does not need to specify the document's visual appearance, it can provide other information which may be useful to other types of processing. For example, in information retrieval, having access to the logical structure allows for more powerful queries (e.g. ”select all articles whose first chapter is classified as “top secret”)[1].

  Because structured documents are not defined in terms of an output format, they can be defined in terms that better match the specific semantics of the application domain. A document designer may define a domain-specific document structure in a way that is similar to the way a database designer defines a domain-specific schema. As such, structured documents support applications that require information that is not expressible in more “general purpose” document formats such as HTML or RTF. For example, standards such as SGML and XML are especially developed to support the definition of document formats that are based on domain-specific markup.

Disadvantages of the multiple delivery publishing model

The advantages in terms of longevity, reusability and tailorability discussed above, do not come for free. The price that has to be paid is generally in terms of increased complexity and major initial investments.

First of all, a suitable document model for the application at hand may not be readily available. Especially for applications that require domain-specific documents, it is likely that a new document model, tailored to that domain, needs to be developed. It may be difficult and expensive to develop a model that

- is sufficiently expressive to describe documents in a specific domain,
- abstracts from all differences among the target presentation environments, and,
- still allows (semi)automatic translation from the abstract model to the final presentations.

The development of such a model can only be successful if both the application domain and the presentation environments are thoroughly understood, and even then the development process will typically require several iterations.

[1] In some cases, access to the logical structure and/or a particular presentation structure might be required: ”select all articles with two figures on the bottom of the third page, and the following keywords in the abstract:. . . ”. While not impossible, such queries are more difficult to resolve because the logical structure and a specific presentation structure have to be combined in order to answer the query.

Second, after an adequate document model has been developed, authors need to be able to create and maintain documents in terms of the new model. This typically requires the development of new authoring tools, and extensive training of the authors, who are used to authoring in terms of the final presentation, and not in terms of a set of underlying abstractions.

Third, tools need to be developed to translate the structured documents to the final presentations. Typically, the quality of these generated presentations need to match, up to a certain level, the quality of the manually authored presentations. This is not only to satisfy end-user requirements, but also to satisfy the authors, who need to give up control over the final presentations. Regardless of all the advantages mentioned above, authors are unlikely to accept a new production environment if it produces presentations of an inferior quality. While one of the reasons for developing structured documents is to make documents less dependent on a specific hardware or software environment, conversion tools play an extremely important role in supporting the processing necessary to present the documents to the end-user. Therefore, the success of introducing structured document technology into a specific organization is highly dependent on the quality of the associated tools.

2.1.3 Structured documents in hypermedia

To summarize the above, deploying structured documents is only worthwhile for content-driven applications, and even for these applications, one needs to verify that the advantages outweigh the disadvantages. It is safe to say that for text-based applications, this is more and more the case. Properties such as longevity, reusability and tailorability have become increasingly important, and as application domains and presentation environments become better understood, the much needed models and tools are becoming generally available.

This trend, however, hardly applies to hypermedia systems. The current generation of multimedia and hypermedia authoring systems makes minimal use of structured documents or the multiple delivery publishing model. Hypermedia documents are still directly specified in terms of the final presentation. This is not surprising, because many hypermedia applications are considered to be layout-driven, and for these applications it is unnecessary or even infeasible to decouple the layout of the document from its content. But not all hypermedia applications are layout-driven. General examples of content-driven hypermedia applications include interactive manuals, product information catalogs, and multimedia encyclopedia. Other examples include hypermedia applications on the Web which, in the very near future, need to be able to adapt to different output platforms. Finally, many of the currently text-based applications that successfully employ structured document technology, require extensions to include link-based navigation and multimedia content.

For this type of hypermedia applications, structured documents have potentially the same advantages, in terms of longevity, reusability and tailorability, as they have for text. It is, however, exactly this type of applications that is minimally supported by the current generation of hypermedia systems. This is partly due to a lack of appropriate concepts to abstract from the presentation details in hypermedia systems. Due to our long experience with text, we have developed a rich and commonly accepted set of abstractions for describing the logical structure of a text, which is to a large extent independent of the final presentation. For hypermedia, we still lack such a set of commonly accepted abstractions, but this may change when authors gain more experience in authoring different hypermedia documents.

Though suitable hypermedia abstractions have been developed for some particular application domains, very few models or tools support authoring hypermedia documents in these abstract terms. The tenet of this thesis is that it is possible to benefit from the advantages of the multiple delivery publishing model, not only for text-based applications, but also for content-driven, time-based hypermedia applications. In order to develop the necessary set of models and tools, we use the document models and tools that have been developed by the research communities we described in Chapter 1 as a starting point. We describe the current state of the art in electronic text publishing, hypertext and multimedia technology. The last section of this chapter analyzes the possibilities of integrating these technologies and develops a framework for multiple delivery publishing of time-based hypermedia documents. The following chapter discusses the application of this publishing model in the context of the World Wide Web.

The remainder of this chapter discusses structured hypermedia documents and their applications, focusing on the difference between structured text and structured hypermedia. To explain the basic concepts underlying structured hypermedia documents, we first discuss the use of structured documents in linear text documents. Then we take a closer look at hypertext and multimedia documents in order to come to a list of requirements for a model supporting multiple delivery publishing of structured hypermedia documents.

The structure of the remaining sections of this chapter is reflected in Table 2.1. In this table, we partition the space of electronic documents along two dimensions. First, we

	Static media (Section)	**Time-based media** (Section)
Linear structure	Text (2.2)	Multimedia (2.4)
Non-linear structure	Hypertext (2.3)	Hypermedia (2.5)

Table 2.1: Classification of electronic documents.

discriminate between documents containing an essentially linear flow of information, and hyperdocuments, allowing users to navigate though the information that has an essentially non-linear structure. Second, we discriminate between documents containing static media (e.g. text, still graphics) and those built of time-based media (e.g. audio, video, animation) and other media items that need to be synchronized. This partition leads to four classes of documents, and the following four sections will discuss text, hypertext, multimedia and hypermedia documents, respectively. Note that there is a significant overlap in these four areas; issues relevant to linear text documents usually apply to hypertext documents as well, the same goes for multimedia and hypermedia documents.

2.2 Structured Text Documents

The previous section introduced the fundamental ideas, design dimensions and terminology related to structured documents and the single source, multiple delivery publishing model. Before we discuss these ideas in the context of hypermedia documents, we first explain their application in the context of linear text documents, the application area for which these ideas were originally developed. We discuss models and techniques that have been developed for processing structured text, and point out which aspects of these models need to be changed to make them applicable to hypertext and multimedia. Finally, we discuss two standards that have significantly influenced the processing of both structured text and structured hypermedia documents: SGML and DSSSL.

2.2.1 Research issues

Research on structured text documents has mainly been carried out in the context of electronic publishing, and has a strong focus on a cost-effective production of large volumes of high-quality documents in an industrial setting. Many of the topics described in the previous section are typical research topics in this area. While originally developed for text, they can potentially also be applied to hypermedia.

The term *document engineering* is sometimes used because the research questions in this area are often similar to those in software engineering. Issues related to document maintenance (including such problems as "legacy" documents, platform dependency, lack of appropriate structuring mechanisms, etc.) are in many ways comparable with those related to software maintenance [39]. The same applies, to a certain extent, to document reuse, interoperability and standardization.

In contrast, most research related to document formatting addresses topics that are typical for text documents, including modeling line and page breaking, kerning, justification, hyphenation, and other topics related to typography. Traditionally, most research has focused on western languages. More recently, typesetting of non-western languages and other internationalization issues have also become a major topic of research. Typical topics include non-western character sets and font design [84, 117], alternative writing directions [20], case-sensitivity, language-specific hyphenation rules [116], and mixing multiple languages within a single document.

2.2.2 Modeling structured text

As stated in the previous section, the essence of the multiple delivery publishing model is the distinction between the notion of a document, and the way that document is to be presented. In Figure 2.1 on page 13, we sketched a three level model with two

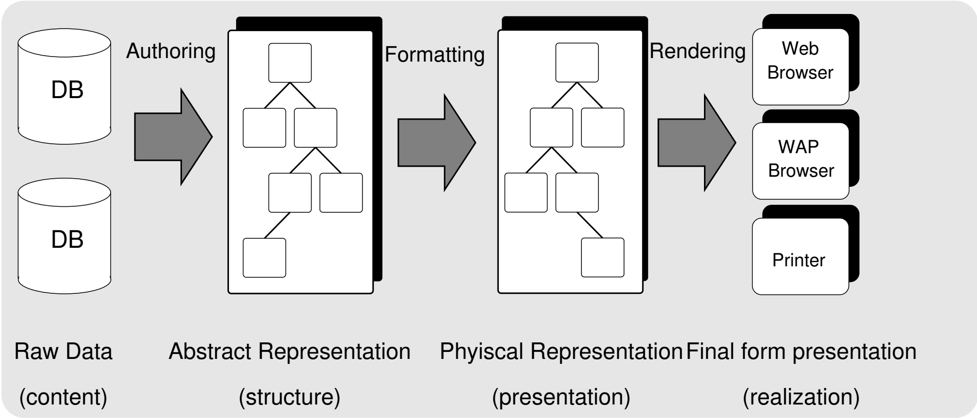

Figure 2.3: Four level version of the multiple delivery publishing model.

important phases to go from one level to the other. In the first phase, the authoring process, the content is selected and structured to form the structured document itself. During the second phase, the formatting process, one or more presentable versions of this document are generated.

For both theoretical and practical reasons, however, it has proven to be useful to further decompose the second phase into two distinct steps. This leads to a refined, four level model, with three phases: authoring, formatting and rendering. Each phase results in a particular representation of the document, as depicted in Figure 2.3. Based on Furuta [103], we describe these phases as follows:

1. **Authoring** — The first phase involves selection and/or creation of the contents, and the creation of the logical structure by the author. The result of this phase is called the *abstract representation*, which describes the document in terms of its logical structure and is the structured document proper. We generally use the term "document" to refer to the result of this phase, unless the context indicates otherwise. As stated before, languages used to describe structured documents are known as *generic markup languages* or *declarative markup languages*.

2. **Formatting** — In the second phase, the appearance of the document is determined, i.e. the elements identified by the logical structure are mapped onto a particular presentation model. If this mapping is defined in a separate, machine readable specification, we refer to this specification as the *style sheet*. Conceptually, the result of this mapping is an intermediate format, the *physical representation*. Preferably, this representation defines the layout constraints and presentation style in a declarative and device-independent manner.

3. **Rendering** — In the last phase, the presentation is realized by transforming the physical representation to the particular output format expected by the target output device (e.g. a specific printer). Note that while the concrete output format may differ, both the presentation and the realization describe the appearance of

the document. The resulting presentation might need, for practical reasons, extra work in an additional post-production phase. In theory all work is done in the earlier phases, and we often refer to the result of the realization phase as being the *final form presentation*.

Conceptually, the borders between these three phases are properly defined: all decisions concerning the document's content and the underlying logical structures are made during the first phase, all decisions influencing the appearance of the document are made in the second phase, and the only objective of the third phase is to effectively realize that appearance in the required target output format.

In practice, however, there is often no sharp boundary between the three phases. For example, while low-level formatting decisions such as those concerning hyphenation and justification influence the document's final appearance, they are typically not made during the formatting phase, but deferred to the rendering phase. The exact page number, for example, on which a specific paragraph will be placed, can in general not be determined until the last phase. Additionally, the main task of the intermediate model (the physical representation) is to abstract from the details of the final-form output models. If such an abstraction does not exist, or is not sufficiently expressive, structured documents are directly mapped onto the final presentation. Even if all three phases are explicitly implemented, the author does not necessarily need to be aware of this: in many WYSIWYG applications these phases are seamlessly integrated into a single interface.

Still, it is useful to distinguish these three separate phases. Distinguishing the first step from the two others has become common practice, and the associated pros and cons have been discussed in the previous section. While the distinction between the second and third phase is less common, it has advantages too, both on a conceptual and practical level. Conceptually, the distinction allows the design of the appearance of the document to be carried out independently from the limitations of a specific output format. This prevents decisions regarding the appearance being mistaken for decisions regarding the realization of that appearance within a specific output target, and *vice versa*. In practice, this distinction allows the development of document formats and tools in which the same layout specification (e.g. the style sheet) can be used for generating similar presentations using different output formats. We discuss these issues in more detail below.

First, we discuss the models related to the results of the first phase (structured document models), then we discuss the models related to the results of the second and third phase of the document preparation process (document formatting models).

Structured document models

Document models for structured documents can be measured along an axis with domain-independent models at one end and domain-specific models at the other. A very well known example of a domain-independent model is the document model of the World Wide Web, as defined by HTML (HTML is discussed in Chapter 3, page 76). A more complex example, Docbook [67], defines an SGML document model that is frequently used for technical documentation in several domains. Other, non-SGML, examples of domain independent models include the models underlying most style templates in commercial word processors and the many standard document classes offered by the LaTeX structured text formatting and typesetting system. These LaTeX classes model typical document genres such as articles, reports, books and letters.

Domain independent models typically provide two types of abstractions. The first is specific for the document's genre (e.g. if the genre is a letter, typical structures may include the letter's main body and an optional P.S. at the end of a letter). Since these abstractions are often used to determine the overall structure of the document, the term *holistic structure* [163] is also used in the literature. Additionally, these models provide a set of generally useful, and often well-known abstractions that are to a large extent independent of the genre. Examples include sections, paragraphs, ordered and unordered lists, emphasis, footnotes, etc. These abstractions can be used to markup substructures of the document (e.g. the text within the body and the P.S. of the letter).

At the other end of the spectrum we find domain-specific document models. These models are often defined by industrial consortia to enhance document exchange among their members. These documents models are also used to structure data in computer-to-computer communication. EDI (electronic data interchange) applications are a good example [211, 230] where document-oriented models provide abstractions that are tailored to a specific domain, abstractions that may only be meaningful to domain experts. The material in the documents is usually highly structured, and this structure represents semantics relevant to the application domain.

While the data is highly structured when compared to typical domain-independent, document-oriented applications, the material is often *not* quite as structured as the data used within a typical database application. Storing material in documents using a domain-specific model is often used as an alternative to storing the same information in a (semi-structured) database, or to exchange structured data across different databases (this issue will be further discussed in the context of XML in Section 3.2.3).

The difference between domain-independent and domain-specific document models often has consequences for the formatting process and the requirements for a style sheet language. For example, most documents using domain-independent models have a coherent, linear narrative. This limits the freedom a style sheet designer has in deciding how the layout of these abstractions should look. Only style parameters, such as font styles and margin styles, can be varied without changing the semantics of the document. Reordering material usually breaks the narrative. Additionally, it is usually convenient to have a default layout and style for the most frequently used structures. As a result, many style sheet languages that are geared towards domain-independent documents are relatively simple, because they only manipulate the properties of a given default layout.

In contrast, most domain-specific document structures do not have a clear overall narrative. For such documents, the mapping from the domain-specific abstractions to a specific layout is not as straightforward as for the well-known abstractions found in many domain-independent models, and there are often no appropriate defaults. To generate the desired presentation for a domain-specific document, style sheets need to explicitly map each logical structure to a specific presentation structure. Additionally, style sheets often need to reorder material, hide information which is not relevant in the context of a particular presentation, or bring in new material originating from other sources.

Style sheet languages that are designed for domain-specific documents are thus required to be far more expressive than more traditional style sheet languages, and these languages often feature a full-fledged document query and transformation language to be able to select the desired material and to present it in the appropriate order. Note that the extra features of these style sheet languages can also be used to specify traditional text processing activities other than formatting. Typical examples include style rules to generate a table of contents or an index.

Document formatting models

Formatting can be described as a conversion process, in which a structured document is converted to a specification that defines the visual appearance of that document. One way of describing the differences between formatting models is to look at the differences between requirements for domain-independent and domain-specific document structures, as described above. Another way is to look at the differences between the result of associated conversion processes. Depending on the result of the conversion, we can differentiate between three different conversion processes:

1. from a structured document to an abstract output representation,

2. from a structured document to a concrete output representation, and,

3. from a structured document to another structured document.

Below, we discuss the need for, and the differences between, these three different conversions.

The first conversion process conforms closely to the phases used in the refined version of the multiple delivery publishing model illustrated in Figure 2.3 on page 21. Here, the result of the formating process abstracts from the format expected by a particular output device. Note that associated formatting models need to cover at least two different levels of abstraction: the concepts used in the source, the structured document, that abstract from the appearance, and the concepts used within the representation that results from the formatting process. These concepts describe the appearance but abstract from the output format of the target output device. Within a formatting model, the concepts in the second level of abstraction are often referred to as *formatting objects*, or, when dealing with text-oriented models, as *flow objects*. The latter often describe well-known typesetting abstractions such as pages, columns, paragraphs, font characteristics, etc.

The obvious advantage of this approach is that the formatting process can be described independently of the target platform and/or output format. In this way, style sheets need only provide a mapping from the logical structures of the document to a set of formatting objects specifying the presentation. These style sheets can be easily interchanged and reused across multiple platforms and output formats. All details related to a specific platform or output format are isolated in the back-end processor that transforms the formatting objects to the target output format. An additional advantage is that the implementation of this back-end processor (that realizes a given appearance within the context of the target output device) can be defined independently from the style sheet language and formatting engine used in the previous phase.

A disadvantage however, is that this approach relies on the existence of a rendering application (i.e. the formatting back-end) which is able to convert the abstract representation to the target output format. Another disadvantage is that the style sheet cannot take advantage of specific features of the output device, unless there exists a corresponding abstraction in the intermediate model. This makes a sufficiently expressive intermediate representation a key requirement (or even the bottleneck) of this approach.

When a suitable intermediate format, or the tools to convert from an intermediate format to the target format, do not exist, the conversion process sketched above cannot be used. Therefore, many formatters use the second type of conversion process, in which the intermediate format is bypassed and the appearance of the document is specified directly in terms of the target output device. The advantage of this approach is that style sheets are able to employ all features of the target output device. The disadvantage is that these style sheets can only be used to generate presentations for this particular output device. To realize the same output on a different output medium requires development of a new style sheet. Additionally, two separate problems have to be addressed in a single style sheet: the specification of the appearance of the document, and how that appearance is realized in the format expected by the output device.

The two different conversion processes sketched above describe the two basic formatting processes. However, there is still a need for a third type of conversion process: the conversion from one particular type of structured document to another one. This type of conversion is often referred to as a *document transformation*, or, to stress the structural aspects, a *document tree transformation*.

Transformation typically involves transforming the logical, abstract structure of the document into another abstract structure, independent of layout and style issues. Document transformations are useful in a number of situations. First, transformations are used for all kinds of document conversions which do not necessarily involve formatting, e.g. conversion of a document using an in-house developed format to a standardized format and *vice versa*. Here, we focus on another application of document transformation: as a pre-processing step in the formatting process. As discussed above, some applications need to reorder, or restructure, the contents of a document to generate the appropriate presentation. If extensive restructuring is required (a process very different from the formatting process itself), it may be useful to carry out the necessary transformations as a separate phase in the total document preparation process (see Figure 2.4 on the next page). Since both the input and the output of the document transformation process is a structured document, transformations can be easily chained, as many times

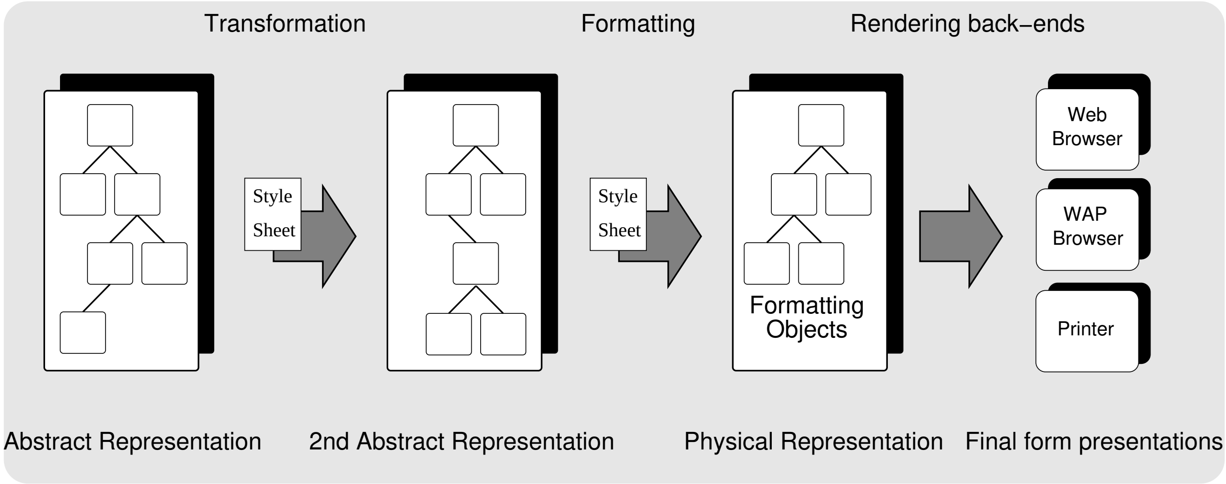

Figure 2.4: Document transformation used as a pre-processing step in the formatting process

as necessary (e.g. the first run may add a table of contents, the next run an index, etc).

Because the use of document transformation differs significantly from the use of document formatting, one can argue that the two processes should be controlled by different languages. Whether this is true is generally a matter of taste. Another argument for separating the transformation language from the style sheet language is that transformations are typically used only in the production process of a document. After the document is published, no more transformations need to be carried out by the client application. The client only needs to support the formatting language to be able to adapt the style to the preferences of the end-user, and the extra functionality needed to support transformations makes the client's style engine unnecessarily complex. For those applications that do need both document transformation and formatting functionality, the required functionality is likely to overlap (since both describe a conversion of structured documents) in which case it may be more effective to combine the two into a single language.

To further complicate the picture, one can abstract from the differences between structured documents, abstract output and concrete output presentations and assume that in practice, all three data structures can be regarded as labeled trees. From this perspective, all three conversion types can be described as tree manipulations, described using a single language (this is one of the assumptions underlying XSL, the style and transformation language designed for XML). We discuss the associated advantages and disadvantages more elaborately when discussing the style sheet languages DSSSL (in Section 2.2.4), CSS and XSL (in Chapter 3).

2.2.3 Relationship with other document models

The conceptual distinction between content, structure, presentation and realization seems to be generally applicable. The document and formatting models described above, however, have been developed for text-oriented applications. In this section, we discuss which aspects of these models are specific for text, and which aspects can be readily applied to hypermedia.

Document models

The examples of document models discussed above, especially the domain-independent models, are all typical examples of text-oriented documents. All feature a hierarchical document structure on top of a basically linearly ordered text-flow. We call this linear order in which the data appears in the document the text's *lexical order* [200]. The lexical order of the main text-flow[2] typically reflects the document's narrative. To avoid changing the semantics of such a document, the lexical order has to be preserved by the style sheet. The text is thus presented to the user in an order that is the same as (or at least highly similar to) the lexical order. While this is often a desired feature of a text-oriented document, the fundamental role of the linear text-flow within these models prevents them from being readily applied to hypertext documents (which have an essentially non-linear structure) or to multimedia documents (which are not based on a text-flow model).

Note that even HTML, the hypertext markup language of the Web, uses a document model which is based on a linear text-flow. Hypertext links within a single HTML document can only be used to navigate along a one dimensional, linear text-flow. Non-linear hypertexts need to be modeled indirectly, for example by combining multiple HTML pages or by employing scripting. Examples of non-linear hypertext document models are discussed in Section 2.3 on hypertext.

Multimedia documents, in contrast with hypertext documents, often have a linear structure which seems, at first sight, comparable with the linear structure of text. The main difference is that for multimedia, the linearity is along a single temporal dimension, and not along a single text-flow dimension. This difference however, has a significant impact on the formatting model, and usually also affects the multimedia document model itself. While many of the basic layout properties of a text document can be automatically derived from the document's lexical text-flow, for multimedia neither the temporal order nor the lexical order of the media items in the document provides much information about the spatial layout of these items. Therefore, spatial layout information which can often be left implicit in a text document, needs to be modeled explicitly in multimedia documents. Both spatial and temporal layout are discussed in more detail in Section 2.4 on multimedia.

While most well-known domain-independent document models are only applicable to text, some examples of similar models for hypertext, multimedia and hypermedia do

[2]Besides the main text-flow, which contains the body of the text, documents often use other text-flows, containing front matter, marginalia, side-bars, footnotes, etc.

exist. Due to our lack of experience with these new media types (when compared with our long experience in authoring and printing text), the latter are often not as mature and sophisticated as the models that are in common use for text. For example, authoring tools may offer pre-defined templates to facilitate authoring of the more common types of hypermedia presentations, such as guided tours (hypertext) or slide shows (multimedia).

As discussed above, many examples of more domain-specific document models do not have the strong overall narrative which is characteristic of many documents employing a domain-independent format. For this type of documents, the lexical ordering of the text in the document does not necessarily need to be the same as the order in which it is presented. For example, the lexical order of the items in a document describing a product catalog does not (necessarily) need to be preserved in a particular presentation of some product items. Neither does the order of the items provide much information to a formatting application about the spatial layout of the individual items. Because the linear, lexical order of these models is often not as essential as in the domain-independent models described above, it is much easier to apply them in a non-linear hypertext environment. For example, the product descriptions are likely to make sense, even if they are read in a different order when a user is browsing the hypertext version.

To apply the techniques discussed above to hypermedia, adapting the document models is necessary, but not sufficient. The associated formatting models — and tools — also need to be adapted.

Formatting models

Formatting text-based documents involves the transformation of an essentially linear stream of text into a sequence of pages. This makes line and page breaking basic formatting processes for text documents. Apart from distributing words across several lines, the associated algorithms usually need to deal with kerning, justification and hyphenation. Page breaking (or *pagination*) typically involves filling a sequence of pages with material from several text flows (e.g. body text, floating material, footnotes, margin notes). Other typical issues related to formatting text include font design, ligature support, encoding of graphical, tabular and mathematical information, floating material, automatic generation of table of contents and indices, and bibliographical and cross referencing. See Furuta [103] for an overview.

The delicate details of these text-specific formatting processes are complicated and beyond the scope of this thesis. For the following discussion, however, it is important to note that text formatting is essentially based on the linear structure of one or more text flows, and that this structure is reflected in the lexical order of the document's content. As discussed above, this characteristic prevents many formatting models, languages and tools that are designed for text-documents, from being applicable to hypertext and multimedia documents. Fortunately, hypertext documents do contain linear textual fragments, to which the more traditional formatting models can be applied. Additionally, even strictly linear text may contain concepts such as a table of contents, indices, footnotes, bibliographic and cross-referencing. These concepts have been part of text formatting for a very long time, and can easily be used for navigation when presented in an on-line hypertext environment. Still, most text-based models require extensions, both to deal with the non-linear structure of the source document and target document and to be able to generate the navigational interface in the target output format.

While text and multimedia layout differ fundamentally, some techniques developed for text can also be applied to multimedia. The "glue-based" filling model of Knuth's TEX [149] system, for instance, is often quoted, and used in contexts other than text (e.g. for temporal alignment of multimedia documents [148] and the spatial layout of windows in graphical user interfaces [157]). Multimedia-specific layout issues will be further discussed in Section 2.4 on multimedia documents.

In the remainder of this section we discuss two international standards: SGML and DSSSL. These standards have been developed for structured text processing, but have also had a significant influence on many hypermedia document and formatting models.

2.2.4 Standardization

The standards that are the most relevant to electronic text documents are the suite of standards related to ISO's Standard Generalized Markup Language (SGML [130] and the Office Document Architecture (ODA [131]). Since ODA is primarily geared to office documents (e.g. letters and reports), and very few implementations of ODA have been realized, we will focus on the standards related to SGML. In this section, we focus on the fundamental ideas underlying SGML and the type of software needed to process SGML documents. An introduction to the SGML syntax itself is given in Appendix B, while Part III of the thesis goes deeper into the software architecture of SGML systems.

SGML's roots are in GML, a markup language defined by Charles Goldfarb and others at IBM. As any other automated office department, Goldfarb's department was frequently subjected to changes to the document processing environment. These changes resulted in many tedious conversion processes, needed to convert all electronic documents to the requirements of the new environment. To avoid these conversion processes, Goldfarb strived for a document format which was geared towards the requirements of the information structures contained within the document, and not geared towards the requirements of the hardware, software or final form delivery format. In other words, Goldfarb was striving for a platform and layout independent markup language.

Realizing that no single language could ever meet the requirements of all the different type of documents used within his organization, a generalized (meta)markup language (GML) was developed. GML, and its standardized successor SGML, allows document designers to define concrete markup languages tailored to the needs of their specific domains. Such a concrete language is defined by a *Document Type Definition* or DTD (see also Appendix B).

While tool support is a topic that is somewhat underspecified in the standard, it has always played an important role in the everyday use of SGML, for several reasons. First of all, SGML's notorious complexity has made tool development (unnecessarily) complex. But there are two, more fundamental reasons.

First, since most SGML documents are not readily printable, conversion tools are required when SGML documents are to be printed. SGML systems are typically used by organizations that implement the single source, multiple delivery publishing model discussed in the beginning of this chapter. In particular, SGML is used to markup the "single source" document that forms the basis of the model. As discussed before, one always needs to convert these types of documents to one or more presentable versions in order to communicate the content of the document to a human user. Tools that perform this conversion have, in general, been complex, expensive and cumbersome to use. This has led to the (paradoxical) situation that while SGML was developed to support application-independent document markup, the applications needed to print and display SGML documents have played a very important role in the everyday use of the standard[3].

Second, developing SGML tools is complex because while many other document processing tools adapt their native file format to the requirements of the application, most SGML tools need to adapt to the requirements of the document. As discussed before, a domain-dependent structured markup language provides high-level markup constructs tailored to the specific domain at hand. These constructs typically change over time and are generally not known at the time the application is developed. This means that the processing software has to be able to adapt (within some very general limits, defined by the SGML standard) to the document format defined by a document designer. Note that this is the exact opposite of the traditional scenario where a (proprietary) document format reflects the features (and often the internal data-structures) of the software used to process the document, and the document format is thus defined by the designer of the processing software. This fundamental and inherent "customization" aspect often complicates the development of SGML tools.

Basic SGML software: parsers

To avoid having to develop software for every particular SGML language from scratch, one needs generic SGML software components. That is, components that can be used to process documents in any SGML-defined markup language. Using these generic components, one can build the necessary language-specific tools (see also Part III of this thesis).

One of the most basic software components in any SGML system is the SGML parser. In theory, the task of the parser is relatively simple: it has to read a structured document and its associated DTD, validate the document against that DTD, and report the contents of the document (and possible validation errors) back to the application. In

[3]For years, the most frequently asked question on the `comp.tex.sgml` newsgroup was: "I received an SGML document. How do I print it?". While certainly an obvious and legitimate question, it proved to be very hard to answer...

practice, SGML's notorious complexity, the terse style and sometimes arcane terminology used in the standard specification, and its many optional features, has made this task extremely complicated, even to the extent that at the time of writing, no single parser fully implements the SGML specification.

To avoid the overhead and complexity associated with a generic SGML parser, many applications use special purpose parsers that only parse the particular markup language used by the application. Typically, these parsers do not implement many mandatory SGML features, and are less strict when it comes to document validation. In general, these systems are not interoperable with other SGML systems. The HTML parsers that are shipped with most of today's Web browsers are prototypical examples of such non-interoperable systems.

Other SGML software

After the parsing and validation phase, an SGML document needs further processing. As discussed above, to be printed or displayed on a screen, tools are required to format the document into a presentable version. Other forms of processing (archiving, indexing, querying, converting, etc) also require tools. Only having an adequate parser is not sufficient for processing SGML documents.

There are two design approaches to developing the other tools that are required. The first, most practical, approach is to build a special-purpose application on top of the parser, which understands the semantics of the markup language defined by a particular DTD. The second, more generic, approach is to build an application that accepts any valid SGML document, regardless of which DTD it conforms to, and is able to perform some specific processing on the basis of an external specification, for example a style sheet.

In both approaches, the development of tools on top of an SGML parser is complicated by the fact that SGML does not specify the application programming interface (API) between the parser and the application, nor does it provide a formal document model that can provide the basis of such an API. The lack of a standardized API makes it virtually impossible for a particular application to switch from one parser to another. While this problem severely limits the flexibility of tools developed using the first approach, a common, well-defined document model has turned out to be an absolute prerequisite for the development of the more generic tools based on the second approach. These tools needed to be able to unambiguously address arbitrary fragments of structured documents, for instance to apply a style rule to a particular fragment, to use the fragment as an anchor in a hyperlink, or to manipulate the fragment in a scripting application. Such an unambiguous addressing technique turned out to be hard to define on the basis of the SGML specification alone, and required the development of a more formal document model.

Such a model has only recently been developed, and is known as the *grove* model. Groves are used by some other, SGML related, ISO standards and are discussed below. A more Web-oriented alternative is the *Document Object Model* (DOM) discussed in

Chapter 3.

SGML formatting: DSSSL

Formatting tools can, as discussed above, be specifically developed for a particular DTD. The advantages of this approach are that it is relatively simple, and the formatting process can easily be optimized for the target DTD. The disadvantages are that the application needs to be adapted every time new features are added to the DTD, and new applications need to be developed for every new DTD that needs to be supported.

To fully benefit from the advantages of SGML, one needs a more general and standardized approach that can be used to format generic SGML documents, an approach that is independent of the concrete markup language defined by a particular DTD. The specification of such a formatting process, however, is not covered by the SGML standard. The lack of standardized formatting specifications at the time the SGML standard was published (in 1986) has had a very negative impact on the interoperability of SGML systems. While the documents themselves could be easily interchanged across various platforms, the specifications or applications needed to print or display the document were platform and vendor specific. This has changed only recently, with the publication of a new ISO standard in 1996, ten years after the publication of SGML. One of the reasons this took so long is that it proved to be very difficult to come up with a language that could be applied to the very broad range of SGML documents that were in use. The new standard, the Document Style Semantics and Specification Language (DSSSL [136]) now provides a standardized way to assign semantics to the syntactical elements of an SGML document.

One of the objectives of DSSSL is to be able to describe these semantics in a platform independent way. To achieve this, DSSSL defines two independent processes similar to the phases depicted in Figure 2.4 on page 26.

The first is the transformation process, and is used to convert a document conforming to one SGML DTD to a document conforming to another one. Such transformations are used, for example, for converting company specific documents to standard conforming ones, or for a "down translation" of a complex and rich document to a more easily processable one (e.g. conversion from a domain-specific DTD to HTML).

The second process is the formatting process. To be able to develop platform independent style sheets, DSSSL introduces an abstract target representation to convert to. This abstraction, the *flow object tree*, is a flow-based model, typically modeling a sequence of pages or a single scrollable window. Objects in the tree are, for example, column, header and footer objects, paragraph objects, anchor objects etc. All objects have a specific set of properties which contain additional formatting information. Note that while the set of flow objects and their associated properties is extensible, using such extensions requires modification of the back-end processors used and thus limits the interchangeability of the document and its style sheet.

Prior to both the transformation and the formatting process, a preprocessing step is carried out, which is known as the *grove-building* process. This involves the parsing

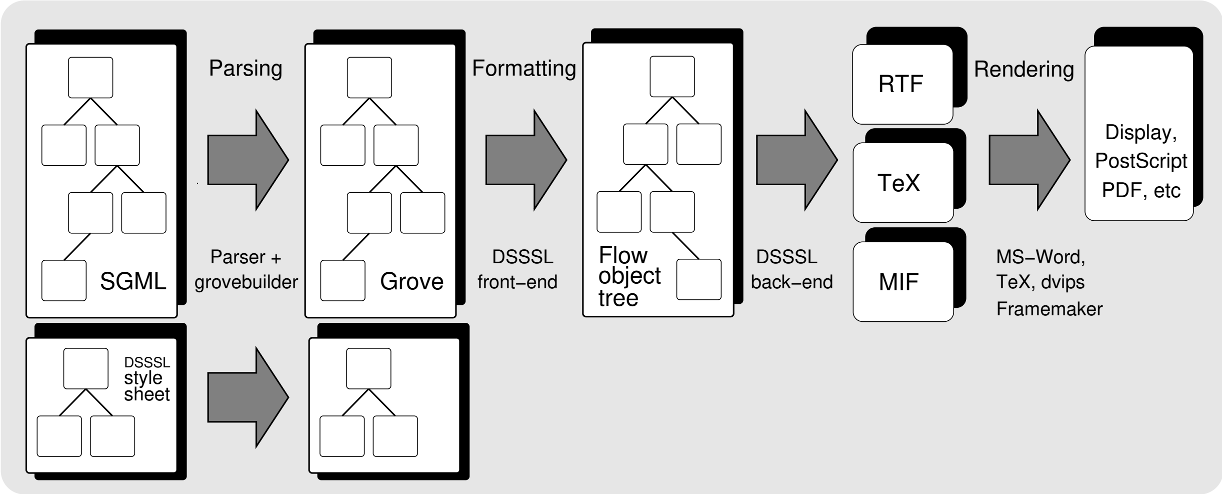

Figure 2.5: The DSSSL formatting process.

and validation of the source document(s), which is converted to a formally defined data structure (a set of trees) called the grove. Note that the use of groves makes DSSSL, at least in theory, even independent of SGML as its input format: other data structures that can be represented as a grove can also be subjected to DSSSL processing. The grove provides the basic model underlying the API between the parser and the DSSSL engine, and style sheets always operate on a document via the grove, never directly on the document. A more detailed discussion of the grove mechanisms is beyond the scope of this thesis, but an important feature of the grove model is that many aspects of a grove (for instance the level of detail, the inclusion of comments or DTD-related data, etc) can be manipulated by specifying a so called *grove plan*. This feature provides extra flexibility for the application (whose style sheets operate on a grove which contains only the information needed), but also involves extra complexity. This is one of the major reasons why an alternative, more simple model, was developed for the Web: the Document Object Model (DOM, discussed in Chapter 3).

A typical DSSSL style sheet is a declarative, platform independent program which specifies the mapping from nodes in the grove to objects in the flow object tree. It is up to a back-end application to realize the layout specified by the flow object tree in a specific target format. The back-end also needs to implement specific typesetting algorithms such as justification, line breaking and page breaking, which are not covered by the DSSSL standard. A DSSSL back-end may choose to delegate the actual implementation of these typesetting routines to other applications (e.g. by converting to RTF or TeX), which can then be used to implement the final processing steps. The total DSSSL formatting process is depicted in Figure 2.5. Note that while DSSSL expressions are expressed using the language Scheme, these expressions are wrapped in an SGML document. This explains why, in Figure 2.5, the DSSSL style sheet first needs parsing by the SGML parser, before its Scheme expressions can be parsed by the DSSSL front-end.

Discussion

The major significance of DSSSL is that it was the first standardized, platform-independent style language for SGML documents. In addition, the lessons learned from DSSSL have had a large impact on the developement of XSL and XSLT, two important recommendations for formatting and transforming XML documents on the Web.

A major part of DSSSL's formatting language has been implemented in a widely used open source application based on James Clark's DSSSL engine [57]. Back-ends are available for converting the flow object tree to SGML, XML, HTML, RTF, TeX and MIF. But DSSSL also has some major drawbacks [225]. First, since the flow object tree models a page representation of the document, DSSSL is only suited for page-based destination formats. In general, multimedia documents can not be represented by DSSSL's flow-based, two dimensional output model, because it cannot be used to model temporal alignment. Conversion to other (semi) structured encodings (such as LaTeX) is also not supported. While DSSSL's set of flow objects can be extended for a particular application, this would require non-trivial extensions of the back-end. Additionally, documents using such extensions can no longer be processed on other standard DSSSL systems. Another drawback of DSSSL is the Scheme-based syntax, and the side-effect free, functional programming style that needs to be used to write DSSSL style sheets. This requires specific programming skills few style authors are willing to master.

2.2.5 Summary

In this section, we took a closer look at the single source, multiple delivery publishing model as it is used in text-based applications. In particular, we discussed the refined version of the model as depicted in Figure 2.3 on page 21. In addition to the distinctions between content, structure and presentation, the refined model also makes a distinction between the appearance of the presentation and how that appearance is to be realized within the context of a given target output. We discriminated domain-independent and domain-specific models and discussed the consequences these differences have for processing, especially formatting, structured documents. Finally, we discussed the two primary international standards that support the multiple delivery publishing model for text: SGML and DSSSL. Both standards will play a dominant role in the remainder of this thesis, for two reasons:

- First, most of the experience in electronic publishing with the single source, multiple delivery publishing model is obtained in SGML or SGML-like environments. Most of the available tools that support this model are also based on SGML. When developing similar tools for hypermedia applications, we choose to take the models and tools developed for text as a starting point, rather than beginning from scratch, and explore if and how these models and tools need to be adapted to support hypermedia functionality. Many examples of document fragments in this thesis use SGML-based syntax.

- Second, SGML forms the basis of virtually all the important hypermedia document formats on the Web (discussed in Chapter 3). Understanding the concepts underlying SGML is a prerequisite for an analysis of the advantages and drawbacks of Web formats such as HTML and XML, as well as the many languages derived from and related to XML. DSSSL has had a major influence on the forthcoming style sheet languages for the Web.

In the following section, we take a closer look at hypertext documents. We discuss the fundamental differences between the models underlying linear and hypertext systems and how these differences affect application of the multiple delivery publishing model to hypertext documents.

2.3 Hypertext Documents

Traditional text documents share a conceptually linear organization of information. For a growing number of applications, this is not sufficient. *Hypertext* documents are characterized by a non-linear organization of textual components, which are related to one another by computer supported links. In this section we discuss the main hypertext research issues in the context of a historical overview of some influential hypertext systems. We explain basic hypertext terminology and standards in the context of the Dexter Hypertext Reference Model. Finally, we discuss the main differences between the models and tools developed for hypertext documents and those for structured text.

2.3.1 Research issues

Hypertext research is, from a historical perspective, strongly influenced by the three pioneers of the field: Bush, Engelbart and Nelson.

Vannevar Bush was the first to describe a hypertext system by proposing a mechanization of the scientific literature system in his famous 1945 article "As We May Think" [53]. While being remarkably foresightful by describing a machine for browsing and authoring of textual and graphical material, Bush did not anticipate the power of digital technology. His *memex* machine was a microfilm and photocell-based system, and Bush admitted that it would be a real technological achievement to make his machine practical.

Bush considered his memex machine as a supplement to human memory. In 1963, Douglas Engelbart wrote — influenced by Bush — his article "A Conceptual Framework for the Augmentation of Man's Intellect" [95], and five years later, the ideas in this article where implemented in the *NLS/Augment* system. NLS featured a non-linear textual database, information filters, window-based views on the filtered data and a mouse-like pointing device. Additionally, it introduced the concept of distributed conferencing and editing.

Parallel to the development of NLS, Ted Nelson — who coined the term *hypertext* — wrote eloquent articles about his *Xanadu* system. The ultimate goal of Xanadu was no less than providing a unified literary environment on a global scale, giving online access to the entire world's literary corpus. Xanadu incorporated automatic protection of copyrights, royalty distribution based on micro-payments and version control. Nelson also coined the term *transclusion*, a sophisticated hyperlink technique, used to define several versions of a document in terms of the original version.

Despite the ambitious goals of the projects described so far, hypertext systems have been practically (and commercially) used since the late sixties. The *Hypertext Editing System* (Ted Nelson and Andy van Dam were the main designers), a predecessor of Brown University's *Intermedia* system, was used in the late sixties to produce the documentation for the Apollo mission. In the eighties, Intermedia was effectively used in an educational environment to teach a course on cell biology and one on English literature. Engelbart's Augment/NLS was marketed as a commercial system by McDonnell Douglas. Carnegie-Mellon's *ZOG* system was used as an information management system on a nuclear-powered aircraft carrier and later developed into a commercial system called *KMS* (Knowledge Management System [10]). See Jeff Conklin's hypertext survey [62] for a more extensive, but somewhat outdated, overview of influential hypertext systems. Most, but certainly not all, of the more recent hypertext research has been carried out in the context of the World Wide Web.

Halasz's seven issues

The hypertext systems described above are categorized by Frank Halasz as *first generation hypermedia* systems (mainframe-based systems such as NLS/Augment, Fress[241], ZOG) and *second generation hypermedia* systems (workstation-based systems such as NoteCards, Neptune and Intermedia; featuring support for graphical information and much more advanced user interfaces, and PC-based systems such as Guide [43] and Hyperties [209], which were more limited in scope and functionality than the workstation-based systems).

All these systems are based on the concept that is essential for hypertext: the concept of *machine-supported links* that can be used by the user to navigate from one information node to another, associated node.

In his article "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems" [112], Halasz discusses seven fundamental limitations of the hypertext model underlying NoteCards and other second generation systems. Since the hypertext model of many of today's hypertext systems is similar to that of NoteCards, most of Halasz's seven issues are still relevant. These issues are further explored in our evaluation of the Web in the following chapter. Since Halasz's work also influenced the design of the Dexter model discussed in the next section, we give a short characterization of his seven issues below:

1. **Search and Query** — A hypertext system offers a navigational interface to the information it stores. Halasz claims that this navigational interface is not sufficient, and that a more traditional query-based interface should also be provided.

2. **Composition** — The basic primitives of the typical second generation hypertext systems — nodes and hyperlinks — also prove to be insufficient. Users want to be able to build composite structures by grouping several related nodes and hyperlinks together into one single unit. Halasz notes that composition makes the semantics of hyperlink traversal less obvious: does activation of a hyperlink to an anchor in a child of a composite imply that only the child (e.g. the paragraph containing the anchor) should be presented to the user? Or does the composite need to be displayed (e.g. the section containing the paragraph)? What if the composite is a deeper nested structure, and is itself the child of another composite? The answer to these questions is application-specific, and in particular cases an author or user needs to be able to control the various options. This requires a richer hyperlink concept, but Halasz does not suggest what such an extended hyperlink should look like. In Chapter 5 we revisit this problem when we discuss the notion of hyperlink context.

3. **Virtual Structures** — Halasz criticizes the static nature of hypertext documents and advocates the use of virtual structures to deal with rapidly changing information within a hyperbase. Virtual structures are a means for overcoming the limitations of extensional defined components that do not change unless explicitly edited by the author.

4. **Computation** — The techniques described so far allow for structures whose contents are determined at access time instead of authoring time. However, Halasz discriminates another case of dynamic behavior: the dynamics in behavior that arises during the presentation by supporting computation in and over the hyperbase. Examples include hypertext documents containing executable scripts, and systems containing computational engines, consuming information from and adding new information to the hyperbase.

5. **Versioning** — According to Halasz, the range of application domains of hypertext systems can be significantly extended by introducing a rich versioning mechanism into the hypertext system. Such a mechanism would not only allow users to store multiple versions of a node, hyperlink or composite, but also to link to a specific version, to the most recent version or even link to the differences (deltas) between successive versions. Multiple versions and their deltas are candidates for the result of a search or can be the basis for the virtual structures described above.

6. **Computer Supported Collaborative Work** — The early hypertext pioneers envisioned hypertext as a natural medium for supporting collaborative work. Still, most second generation hypertext systems feature only limited support for CSCW. Most of them are either designed as a single user system or focus on browsing rather than authoring.

7. **Extensibility and Tailorability** — Halasz notes that the generic nature is both a blessing and a curse. Users can use the primitives offered by a hypertext sys-

tem for any application they consider suitable, and can impose domain specific semantics on these primitives at will. However, the generic hypertext system cannot operate directly on these semantics, which makes the system often less suitable for a specific task than a special purpose application. To overcome this limitation, Halasz advocates systems that are designed to be extensible and tailorable by the average user by making use of a programmer's interface or the user interface itself.

Halasz's seven issues article ranks amongst the most frequently cited papers in the hypertext literature, and has significantly influenced the hypertext community's main reference model, the Dexter model discussed in Section 2.3.2 below.

Engelbart's essential elements of an open hypermedia system

Open Hypermedia Systems (OHS)[4] research addresses the problems related to designing architectures that offer hypertext services to a wide variety of (existing) applications. The systems are "open" to the extent that they are able to offer hypertext functionality to third-party applications without modifying the document models and formats of these applications [177]. An OHS can manage external hyperlinks from and into (read-only) data, a feature that is not only important for adding hyperlink functionality to document formats that are not "link-aware", but also for hypertext systems that support personal annotations and various types of computer supported collaborative work.

Issues related to the software architecture of open hypermedia systems are discussed in part III of this thesis. Here, we discuss the requirements listed in another frequently cited paper: Doug Engelbart's "Knowledge-Domain Interoperability and an Open Hyperdocument System" [96]. The article lists twelve "essential elements of an open hypermedia system", and is based on Engelbart's experiences with the Augment system and his research directed at knowledge intensive work in large organizations. Below, we categorized these requirements into four groups, addressing issues regarding general document structure, basic hyperlink functionality, off-line hyperlink traversal and computer supported collaborative work.

- **General document structure** — These first three elements are not directly hypertext related but define some general requirements on the document structure. Note the emphasis on ease of use when dealing with complex, structured and mixed-media documents:

    1. **Mixed-Object Documents** — documents that contain objects of different media types can be manipulated by the user as a single entity.
    2. **Explicitly Structured Documents** — the objects in a document may be arranged in an explicit hierarchical structure, all subtrees in this hierarchy should be addressable for direct access and hyperlinking purposes.

[4]Note that while most open "hypermedia" systems allow hyperlinks from and to multimedia data, we classify them here as *hypertext* systems since they do not typically provide explicit support for multimedia synchronization.

3. **View Control Of Objects' Form, Sequence And Content** — the view of a document may be controlled, viewing options to deal with the complexity of the documents are available. Examples of such options may include outline views and filtering. The structure and content of the document is directly editable via its view.

- **Basic link functionality** — The following three requirements all relate directly to hyperlinking, and suggest an environment where linking is ubiquitous: available for all documents on every appropriate level of granularity. Note that this implies the use of external linking to be able to link into and from media formats that are not "link aware". Also note that all hyperlink information is supposed to be bi-directional by nature:

4. **The Basic "Hyperdocument"** — documents may contain machine-supported links, to and from objects, within and outside the document.
5. **Hyperdocument "Back-Link" Capability** — when reading a document, a user can get information about hyperlinks from other documents that point to the current document.
6. **Every Object Addressable** — every object within a document is addressable for linking purposes.

- **Off-line link traversal** — These requirements stress the crucial role of hyperlinks within the organization: hyperlink information is too important to be unavailable for users that work off-line on a hard-copy version of a document:

7. **Link Addresses That Are Readable and Interpretable by Humans** — all hyperlink destinations can be made human-readable so that they can be followed "by hand".
8. **Hard-Copy Print Options to Show Addresses of Objects and Address Specification of Links** — hyperlink destinations are also readable in hard-copy versions of the document.

- **Computer Supported Collaborative Work** The remaining four requirements emphasize the use of the OHS as a tool for collaboration among the members of an organization:

9. **The Hyperdocument "Library System"** — documents may be submitted to a library-like service that provides catalog services and guarantees long term access. Documents within the library may link to previously submitted documents, back-link functionality is available for all documents.
10. **Hyperdocument Mail** — all hyperlink services should also be available in the user's mail system.

11. **Personal Signature Encryption** — if appropriate, a user may sign (part of) a document by using a personal signature that can be used to verify the authenticity of the document.
12. **Access Control** — documents in personal, group, and library files can have access restrictions on each level in the hierarchy.

Note that Engelbart's requirements partly overlap with Halasz's seven issues: both emphasize the use of hypertext as a tool in computer supported collaborative work. We will revisit Halasz's seven issues and Engelbart's twelve OHS elements in our evaluation of the Web in Chapter 3.

Other hypertext research issues

Another important item on the hypertext research agenda is the improvement of the user's interaction with a hypertext, with a strong focus on addressing the infamous "lost in hyperspace" problem. To analyze, compare and improve the hyperlink structure of hypertexts, many metrics and structuring guidelines have been developed [34]. Much hypertext literature is devoted to improvements to interfaces supporting hypertext navigation. Especially larger hypertext systems require sophisticated interfaces to prevent disorientation. Several tools to visualize hyperlink structures by means of site maps or concept maps have been developed, including hyperbolic ("fish-eye view"), three-dimensional and other visualization techniques [83, 175]. The spatial arrangement of information in these interfaces has even inspired the development of systems where the traditional hyperlink has been completely replaced by spatial relations. Even without hyperlinks, the interface of these systems turned out to retain a "hypertext feel", and are thus known as *spatial hypertext* [164, 208].

From the early nineties, hypertext became more widely applied, and part of the hypertext research moved its focus from the fundamental models and tools (needed to develop hypertext systems) to the models and tools needed to develop individual hypertext applications on top of these systems. Several methodologies have been developed to facilitate hypertext application design, including RMM [141], HDM [106] and OOHDM [207].

2.3.2 Modeling hypertext documents

The hypertext community's main reference model is the *Dexter hypertext reference model* [113] (a more accessible description of the model can be found in a special issue on hypermedia of the Communications of the ACM, see [114]). The Dexter model attempts to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards. The model is the result of various meetings — the first one was in 1988 at the Dexter Inn — of experienced hypermedia system designers. The model has been formalized using the Z specification language. This Z specification [113] gives an axiomatic description of the model and will be discussed in detail in part II.

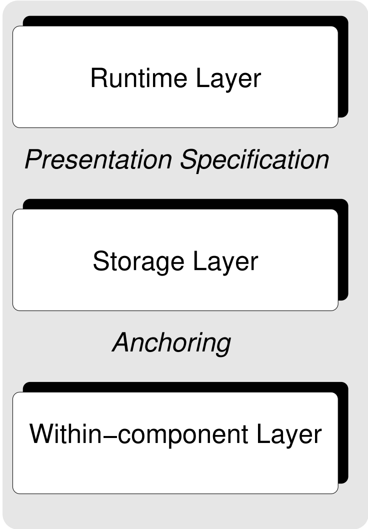

Figure 2.6: The three layers of the Dexter model.

The Dexter model is, as depicted in Figure 2.6, divided into three layers. This division reflects in many ways the research issues discussed above. First of all, Dexter primarily models the link relationships among portions of the data stored in the system, it does not model the data being linked. The first layer of the model, the *within-component layer*, is introduced to isolate the other layers from all data and media-specific details. The elaboration of the within-component layer is considered beyond the scope of the model. The two remaining layers can be said to represent two alternative views on what the important characteristics of a hypertext system are.

The first view focuses on the underlying data structures used to model hyperlinks, and is reflected in the middle layer of the model, the *storage layer*. This layer describes a hypertext system's networked data structure of nodes and links. Note that in the remainder of the thesis, we follow the Dexter model and use the more neutral term *component* for all variations of nodes, including links. The Dexter storage layer also provides a good basis for describing the document model of many hypertext systems.

The second view stresses the importance of the unique, associative user interface and behavior of a hypertext system. This view is reflected in the third layer of the model, the *runtime layer*. This layer provides a more process-oriented view on the system, modeling the behavior of the various operations supported by the hypertext's user interface.

The details of the model will be discussed in Part II, below we discuss some Dexter terminology that will be useful in the following sections: anchoring, presentation specification, link externalization and querying.

Anchoring An important contribution of the Dexter model is the notion of *anchoring*. This concept is introduced to describe the main interface mechanism between the within-component layer and the storage layer. Anchoring is used to be able to address locations or items within components, without knowledge of the inner structure of the data that is encapsulated by the component. It allows linking into components while keeping the link data structures (as defined in the storage layer) independent of the encoding of that component (as defined in the within-component layer). This clear distinction between linking and anchoring was not common in the hypertext systems of the eighties. Note that the difference between a link and an anchor is still rather fuzzy on the Web, since the two concepts are merged into a single HTML element.

Presentation specification The interface between the storage layer and the runtime layer is accomplished using the notion of *presentation specifications*. Presentation specifications are a mechanism by which information about how a component is to be presented to the user can be coded into the storage layer. The way a component is presented can depend on properties of the component itself as well as on the specific runtime layer doing the presentation. One could argue that, within the Dexter model, presentation specifications play a role similar to the role of style sheets in structured text. Note again that the developers of the Dexter model were more concerned with modeling the data structures and interfaces associated with the relationships among the information than with the modeling of the (presentation) of this information itself: while presentation specifications are an inherent part of all components, Dexter does not define the presentation specifications themselves.

Link externalization A Dexter link is a so-called *first class citizen*: it is modeled as an individual component, independent of the components that contain the underlying data or composition information. Links form a structure that is superimposed on, and independent of, the inner structure of the underlying data. Note that this is different from the concept of cross-references and other links found in HTML and other structured text documents, which are modeled as an integral part of the structure of the document itself. On an implementation level, the latter are encoded using embedded markup, and are as such also physically an integral part of the document. In Dexter-based systems, links are typically stored separately from the data being linked. The distinction between external and embedded markup is, especially in the context of hyperlinks, a frequently recurring issue in the hypertext literature [42, 68]. Support for external links is generally regarded as a requirement for supporting personal annotations to material to which the user has no write access. External links also simplify the specification of more complex forms of linking, including the n-ary and bi-directional links supported by the Dexter model. Note that Dexter models anchors, unlike links, as part of the component. Many of the advantages of external links also apply to external anchors, and many (open) hypertext systems also support external anchoring.

Querying Dexter components are always indirectly addressed, by means of a *component specification*. This specification is mapped onto the component's unique identifier, which is in turn mapped onto the component itself. This indirection is used to model queries and is primarily used for a more flexible specification of the component in the endpoints of the link.

Evaluation

The first two of Halasz's seven issues are clearly reflected in the Dexter model. Navigation-based and search/query-based interaction (issue 1), and composition (issue 2) are explicitly supported. One could argue that extensibility and tailorability (issue 7) are implicitly supported by not restricting the data structures of the within-component layer, the presentation specifications and the attribute/value pairs. Relatively small extensions to the Dexter model supporting virtual components and computation have been described in the hypertext literature [108]. The Dexter model fails, however, to address issues related to:

- **Versioning** — In addition to the arguments provided by Halasz (issue 5), versioning has been an important aspect of Nelson's Xanadu system. Being able to build new versions of a document from other versions is one of the basic ideas underlying Xanadu's transclusion mechanism. Since Dexter does not address versioning issues, it cannot model transclusions. Note that Dexter also lacks explicit support for more simplified forms of transclusion such as *stretchtext* where, upon traversal, the destination is integrated into the current document. Such links were, for example, one of the characteristic features of the Guide system [41].

- **CSCW** — While computer supported collaborative work (issue 6) was an essential aspect of hypertext systems as envisioned by Engelbart, Dexter does no attempt to model CSCW issues. As said before, most second generation systems were implemented as single user systems. Even on the Web, where many users have access to documents over the network, support for collaborative work is very limited. Issues related to CSCW that are also not addressed by Dexter include security and support for dangling hyperlinks, which are essential in open systems such as the Web.

- **Multimedia** — While Dexter does not explicitly restrict the use of multimedia data types inside its components, it does not support the synchronization and scheduling of these components. In Section 2.5 on hypermedia, we discuss the Amsterdam Hypermedia Model, which defines extensions to the Dexter model to address these and other multimedia related issues.

Despite these limitations, the Dexter model still provides a relatively simple hypertext model which is sufficiently expressive to describe the model underlying most well-known hypertext systems of the late eighties, and also provides a good basis for a model describing more recent systems. Its distinction between links and anchors provides a useful means to define links independently of the encoding of the media items they link into. Apart from the ability to model external links, it has many other powerful linking features such as n-ary links and generic composition, features which are still rarely supported by the systems that are common today.

2.3.3 Relationship with other document models

The hypertext research issues and models described above are very different from those discussed in the previous section on structured text. From a process oriented perspective, the main differences relate to the more interactive behavior of on-line hypertext systems, when compared to the more batch-oriented models we have discussed for text. The latter are geared towards the efficient production of large volumes of high-quality print, focusing on the production and publishing process. Hypertext systems focus more on the interaction with the end-user, who often has a dual author/reader role. In line with Engelbart's approach, many hypertext systems facilitate the end-user performing knowledge intensive tasks, where the system supports the user in making the interrelationships among different pieces of information explicit.

Another difference is that both markup and style sheets, as used for structured text documents, usually apply to the entire document, which has advantages both in terms of simplicity and consistency. In hypertext, the notion of the "entire document" often does not exist, as documents are often made of many interlinked but individual components. Components originally authored for one document can be linked to another document, and in such situations striving for a consistency in markup terminology or layout may not be appropriate. On the other hand, systems based on stretchtext links (such as Guide) require full integration of textual components within the current document. In these systems, a link destination is displayed inline with the current component, replacing its source anchor. This requires dealing with potential conflicts between different markup models or inconsistent presentation specifications. Guide used separate namespaces to avoid conflicting markup when integrating components [41]. The problem of combining layouts was partially solved by Guide's inheritance mechanism for presentation properties. This mechanism helped in achieving a consistent layout because the inheritance tree was based on the strict hierarchical structure of the stretchtext presentation, rather than on the underlying graph of the hyperlink structure.

Note that hypertext systems that explicitly address issues related to structured markup and multiple delivery publishing, such as Guide, are an exception. Most hypertext systems use either a single browser interface to present and manipulate all information, or are built independently of the applications that are responsible for the final presentation. Accordingly, multiple delivery publishing has generally not been regarded as an important issue in hypertext research. Only a few of the issues we discussed for structured text have been applied to hypertext. Examples include presentation-independent linking, adaptive hypertext, platform independence and interoperability.

Presentation-independent linking

The power of hypertext lies in the generally applicable concept of the link. Links have been successfully used to describe rich semantic relations between information, but also to describe navigational interfaces. When the difference between these two link types is important, we use the term "link" to refer to the first, generic, type of link, and use the term "hyperlink" to refer to a link that specifically relates to navigation.

Most hypertext applications, however, either ignore the differences between these two links, or implicitly assume a simple one-to-one mapping: semantic links in the document are directly mapped to navigational hyperlinks in the presentation. Different presentations of a document, however, often require different navigational interfaces. For such applications, a direct mapping from a semantic link to a specific type of navigational hyperlink is too rigid.

A more flexible approach, that also fits better in the multiple delivery publishing model, separates these two link types. For usage on the document level, links are provided that are especially developed to make semantic relations within and outside the document's content explicit. These links are often typed [219] to indicate the nature of the relationship, provide additional information on the semantic role each anchor plays in the relationship, indicate the semantic direction of the relationship, etc. Depending on the individual presentation, semantic links may or may not need to be mapped to navigational hyperlinks during the generation of the presentation. In addition to the hyperlinks that were derived from links that were explicitly encoded in the document, one may want to dynamically add other hyperlinks to adapt the navigational interface to specific circumstances.

For each navigational hyperlink, various types of stylistic information need to be provided. Examples include the way to visually indicate the presence of a link marker, presentation information needed to visualize the hyperlink in a site map, etc. Additionally, the mapping should define the hyperlink's traversal behavior. This includes the traversal direction of the hyperlink and the effect that traversal has on the presentation of the source and destination of the hyperlink. This information is often implicit, or built into the hypertext application. Most Web browsers, for example, offer the user a choice of whether the destination should be presented in a new window, or in the current window, replacing the current document.

As stated above, only few hypertext systems discriminate between semantic links and navigational hyperlinks. As a result, presentation-oriented information is mixed with the semantic link information, and both link types are embedded in the document. While this increases the control the author has over the presentation, it reduces the chance the same information can be used in another context. Therefore, a more flexible approach would allow specification of information regarding hyperlink style and traversal behavior separately from the main document, for example in a style sheet. The advantages and drawbacks of this approach are similar to those related to the external specification of other types of presentation information, as discussed in the introduction of this chapter (see page 15.

This approach to linking, based on the structured document approach, requires support for linking on many, if not all, levels of the multiple delivery publishing model. On the document structure level, links need to be supported to describe semantic relations. On the presentation level, hyperlinks are needed to describe the navigational interface. The style sheet language needs to support an adequate mapping between those two levels [227]. Finally, the application's target output format needs to be able to realize this by means of an appropriate user interface.

Note that in some models, including the Dexter model, the content is modeled as a black box, that is, the content is modeled as being a raw block of data without inner (link) structure. The role of the hypertext system is to add a hyperlink layer to a collection of data which is by definition deprived of hyperlinks. More and more media types, however, have built-in support for hyperlinking. To handle such media types as content, hypertext systems need to be able to combine (possibly conflicting) hyperlinks from different sources, defined both on the content and structure level. So an adequate model for this type of link integration requires link support at all levels, even at the content level.

Adaptive hypertext

Adaptive hypertext recognizes the fact that different users might benefit from different navigational interfaces. In adaptive hypertext the goal is in many ways similar to that in multiple delivery publishing: (semi)automatic generation of different hypertext presentations from a single source hypertext. A difference is that the focus is not on supporting different output media or different layouts, but on supporting different types of users. Extensive research, especially on the use of adaptive hypertext in an educational environment, has explored different techniques to adapt both the content and hyperlink structure in order to generate presentations that are tailored to individual users from a single source hypertext [44]. Adaptive hypertexts typically maintain a user model, and an explicit model that determines how the hypertext needs to adapt to changes in the user model. A description of an adaptive hypertext model in terms of the Dexter reference model is given in [72].

Platform independence and interoperability

Other research issues we have discussed for structured text include platform independence and interoperability. These topics received relatively little attention during the development of most hypertext systems in the eighties. The majority of the second generation systems described above ran on a single workstation or PC, and was not connected to other hypertexts over a network. Hence, there was hardly any need for standardization of hypertext document formats. Interoperability has more recently become relevant in the context of the Web (discussed in Chapter 3), and in the context of open hypermedia systems (OHS) research discussed above.

Since the primary goal of an OHS is to extend the user's existing (desktop) applications with hypertext functionality, an OHS cannot rely on embedded markup techniques to store the necessary information into the target documents. This would, in most cases, break the existing application. Additionally, it would prevent users from adding links to documents to which they do not have write access.

In an OHS, all information related to links and anchors is stored separately from the documents themselves. This type of information is typically maintained by a middleware component called a *link server*. Client applications can query the link server to retrieve link information relevant to the documents they are currently dealing with. Interoperability focuses on standardization of the protocol between client application and link server, and not, as in the case of the structured text, on standardized document formats. In fact, since most standard markup languages (including HTML) use embedded markup to store link related information, these techniques are not suitable as the basis for an OHS.

Traditionally, OHSs only supported navigation-oriented links. But recently, the same techniques have also been applied to other, more domain-specific structures. OHSs have been used to deal with spatial hypertext structures, to manage taxonomic link structures in applications that focus on information classification, and to process workflow relations in workflow management systems [236]. From this perspective, one can say that these systems go a step further than structured documents. Structured documents only externalize the presentation-related information, embedding other structural information in the document. In contrast, the OHS approach essentially promotes separate storage, encoding and processing of all important structural information.

2.3.4 Standardization

While one of the goals of the Dexter model was to provide a basis for standardization, a Dexter-based hypertext document standard has never been developed. Dexter is used, however, within the OHS community as the basis for the development of a standard network protocol. This protocol describes the communication between the client applications and its link servers (more in Chapter 7). The Dexter model has also influenced the forthcoming linking specification for the Web (XLink), to be discussed in Chapter 3.

HTML and the other protocols used on the Web, are viewed by many as today's *de facto* hypertext standards. These protocols are also discussed in Chapter 3. Within the SGML community, the Text Encoding Initiative (TEI) has extensively added to SGML's (hyper)link facilities. The TEI project has gained much experience in marking up complex electronic documents in the humanities. Many of the ideas for defining anchors in the TEI are used for the XPointer specification described in Chapter 3.

ISO standardization efforts for hypertext have culminated in the HyTime [134] standard. HyTime provides many hypermedia related extensions to SGML, of which anchoring and linking are only two examples. To support anchoring, HyTime provides the *locator* concept. Locators can be used to point to arbitrary portions of SGML and non-SGML documents, and come in many different flavors. Once pieces of information have been located, they can be linked using one of HyTime's linking constructs. Since HyTime supports n-ary and bidirectional linking, and supports both embedded and external encoding of link information, it can very well be used to encode Dexter-like hyperlinks. The main difference between linking in HyTime and Dexter is that HyTime links are presentation independent, while Dexter hyperlinks contain explicit presentation specifications. HyTime linking constructs are geared towards encoding links in documents that employ structured SGML markup. Hyperlinks in final form representations need more presentation specific information and are generally beyond HyTime's scope. HyTime linking forms the basis of the recent ISO topic maps standard [139]. Additionally, many HyTime concepts are used in the XLink specification for the Web, though in a much simpler form.

2.3.5 Summary

In this section, we discussed the main hypertext research issues and provided a comparison between research related to hypertext and structured text. Structured text models provide generic facilities for the specification of content, structure, presentation and realization. Hypertext models, including the Dexter model discussed in this section, provide elaborate ways of addressing parts of a document (anchoring) and specification of relationships among these parts (linking). Links are used for encoding semantic and structural relationships and for describing the document's navigation interface. The non-linear, link-based structure of hypertext documents provides an alternative to the linear structure implied by the lexical flow of the more traditional structured text documents.

Hypertext documents are traditionally geared towards a single user interface (e.g. a browser), or use the native document format of a particular desktop application in combination with external storage mechanisms to store link-related information. Only a few systems, such as Guide, have explicitly addressed issues related to multiple delivery publishing.

However, in more and more hypertext applications — especially on the Web — there is currently a need to be able to deliver alternative versions of the same hypertext to adapt to the end-user's preferences or platform. This need can be addressed by combining the virtues of both structured text and (adaptive) hypertext. It requires an extension of the multiple delivery publishing model with support for linking on all levels (content, structure, presentation and realization). Such an extended model needs to be able to distinguish between defining the semantics of a relationship (defined by a link on the level of the document structure) and the traversal behavior of the associated navigational interface (defined by the hyperlink in the presentation).

Embedded markup is in many tools for structured text the only supported form of markup. While embedded markup can be used to encode links as part of the main document structure, it is not suited for encoding the wide variety of links that have been in use in many hypertext systems. To enrich structured text with these more advanced link types, a document processing environment also needs to support a non-embedded encoding of links and other structures. Examples of standards supporting non-embedded linking include ISO's HyTime standard and W3C's upcoming XLink specification [75].

Whether links are embedded in the document or not, being able to encode links on the structure level is not sufficient. When a structured hypertext document is converted to a final-form presentation format, structured relationships defined on the document level are typically converted to navigation-oriented, hyperlink objects in the presentation format. This requires that both the conversion mechanism (e.g. the style sheet language) as well as the output format should support hyperlink-based navigation. Examples of style sheet languages supporting hyperlink-based navigation include ISO's DSSSL and W3C's CSS, examples of (final-form) document formats supporting hyperlink-based navigation include Adobe's PDF and W3C's HTML. ISO's Standard Page Description Language (SPDL, ISO/IEC 10180:1995) does not yet support hyperlink-based navigation.

2.4 Multimedia Documents

Due to the large amount of resources needed at the end-user's desktop, the history of digital multimedia systems is strongly tied to that of personal computing. The computational power and other resources needed to run multimedia applications became only recently available to the general public. Multimedia requires fast (co)processors, fast and high-density storage devices (e.g. CD-ROM) and, in a distributed environment, fast networks. In addition to simple linear play-out, multimedia information systems should support other temporal operations, including reverse play-out, slow-motion, fast forward, fast backward and random access. Although these features are all very common in existing (analogue) technologies as VCRs, in a multimedia information system they are often hard to implement due to non-sequential storage, data compression and distribution or random network delays [159].

2.4.1 Research issues

Until the early nineties, multimedia document models and authoring and presentation systems have received scant treatment in the literature. Nowadays, a huge number of conferences, books and journals are explicitly devoted to multimedia related topics.

Typical multimedia research issues include audio, still image and video compression algorithms, mega, giga and even terabit network architectures, and a diverse range of multimedia applications (such as video conferencing, video-on-demand, home electronic catalog ordering, virtual class rooms, multimedia mail, computer supported collaborative work, etc). As in the previous sections, we focus our discussion on the more document-related research issues.

Besides the amount of resources needed, multimedia documents differ conceptually from text and hypertext documents because of the temporal dimension of the continuous (or time-based) data that constitutes a multimedia document. Continuous data values can be considered as composed of a sequence of events, linearly ordered in time. For example, an audio fragment is essentially a sequence of samples, and a video clip consists of a sequence of frames, both ordered in time. In contrast, text, graphics and still images have only spatial dimensions and these media types are often referred to as discrete media types. This temporal dimension plays an important role in virtually all multimedia related research.

Below, we discuss the main aspects that differentiate multimedia documents from text and hypertext documents. These aspects include document-level temporal synchronization and quality of service, and multimedia spatial layout.

Temporal synchronization

A multimedia document combines a number of relatively independent media items into a single, integrated, presentation[5]. The document needs to specify the relative timing constraints of its media items, i.e. the constraints that specify the *intermedia synchronization* (as opposed to *intramedia synchronization*, which concerns the synchronization of the events within a single data stream). Many different temporal models for defining intermedia synchronization have been described in the multimedia literature. The temporal model is sometimes hidden from the author, but in many multimedia authoring systems the paradigm of the authoring interface is directly based on the underlying temporal model.

Current multimedia PCs and workstations are quite successful in presenting synchronized multimedia, providing the sheer bulk of data is stored locally. In such an ideal play-out environment, the differences between the proposed temporal models are minimal. Networked multimedia applications with distributed data streams, however, require more flexible synchronization and other advanced data processing. In these environments, with limited resources, the runtime system needs information to adapt the presentation of the document to the (changing) environment. In networked multimedia, unsynchronized play-out is often due to the limitations of the document's underlying temporal model that assumes an ideal play-out environment. Below, we evaluate several temporal models on their ability to deal with such non-ideal play-out environments.

Quality of service and content adaptation

Current operating systems and networks typically process data "as fast as possible". In the case of multimedia, it is likely that the data is processed either too fast or too slow. The first case can lead to buffering problems, and may very well result in wasting the resources needed by another (multimedia) application. In the second case, the application may not be able to correctly present its time-sensitive information. A "best-effort" strategy is not sufficient, since most multimedia applications require hard or deterministic guarantees about the resources that are available.

Quality of Service (QoS) management addresses both problems. Applications request the needed QoS for a given resource, and a resource manager may grant the request, deny it, or advise lower service level parameters. In the last two cases, an application may decide to try again later or accept the lower service level. In the first case, the application can expect the requested service level, which should be guaranteed by the resource manager for a known period of time. QoS management can be done on several levels, resulting in hardware, network, operating system, and software level solutions. In this thesis we focus on the document-level issues, and consider system-level

[5]See [123] for a more elaborate overview of the role of temporal relations at different abstraction levels

QoS beyond our scope (see [50] for more information on system-level QoS).

On the document level, QoS issues typically become apparent in facilities supporting graceful degradation of the presentation in case the available resources are not sufficient for optimal play-out. For the multimedia author, this typically involves providing alternatives for media items that are unlikely to be available on all target platforms. For example, an author may provide both a high and a low resolution version of a video fragment, together with a text description of the video, to guarantee appropriate playback on high and low bandwidth systems and on systems that do not support video at all. Providing alternative content usually does not suffice. A multimedia document needs to contain sufficient information on which the playback environment can base the QoS negotiation process and select the content that provides the best playback given the available resources. In the example above, the document may need to provide detailed information about the network bandwidth and video decoders needed to render the video fragments, so that the end-user's system can make an informed choice when selecting between the two versions of the video and the text description.

Multimedia layout

A multimedia author needs to decide, not only *when* the media components should be played in the presentation, but also *where*, on which region of the screen. Modeling the spatial layout for multimedia presentations is rather different than that for text.

A first difference is that where text-based layout models are mainly geared towards rendering text-flows to multiple pages or scrollable view ports, multimedia models deal mostly with a single, non-scrollable screen area.

Another difference is based on the fact that the rendered spatial position of a word is highly dependent on its position in the text-flow. The word's spatial position needs hardly ever to be specified explicitly: the lexical order of the content of text-based documents provides — when combined with a relatively small number of style rules — sufficient information to derive the spatial layout of each textual element. In fact, explicit specification of an element's spatial position is in most cases highly unwanted, because it prevents a "re-flow" of the text, i.e. an automatic recalculation of the layout when new material is inserted or, in the case of online display, a resize operation of the view port.

In contrast, the spatial layout of media items in a multimedia presentation cannot be derived from the lexical order in which these items appear in the document, nor can it be derived from the temporal order in which these items appear in the presentation. For most media components, the spatial layout needs to be specified explicitly. When new material is added, the other media items often need to re-flow in time, and their spatial layout is not necessarily affected. When resizing the view port, the effect on a media item's spatial layout may vary, even within a single presentation. Some media items may need to be scaled proportionally, others may need to preserve aspect ratios, text items may need to re-flow, while the layout of yet other items may remain unaffected. An author should be able to specify such behavior when defining the layout of

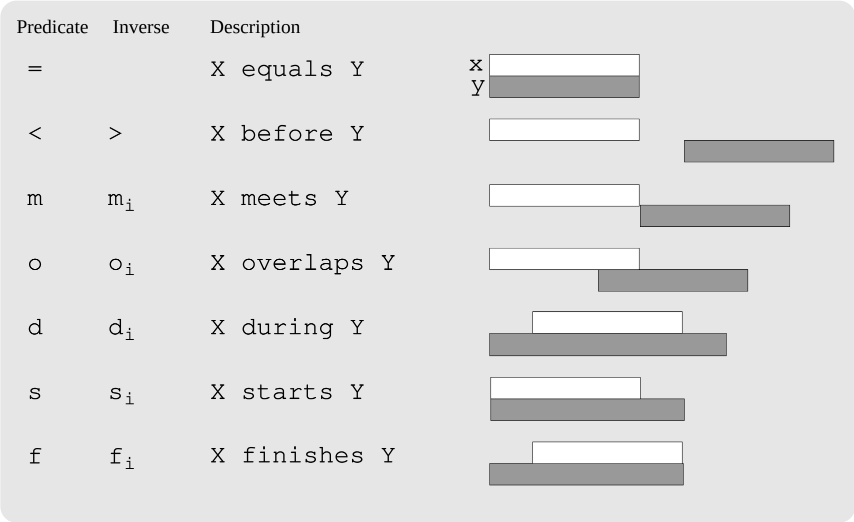

Figure 2.7: Allen's 13 relationships.

his multimedia presentation.

The temporal ordering of the media items often reflects the overall narrative, which should not be changed by a style sheet. The same applies, to a certain extent, to the document's spatial layout. When multiple images and associated text descriptions are positioned on a screen, people use the spatial layout as the main clue for determining the associations between the descriptions and the pictures. Therefore, it is often convenient to distinguish these spatial constraints from other, more variable, presentation-oriented information (e.g. style).

2.4.2 Modeling multimedia documents

Different techniques have been developed and described in the literature to specify the temporal behavior of multimedia documents. In "Maintaining knowledge about temporal intervals" [11], Allen introduces an interval-based temporal logic and a reasoning algorithm based on constraint propagation. While neither multimedia nor synchronization issues are mentioned, Allen's article ranks among one of the most cited articles in the multimedia literature, where the article is typically quoted for the thirteen predicates describing possible relationships between temporal intervals. Allen introduces an equality relation and six other relations with their inverse relations (see Figure 2.7) to model all relationships between two temporal intervals. Reasoning over several predicates employs the transitivity properties that hold for most relations, but also for combinations of relations (e.g. $(a = b) \wedge (b < c) \Rightarrow a < c$). Allen introduces reference intervals to guide the computation process and limit the number of indirect relations that need to be calculated. These reference intervals describe *during hierarchies*, hierarchical structures where each child interval occurs (in one way or another) *during* its parent interval. Variants of Allen's during hierarchies are used in some of the temporal models discussed below.

Time-line models

The most basic models are based on an explicit timeline. All start and end times of the media items are specified in the document relative to this timeline. The advantage of these models is that they are simple to implement and provide sufficient timing information to the scheduler in ideal play-out environments. Another advantage is that they provide a good basis for an intuitive authoring paradigm, since authoring can be carried out by positioning icons that represent media items on a timeline (this is one of the paradigms used, for example by commmercial authoring tools such as MacroMedia Director [162]).

A drawback of timeline models is that in networked environments, resources are limited and the scheduler cannot always meet the specified begin or end times. In these cases, pure timeline models do not provide sufficient information upon which an alternative scheduling strategy can be based: the presentation environment can only provide a "best effort" strategy and try to keep the average deviation as small as possible.

Event-based models

A common scenario which causes problems in pure timeline models is a schedule where two videos, A and B, are to be played in sequence. In a timeline model, the begin time of B is specified relative to the timeline. As a consequence, when A is delayed, B is likely to start too early. The obvious solution is to specify the beginning of B relative to the end of A. This type of relative specification is supported by event-based models, where the end of A fires an event that triggers the start of B. The advantage is that this model supports scheduling of items relative to other objects, and not only relative to an absolute timeline. Another advantage is that because it is easy to incorporate new types of events, such as user interaction events, this provides a general and natural model to support all kinds of interaction. By introducing timer events, the event model can easily be combined with the timeline model sketched above.

A drawback of event models is the significant (programming) effort which is required to specify the correct generation, propagation and handling of all events in a multimedia document. However, this effort can be largely reduced by using an adequate authoring tool. A more fundamental problem is that when the document is modified (e.g. media items are moved, added or removed), it is not always clear how the event wiring should be adapted to reflect the changes appropriately.

Structure-based modeling

Structure-based models, including CMIF [52, 228] and SMIL [231], use a hierarchical document structure to automatically generate the bulk of the timing information. The document hierarchy in these models reflects the temporal structure of the presentation. The components in the tree are either atomic or composite. Atomic components contain (or refer to) the actual multimedia data and refer to a specification of the component's spatial position on the screen. Composites typically come in two flavors: parallel and sequential. Note that these parallel/sequential hierarchies are just a special case of Allen's during hierarchies.

In practice, the information provided by the temporal hierarchy is sufficient to derive the most common types of synchronization, but an extra concept is needed to be able to specify all thirteen Allen relations. CMIF introduces the *synchronization arc* as a general temporal relation between two components. Typical attributes of a synchronization arc include the delay between two nodes and acceptable deviations. Instead of synchronization arcs, SMIL allows authors to specify additional synchronization properties in the attributes of its elements. These properties are more strict when compared to CMIF's synchronization arcs (e.g. SMIL 1.0 does not support specification of acceptable deviations). Most constructs in CMIF and SMIL are explicitly modeled by the Amsterdam hypermedia model, and further discussed in Chapter 5.

An advantage of the structure-based approach is that when the document is modified, the hierachical structure isolates modifications to a specific subtree. This allows, for instance, the modification of a document using cut, copy and paste operations on a specific subtree. After the operation, the system can calculate all new timing constraints automatically. Note that the isolation of temporal constraints can be broken by "out-of-scope" synchronization relations which specify relations between the subtree that is modified and the rest of the document. SMIL 1.0 simply forbids these types of relations by limiting the scope of synchronization attributes, and only allows a reference to the direct parent or sibling element. In CMIF, out-of-scope synchronization arcs are allowed, but cut and paste operations on subtrees often require manual editing of the out-of-scope synchronization arcs to restore the correct timing relation.

Constraint-based models

In constraint-based models, temporal constraints are defined explicitly in the document. A constraint solver is used to calculate the information needed by the runtime environment to determine possible begin/end times of the media items. Again, the equivalent of Allen's thirteen relations can be used as the basic set of constraints, which is, for example, the case in the Madeus system [145]. An advantage of constraint models is that they support a higher level of temporal specification than the hierarchical parallel/sequential systems discussed above.

While a set of constraints does not need to be hierarchically organized, it may take advantage of the properties of Allen's during hierarchies. The Madeus system for example, uses the hierarchical structure of its documents to localize the scope of its constraints. This approach simplifies authoring in a way similar to the approach discussed for structured documents.

Additionally, the same constraint system that is used for temporal specification can often be reused for spatial layout. If the constraint solver is part of the runtime playout environment, it also allows the system to adapt to network or other delays by trying to calculate alternative scenarios that still satisfy the constraints specified by the author.

Drawbacks of constraint-based systems include the effort required to train multimedia authors in using a constraint-based paradigm and the extra complexity associated with the constraint solver, both in the playback and authoring environment. Another problem, which applies to many types of non-embedded information, is the authoring effort required to keep the set of constraints up-to-date when the document is modified.

Note that variants of the temporal specification techniques described above are also used for the specification of spatial layout. Examples include absolute positioning using spatial coordinates in a two-dimensional coordinate system, positioning of media items relative to other (groups of) media items, and the use of spatial constraints. Again, such specifications often employ structures similar to Allen's duration hierarchies to deal with the complexity involved.

Formal modeling techniques

Several formal techniques have been used to model the temporal behavior of both multimedia documents and multimedia systems. The most dominant ones are based on temporal logic, process algebra or Petri nets. See Chapter 6 for a more in-depth assessment of formal specification techniques for (distributed) multimedia systems.

Temporal Logic Temporal logics are modal logics where the modal operators are used to express temporal relations. Typical operators include $\bigcirc$ for next, $\diamond$ for eventually, and $\square$ for henceforth. Indeed, the more common temporal logics only deal with qualitative time. Very few temporal logics also support the quantitative time needed for the specification of multimedia documents. Temporal logics have been successfully used to specify and prove various qualitative aspects of the dynamic behavior of multimedia systems.

Process Algebra As is the case with most temporal logics, most common specification techniques based on process algebra also do not support quantitative time. However, Duce et al. [79] used the ISO standard LOTOS for the (partial) specification of the PREMO multimedia presentation environment [215]. Timed extensions to LOTOS come in several variants, most of them based on new delay operators and time stamps attached to events. Blair et al. [28] provide an assessment of several extensions to LOTOS. Furthermore, they separate the specification of the behavior of the system from the specification of the (real-time) requirements.

Petri Nets Petri nets (or better, several extensions to Petri nets) have also been used to model multimedia systems. Merlin's Time Petri nets label each transition with an interval label $[\alpha, \beta]$ to indicate that if a transition is enabled at time τ, it will fire in the interval $[\tau+\alpha, \tau+\beta]$. Other models use timed places [158, 187]) or timed arcs [235]. Some models support composition based on Allen's thirteen relations to build larger Petri nets out of smaller ones. Whereas temporal logics and process algebras have been used primarily for specification and analysis of the behavior of network protocols and real-time systems, Petri nets have also been used for specification of the temporal behavior of multimedia documents.

Video Algebra An example of a technique that has been especially developed for the specification of multimedia, and goes beyond the specification of temporal behavior, includes the video algebra introduced by Weiss et al. [238]. Their algebra can be used to specify composition, retrieval, navigation and play back of digital video. It models hierarchical video structures (shot, scene, sequence), temporal composition of video segments, video annotation and associative access based on content structure or temporal information. Typical algebraic operations include creation of a video fragment from a raw video stream, composition operators (such as union, intersection, concatenation etc), video effects (such as speed modification and transitions), query operations (returning video fragments matching a certain query), output characteristics (placement in the parent window, audio characteristics) and description operations (such as annotation).

2.4.3 Relationship with other document models

Multimedia presentations were, until recently, typically encoded using binary, proprietary and final-form document formats, optimized for a specific target platform and distributed on CD-ROM. Interoperability, reuse, platform independence and other issues have mainly been addressed in the context of (monomedium) file formats. Multiple delivery publishing as advocated by the structured text community, has received relatively little attention. Text-based, structured markup has never been common for encoding multimedia documents.

This started to change when text-based markup became the *de facto* standard for documents on the World Wide Web. Both end-users and content providers have learned how to effectively employ techniques such as style sheets, fill-in forms and scripting for text-based Web pages, and prefer to use a similar set of tools and techniques for Web-based multimedia presentations. While binary and proprietary formats are also used on the Web, formats such as dynamic HTML (a combination of HTML, style sheets and script-based techniques) and SMIL (see Chapter 3) indeed use text-based markup for the encoding of multimedia presentations.

Another consequence of multimedia on the Web is that many of the CD-ROM-based, single platform formats no longer suffice. The Web, with the unpredictable network delays and various browser platforms (ranging from high-end PCs to mobile phones) requires a more flexible specification of multimedia presentations.

An encoding of multimedia presentations based on structured markup may provide the necessary integration of multimedia with current Web formats, while the needed flexibility can be provided by a potential multiple delivery publishing format for multimedia. Generic markup languages such as SGML and XML do not constrain the semantics of the markup, and thus allow the development of adequate markup schemes that explicitly support the specific semantics of multimedia documents.

Unfortunately, multimedia does not fit as easily in many other models associated with structured document processing. An important problem is the strict separation between structure and presentation underlying structured text. For hypertext, we could maintain this separation by discriminating semantic, presentation-independent links from presentation-oriented navigation links. For multimedia, however, we pointed out that some temporal and spatial constraints are essential for the document's narrative and rhetorical structure. From a structured document perspective, this would suggest that those spatio-temporal constraints that are essential from the perspective of the document's semantics are best specified in the source document structure, while other aspects of the temporal and spatial layout are more likely to vary across presentations and thus better specified externally, for example in a style sheet. This essentially replaces the separation between structure and presentation by a (much weaker) separation between constant and varying document properties.

In the previous section we discussed hypertext in terms of a structured document model, and explained why hypertext requires support for linking on all levels of this model. A similar argument applies to multimedia. A structured multimedia document model requires support for explicit specification of temporal information on all four document processing levels. On the content level, this information typically relates to (intramedia) synchronization of the various continuous (monomedia) media items. On the level of the document structure, this involves specification of the relative temporal ordering of these items and other information that is essential for the document's narrative. On the presentation level, the precise intermedia synchronization of the presentation needs to be determined, and finally, this synchronization needs to be realized in the format expected by the runtime environment.

Therefore, simply encoding multimedia presentations by using structured markup provides only a small part of the solution. Extending the models on the other levels is typically much harder. Generic markup languages may be neutral with respect to the presentation and other semantics of the document, this does definitely not apply to the style sheet languages, models and tools used to process these documents. For example, style sheet languages such as CSS and DSSSL do not account for the temporal behavior of a document, and, as such, cannot be readily used for defining the presentation semantics of a multimedia document. Link models often make explicit assumptions about traversal behavior which is geared towards static, text-based documents. Scripting and form-based interaction have also proved to be very effective in a text-based environment, but interfere with the scheduling techniques used in a multimedia environment.

An additional problem is caused by the fact that many of the temporal models that are successfully used in multimedia, cannot be readily used to extend existing text-based applications. For example, a major drawback of models that are based on a temporal document structure is that these models "claim" the document structure for specification of the temporal behavior of the presentation. This makes it hard to combine these models with existing document formats, which already use the document structure for other purposes. Another drawback is that modification of the temporal order in the presentation requires changing the structure of the document, which goes against the principle of separating structure and presentation. This is especially a drawback for applications which need multiple presentations of the same document, in which the temporal order of the media items varies in each presentation. For such applications, constraint models may be a better alternative. In such systems, the temporal information is not implied by the main document structure, and can be stored externally to the document.

But temporal characteristics are not the only reason why multimedia documents differ from text and hypertext. As discussed above, the spatial layout requirements for multimedia differ fundamentally from those for text. Document browsers, for example, typically base their layout on a scrollable view port, a user interface technique that seldom suits multimedia presentations. As such, the specific requirements related to spatial layout form a second multimedia document characteristic that is likely to require a change in many of the existing models for structured text processing.

The third multimedia document characteristic that differentiates multimedia from text is content adaptation: specification of alternative media content to adapt to different play-out environments. Because this involves platform-specific information, it is often desirable to separate such information from the main document structure. In practice, however, replacing one media item by another (e.g. a video by a textual description of that video) might have a significant impact on the overall presentation. Therefore it is usually appropriate to enable authors to control this process and to provide support for content adaptation on the level of source document structure. But again, document level support is not sufficient. These alternatives need to be appropriately translated up to the higher levels of the document processing model in order to allow content negotiation and adequate support on the end user platform.

These three specific document characteristics (temporal synchronization, multimedia spatial layout and alternative content) prevent current models and tools supporting structured text from being readily applicable to multimedia documents.

2.4.4 Standardization

The main multimedia standardization efforts in this area relate to the last level of the multiple delivery publishing model, including final-form document formats, real time network protocols (e.g. the suite of IETF protocols for multimedia networking on top of the Internet Protocol, including RSVP, RTP, RTCP and RTSP [35, 205, 206]) and multimedia presentation environments (e.g. ISO's PREMO [215]). A large number of standards for single media formats (e.g. audio, image, video, etc) exists, and these standards are frequently re-used by standards that define multimedia presentations by specifying synchronized and coordinated playout of multiple single-media items. The main standards in this area are the sets of standards by ISO's multimedia and hypermedia experts group (MHEG) and ISO's motion picture experts group (MPEG). While the latter acronym is primarily associated the standards for audio and video formats (as defined by MPEG-1 and MPEG-2), later versions also include support for other (structured) media types (MPEG-4 [140]), and standardization related to metadata and multimedia information retrieval (MPEG-7). The most well-known standard from the MHEG family is MHEG-5 [137], which specifies an object-oriented model for multimedia applications that run in a minimal-resource environment (primarily TV set-top boxes).

One of the few standards that addresses the structure level of multimedia documents is ISO's HyTime standard [134]. In addition to the hyperlink-related constructs discussed in the previous section, HyTime specifies abstract facilities for encoding temporal and spatial relations in structured documents. HyTime provides abstractions which can be used to position arbitrary information in an n-dimensional coordinate space. How these abstractions are to be presented to the user is up to the multimedia application, and not within the scope of HyTime. At the time of writing, there is no standardized or even commonly accepted method for defining a mapping from structured multimedia documents to a concrete, directly presentable multimedia format. This severely limits the practical value of the HyTime standard.

2.4.5 Summary

With the increase of multimedia support on the Web, text-based markup is now a technique that is commonly used for the encoding of multimedia documents. Additionally, there is a need for encoding these documents in a platform independent and adaptable manner. A multiple delivery publishing model that is suited for multimedia documents could fulfill this need.

In the previous section on hypertext we concluded that support for hypertext linking requires an extension of all four levels of the multiple delivery publishing model. In this section, we showed that multimedia documents require explicit support for spatio-temporal relations. As is the case for hyperlinks, spatio-temporal relations need to be supported on all levels of the multiple delivery publishing model, and require a similar extension of the associated languages and tools.

Most multimedia models and standards, however, focus on the media content and the final presentation of this content. On the intermediate levels, modeling of multimedia concepts is hardly supported. While the HyTime standard is one of the few models that provides support for multimedia modeling on the level of the abstract document structure, there is no standard or commonly accepted way of defining the presentation semantics of a HyTime document.

Note that in our link-based extension of the multiple delivery publishing model it was possible to maintain a strict distinction between structure and presentation. In the multimedia based extension, this distinction is replaced by a less strict distinction between those aspects of the document that are likely to vary (e.g. style aspects) and those that are likely to be constant (e.g. the document structure including basic temporal and spatial constraints that are tightly bound to the document's narrative and rhetorical structure).

2.5 Hypermedia Documents

The word "hypermedia" already suggests a combination of "hypertext" and "multimedia", and hypermedia documents do indeed combine aspects of hypertext and multimedia documents. But a wide range of hypermedia documents and systems exists, based on different aspects of hypertext and multimedia, in different combinations.

Most hypermedia systems provide functionality that is perhaps best described as "multiple media hypertext". These systems were originally developed as hypertext systems, and afterwards extended to handle new media types. They are based on the node/link model of hypertext, only now the nodes may contain media types other than text. Because the addition of multimedia in these systems does not change the underlying data and process models, researchers in this area often use the terms "hypertext" and "hypermedia" as being interchangeable. A good example is the Dexter model discussed before: addition of new media types in Dexter's within-component layer does not affect the basic hypertext data structures modeled by the storage layer, nor does it affect the navigation-based interaction process modeled by the runtime layer. Another example is HTML, where neither the potential of including other media types in an HTML document, nor the presence of links to other media objects have fundamentally changed HTML's document model or the browser interface.

Other hypermedia systems provide "interlinked multimedia". They have their roots in multimedia rather than hypertext. Typically, these are multimedia systems that are extended to provide navigation-based interaction in addition to the more traditional — VCR-style — interaction mechanism. These systems support the specification of synchronization constraints on their constituent media items, and have built in support for defining multimedia layout as discussed in the previous section. Since the hypertext node/link model is subordinate to the spatio-temporal composition model, their linking facilities are usually not as sophisticated as those provided by their hypertext counterparts. For example, these systems are typically based on a linear, time-based document model, and links can only be used to jump up and down the linear timeline, or to jump to other documents. A good example in this category is the document model underlying the Synchronized Multimedia Integration Language (SMIL 1.0 [231]). As its name suggest, SMIL has a strong focus on multimedia synchronization, with an extremely simple hyperlink model.

A full integration of hypertext and multimedia could by described as "non-linear multimedia", but is not (yet) widely supported (see, for example, the extensions of Halasz's seven issues to distributed hypermedia systems by Buford and Rutledge in [47]). Non-linear multimedia also combines support for multimedia synchronization with hyperlinking, but provides additional support for non-linear document structures. This more fundamental integration of hypertext and multimedia is the topic of this section. It combines the advantages of both paradigms, but gives also rise to some new problems.

2.5.1 Research Issues

In the previous sections we discussed the main hypertext and multimedia research issues. Typical research issues that especially relate to the integration of multimedia and hypertext documents include:

- the combination of the linear, time-based structure found in multimedia documents with the non-linear, link-based structure which characterizes hypertext documents,
- link traversal and other forms of user interaction in a presentation with multiple active media streams,
- linking from and to alternative content, and
- efficient multimedia playback in an interactive environment.

Link-based versus temporal structures

As pointed out by Hardman et al. [122], many hypermedia systems employ hierarchically structured document formats, but offer support for dynamic media types only in the leaf nodes of the document tree structure. A straightforward addition of multimedia content to an existing hypertext system will result in a system that lacks synchronization facilities. This argument also holds the other way round. Adding hyperlinks to a linear multimedia document will result in an essentially linear hypermedia document, where the links can only be used to fast-forward or rewind the presentation. The only way to escape the linear structure of the document is to provide links to other, separate documents. This solution will require authors to split up their presentation in several interlinked, but otherwise unrelated parts, and is likely to destroy the perception of a single, integrated presentation, from the perspectives of both the author and the end-user. A more sophisticated model for integrating hyperlinking and temporal synchronization, based on non-linear temporal structures, is proposed by the Amsterdam Hypermedia Model (AHM), discussed below.

Link navigation in documents with multiple active media streams

When multiple media streams are active (which is usually the case in multimedia presentations), link traversal does not necessarily have to affect all media streams. Somehow, the author needs to be able to control the scope of the effect link traversal has on the behavior of the media items in the presentation.

Note that the notion of link scope is applicable to all systems that support both linking and composition. As such, it could also be applicable to hypertext. As mentioned by Halasz (in one of his seven issues [112], see also page 36, Section 2.3 of this thesis), the semantics of link traversal from and to nodes within a composite component is not clear. In the Dexter model for example, when the destination of a link is a component that is part of a composite component, it is not clear what the effect of link traversal is on the other parts of the composite. Presentation of the destination component without its siblings, or even the entire composite, might be meaningless. Since it is dependent on the application whether this is the case or not, this kind of link information should be an explicit part of a hypermedia document model.

Linking to alternative content

When alternative multimedia content (as described in the previous section, see page 50) is used as the destination of a link, this destination could turn out to be unavailable during link traversal in a specific presentation. To avoid this scenario, linking to alternative content can be forbidden (this is, in fact, the case in the SMIL 1.0 switch). A better solution could be to allow the use of a mechanism which is guaranteed to resolve to a valid anchor value for all alternatives specified. This would suggest a more advanced model supporting indirect anchoring.

The same problem, however, could also be regarded as just one example of a more fundamental issue. In most models, links and anchors are defined in terms of the underlying document structure, but their effect is defined at the presentation level. For applications that use a direct and simple mapping between the document structure and the presentation, this is not likely to be a problem. But in more and more applications, there is an increasing difference between the document and presentation structure. Adaptation of documents to a specific quality of service or to other system specific requirements increases the differences between the document and the final presentation. The same applies to other adaptive techniques, including style sheets, transformation sheets, and scripting. All these techniques can cause fundamental differences between the source document and its ultimate presentation. These differences may lead to situations in which a link defined on the document level can no longer be appropriately translated to the presentation level. On the other hand, if the user interacts with a specific part of the presentation (e.g. by clicking in a specific region), these techniques may make it extremely hard to find out which anchor or link is associated with that region.

Interaction versus performance

Most networked multimedia applications benefit from the linear structure of their documents by pre-loading, buffering and preparation of data needed in the future, in order to improve performance. In an interactive hypermedia environment, efficient techniques to predict the "next" items that are needed, are hard to develop because this requires insight in the choices a user will make at runtime. Still, because of the sheer size of audio and video fragments, most multimedia applications will need such techniques in order to meet performance requirements.

Most multimedia systems support the common VCR style interaction controls (e.g. play, pause, fast forward, etc.). In addition, hypermedia systems require pure random access to be able to link into the middle of a presentation. At the same time, it is the non-linear structure of a hypermedia presentation that complicates the realization of random access methods. In a non-linear presentation, the state of the presentation at a particular moment is typically dependent on previous user interaction. Starting the presentation at a random moment is thus more complicated than in the case of a linear, non-interactive multimedia presentation.

2.5.2 Modeling hypermedia documents

In this section, we discuss the Amsterdam Hypermedia Model (AHM [119]) as an example of a Dexter-based model that combines temporal synchronization and spatial layout, hyperlinking and a non-linear temporal structure. The model does not, however, address issues related to alternative content, nor does it model user interaction that goes beyond hyperlink navigation.

Document structure

The AHM extends the Dexter hypertext model with a structure-based temporal model, which can be considered as a generalized version of the parallel/sequential hierarchy discussed in the previous section on multimedia documents (see page 53). It also extends the Dexter link with a notion of link scope as discussed above, and models multimedia spatial layout by introducing the concept of a *channel*.

The AHM uses Dexter's composition mechanism in two ways. Composition is used to model the temporal structure of the document (*temporal composition*), this allows components to be grouped on a single, linear timeline. The other mechanism, *atemporal composition*, groups components without imposing any predefined temporal constraints. Since an atemporal composite itself can be grouped within a temporal composite, it effectively models a non-linear temporal structure by creating an open slot in the linear timeline. The contents of this slot are determined at runtime, typically as the result of link traversal. See Figure 2.8 for an example. Here (a fragment of) the document structure consists of a temporal composite (T_1) as the root component, which contains another temporal composite (T_2) and an atemporal composite (A). The two temporal composites specify different time constraints: the children of T_1 are played in parallel, the two atomics of T_2 ($1a$ and $1b$) are played in sequence. Because it is an atemporal composite, A does not specify the temporal behavior of it's children, the atomics $2a$ and $2b$. Whether or not, at what time and in what order these atomics will be played is determined at runtime, typical by means of link navigation (that is, the user needs to select links to either $2a$ or $2b$, or to both).

Note that since the durations of $2a$ and $2b$ differ, the duration of A can typically only be determined at runtime. Despite the fact that A is an atemporal composite, it has to respect the basic rules of the Allen during hierarchy: its children cannot begin before the begin of their parent, nor can they end after their parent ends. The same applies to the atemporal itself: it can only be active when its parent (T_1) is active. So the non-linear aspect of the atemporal composite is that it creates a "hole" in the document timeline, and the content, the order and the timing of the content of that hole are only determined

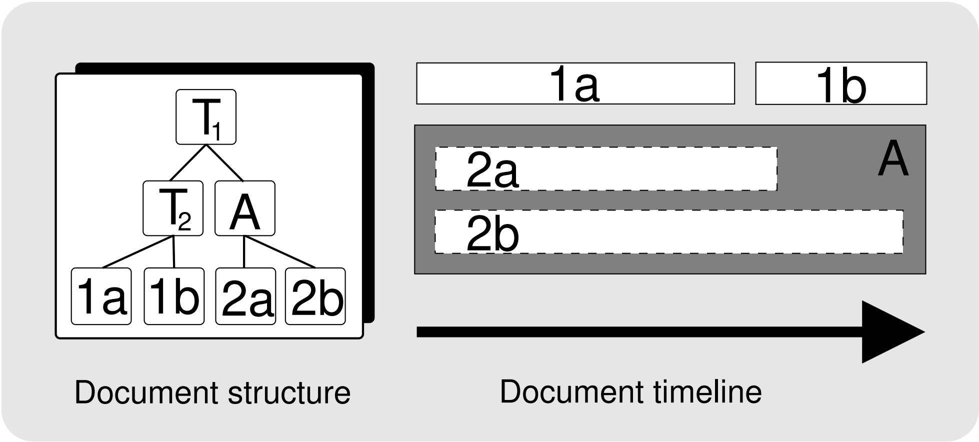

Figure 2.8: Atemporal composition in the AHM. The exact timing and order of the children of A is determined at runtime.

at runtime. At the same time, the atemporal composite is fully integrated in the linear timeline of its parent, and as such it has to conform to the rules of the during hierarchy.

Link scope

Suppose that the components $2a$ and $2b$ in the example above are used as the targets of two links, and also assume that both $2a$ and $2b$ should always be presented in combination with $1a$ and $1b$ (this could be the case, for example, if $2a$ and $2b$ are different subtitles associated with videos $1a$ and $1b$). The links need not only to be able to point to (anchors in) $2a$ and $2b$, they also need to specify that upon link traversal, the entire subtree below T_1 needs to be played. Note that a similar issue applies to links that start from $2a$ or $2b$. Such links need to define, for instance, what to do with the other children of T_1 in case $2a$ or $2b$ are stopped or paused due to link traversal. We call T_1 the (destination or source) scope of the link[6].

This concept of link scope is defined in the AHM by extending the Dexter link model with *link contexts*. Contexts are used to explicitly encode the scope of the source and destination endpoint of a link [120]. This allows authors to create, for example, hyperlinks that replace the entire, currently running, presentation by the presentation which is the destination of the link. Other links, however, may only replace a single video stream by a new one, or leave the currently running streams active and start a new audio track in parallel.

Closely related to contexts are *activation states*, which specify for each component in the source context whether play-out of the component should be stopped, paused or continued in case of link traversal. The same applies to the destination context, where the activation state is used to model for each component whether it is to be activated in a normal playing or paused state. A common scenario for the latter is linking to a specific

[6]Modeling the link as a link to T_1 instead of a link to $2a$ or $2b$ gives similar problems, because now the link needs to specify, in addition to its target, whether $2a$ or $2b$ should be played.

frame in a larger video fragment, where the video should be paused on the target frame when the link is followed.

Spatial layout

Finally, the AHM explicitly models spatial layout by assigning each media element a *channel*. The channel is used to localize the definition of screen size and position (and other properties), so that it can be shared across multiple media elements. The AHM channel is, from the perspective of spatial layout, similar to the region model of SMIL 1.0. In both models, screen space is specified relative to a coordinate space, either in absolute coordinates or in percentages of the total screen size. In addition, channels in the AHM can be hierarchically organized. While not explicitly modeled by the AHM, this allows for a number of interesting features, including the use of this hierarchy to define inheritance of visual style properties, as an alternative to the more common inheritance in terms of the (temporal) document hierarchy. These issues are discussed in more detail in Chapter 5.

2.5.3 Relationship with other document models

In Section 2.3 on hypertext, we discussed how linking can be added to a document model in a way that maintains a strict separation between structure and presentation. In multimedia, part of the spatio-temporal relations in the presentation are tightly bound to the document's overall structural semantics. A strict separation between structure and presentation is therefore much harder to make.

In hypermedia, the distinction between structure and presentation is further blurred by concepts such as link context and non-linear temporal structures. Context information, for example, can be regarded as part of the link structure and semantics, in which case the context of the link will not vary in different presentations. Alternatively, it can be thought of as part of the presentation information, in which case the context only influences the style (e.g. the scope of a transition effect) which is likely to vary in different versions of the document. The mix of temporal and non-temporal structures in a single hierarchy as discussed above, also leads to a combination of structural and presentational aspects of the document. Non-linear temporal constructs as discussed above, for instance, can be effectively used to improve the logical structure of a multimedia document, for example to separate material belonging to the main narrative from optional material. The same constructs, however, could implicitly specify a duration hierarchy that conflicts with the preferred timing of a specific presentation. Below, we take a close look at these issues by discussing the abstractions of the AHM and the HyTime standard in terms of the multiple delivery publishing model.

AHM and multiple delivery publishing

The Amsterdam hypermedia model (AHM) was developed as a platform-independent hypermedia model. Meeting other requirements related to multiple delivery publishing was never an explicit part of the design rationale. Nevertheless, it is worthwhile to briefly discuss the AHM in terms of the content, structure and presentation layers of the multiple delivery publishing model. Note that since the AHM does not specify a specific file format, the differences between the presentation and realization layers are not relevant.

Following Dexter, the content level is modeled in the AHM's within-component layer and considered to be beyond the scope of the AHM. The main interface between the within-component layer and the storage layer, or, from our perspective, between the content and the structure level is provided by the anchor concept. Anchoring is used for defining the endpoints of both links and synchronization arcs. To allow spatio-temporal composition, the within-component layer also communicates the bounding-boxes (spatial dimensions and duration) of the media items to the higher layers.

The structure level itself is modeled by the various components in the storage layer. The focus is on the temporal and atemporal composites and on the link component. Together, the components model the temporal and navigation structure of the document.

Note that all explicit temporal information is modeled using synchronization arcs and other attributes that are part of the presentation specification. Presentation specifications in the AHM play a role that is comparable with the role of style sheets in the multiple delivery publishing model. From this perspective, the AHM separates presentation information from the main document structure.

This document structure, however, specifies a strict duration hierarchy. A component can never start before the start of its parent, nor can it end after its parent has ended. This means that the only way to alter the temporal scenario of a presentation without modifying the document structure, is by modifying the synchronization arcs. Such changes will "bubble up" the temporal hierarchy and affect the start and end-time of other media elements. If this does not yield the desired effect, the document structure itself needs to be changed in order to change the duration hierarchy.

Note that some temporal information, such as the intrinsic duration of a media item, is necessarily defined at the content level. In other words, the presentation's temporal behavior cannot be localized at the presentation level, but is necessarily spread across the media content, document structure and presentation specifications. While this is in line with our conclusions of the previous section, the AHM does not make an explicit distinction between spatio-temporal constraints that directly affect the document's narrative versus spatio-temporal constraints that only affect the style of the presentation.

As most models, the AHM combines both semantic and navigational aspects in its link model. Linking and link context is entirely defined as part of the document structure. Defining alternative navigation structures or link contexts at the presentation level (e.g. by a style sheet) is not supported. While anchors are modeled as being part of a component, the AHM does follows Dexter's notion of links as first class components. Since there are no predefined semantics attached to the position of the links within the document tree, the specification of links can be easily separated from the temporal hierarchy. Activation states (defining the runtime behavior of the link and its contexts) are also defined at the structure level, and not at the presentation level. The only concepts related to links that are entirely defined at the presentation level are the anchor style definitions.

Structured hypermedia and HyTime

Unlike the AHM, the HyTime standard was explicitly developed for applications that are based on a multiple delivery publishing model. HyTime is one of the few hypermedia formats that abstracts from the presentation details of a hyperdocument. HyTime extends, as discussed in the sections on hypertext and multimedia, the basic hierarchical composition facilities of SGML by defining abstractions for addressing, hyperlinking and alignment. HyTime does not provide explicit concepts for spatial or temporal alignment, but allows an application to define a set of abstract, n-dimensional *finite coordinate spaces*. Information in the document which needs to be aligned with respect to a defined coordinate system can be associated with a HyTime *event*. Each event needs to define its *extent* in terms of the axes defined by the finite coordinate space.

While HyTime's coordinate spaces can be used to define all kinds of spaces (e.g. a color space) they are commonly used to align the contents of a document in space and time. Examples include positioning the description of a historic event on a timeline, the position of a city on a map, or the location of a specific part in a three-dimensional model of an airplane. Note that this type of space and time information is very different from the use of the same dimensions in the AHM. In the AHM, space and time information describes where media items will appear on the screen, and when they should be started or stopped.

The spatio-temporal and navigation characteristics of the presentation may be closely related to the space, time and link structures in the underlying HyTime document, but this relationship need not be a simple one-to-one mapping. The temporal ordering of historic events, for example, might be represented by a spatial ordering when the events are positioned on a graphical representation of the timeline, as is common in history textbooks. In an alternative presentation, the user might be able to navigate from one event to the other. See Rutledge et al. [199] for a more in depth treatment of the relation between space, time and linking on these two different levels.

2.5.4 Standardization

The most significant ISO standards for hypermedia documents, addressing both synchronization and linking, are HyTime and MHEG-5. They can be seen as complementary since MHEG defines a final form representation for hypermedia documents (in the way PostScript defines a final form representation for text documents), ready for delivery over a network to a remote presentation system. MHEG documents use a low-level encoding format to minimize the requirements of the presentation system (typically a television set-top box). Most temporal synchronization is realized (server-side) by an interleaved data representation. MHEG can model simple non-linear presentation structures by supporting link-based navigation through different scenes.

In contrast, HyTime captures the logical structure of a hypermedia document (as SGML does for text documents), leaving much of the specification of presentation details and user interaction to the processing application. HyTime documents are declarative in nature, have rich semantics which capture the logical structure of the document and need further processing in order to generate the final presentation. Note that such a presentation could very well be encoded by using, for instance, MHEG-5. See [196] for a comparison of HyTime and MHEG.

2.5.5 Summary

In the previous sections we concluded that in order to support hypertext and multimedia, the multiple delivery publishing model needs to be extended with facilities for linking and spatio-temporal constraints on all levels of the model. In this section, we saw that these facilities are not independent, and should be integrated in order to provide full hypermedia support.

The distinction between structure and presentation is blurred by often essential spatial relationships and concepts such as (a)temporal composition and link context. Therefore, a multiple delivery publishing model for hypermedia should not be based on the separation of structure from visual appearance, but on separating those aspects that are likely to vary across presentations from those aspects that are likely to remain constant. While such a model keeps the main advantages of the multiple delivery publishing model, a disadvantage is that it requires support for spatio-temporal composition and linking on all levels of the document processing model. Additionally, it complicates the authoring process because specification of the visual appearance can no longer be localized in the document's style sheet.

We discussed the Amsterdam Hypermedia Model (AHM) as an example of a non-linear multimedia document model. The AHM breaks down Dexter's notion of a presentation specification (whose internal structure is not defined in Dexter) into a style part (whose inner structure remains undefined in the AHM), and a part that specifies the presentation information which is considered part of the document's structure and semantics. This includes basic spatial layout information, link context and additional temporal information. A more formal description of the AHM will be given in Chapter 5.

2.6 Conclusion

Structured documents have been used primarily for content-driven, text-based applications, and are characterized by the separation of the main document structure from presentation related information. This distinction between structure and presentation underlies the multiple delivery publishing model as described in this chapter, which has proved to have significant advantages in terms of document longevity, reusability and tailorability. Disadvantages include an increasingly complex document processing model and a larger dependence on high quality tools.

Structured text documents come in two flavors: domain-independent and domain-specific. The models of the documents from the first category are mainly based on abstractions commonly found in texts such as sections, footnotes, bulleted lists, etc. These concepts typically model a linear, text flow structure, building a coherent narrative. This narrative needs to be conveyed to the user by converting this structure into a presentation. Style sheets map the structures in the document onto the structures that characterize the presentation, such as pages, columns, headers, footers, etc. Because the structures on both levels are usually well-known, reasonable default mappings can be defined, in which case style sheets only need to add information commonly recognized as “style” information. Different styles influence the appearance of the document, but have minimal or no impact on the underlying narrative that needs to be conveyed.

The models of the documents in the second category are based on domain-specific structures. These documents often lack an overall narrative that needs to be preserved in the presentation. To convert these documents into a specific presentation, the style sheet needs to select the material to include in the presentation, and the order in which this material is presented. Additionally, applications typically use many different domain-specific structures, for which it is hard to define useful default mappings. For this type of application, the name “style sheet” may be confusing, because the style sheet not only adds style information, but completely controls the presentation. These style sheets are more likely to have a significant effect on the semantics of the “message” that is conveyed by the presentation.

Both the domain-independent and domain-specific models sketched above are content-driven. The focus is on the content, which need to be presented in various forms, using several layouts. Hypermedia applications are more presentation-oriented. Some hypermedia documents are said to be layout-driven, and for this type of documents, the separation of structure and presentation is *a priori* neither feasible nor desirable. But there is a growing number of content-driven hypermedia applications which could, in theory, benefit from the advantages of structured documents. Nevertheless, structured documents are rarely used for hypermedia applications. This is partly because it is hard to develop generally useful, domain-independent models for hypermedia that abstract from the presentation. But while domain-specific hypermedia models do exist, tools for authoring and converting these documents to hypermedia presentations are hardly available.

In this chapter, we discussed several differences between traditional structured text, hypertext, multimedia and hypermedia documents. These differences mean that many models, tools and techniques that are developed for structured text, are not readily applicable to structured hypermedia. The most important differences between structured text, multimedia, hypertext and hypermedia documents are summarized in Table 2.2.

One of the most important differences we have discussed in this chapter between structured text, multimedia, hypertext and hypermedia is the main document structure. This typically reflects a linear, text-flow based structure for text, a linear, time-based flow for multimedia and a non-linear, interlinked structure for hypertext and

	Text	**Multimedia**	**Hypertext**	**Hypermedia**
Main structure	linear (text)	linear (time)	non-linear	non-linear
Behavior over time	static	scheduled	interactive	mixed
Structure encoding	embedded	embedded	external	mixed
Presentation encoding	external	embedded	external	mixed
Separation between:				
Application/document	strict	fuzzy	strict	fuzzy
Procedural/declarative	strict	fuzzy	strict	fuzzy
Structered/visual markup	strict	fuzzy	fuzzy	fuzzy
Presentation/realization	common	very rare	rare	very rare

Table 2.2: Structured text, multimedia, hypertext and hypermedia documents.

hypermedia documents. Another difference is the behavior of the corresponding presentations, characterized by the static nature of text, the inherent time-based nature of pre-scheduled multimedia versus the highly interactive navigation style that is typical for most hypertexts. Hypermedia documents integrate (or better: should integrate) both pre-scheduled and interactive behavior. Additionally, in structured text it is common to encode the document structure (including links and metadata) by means of embedded markup, using style sheets to encode the presentation-related information externally. Most multimedia applications encode all information into a single document, while hypertext (especially open hypertext) advocates the externalization of links and other important document structures. In hypermedia, both techniques are used.

The strict separation between the application and the document that characterizes most text and hypertext applications, is, in the case of multimedia and hypermedia, often blurred by scripting languages and techniques that are application or platform independent. The same techniques also blur the line between declarative and procedural parts of a these documents. Many of the current declarative hypermedia and multimedia models are not sufficiently powerful to cover the wide range of interactive and dynamic behavior that can be obtained by using more procedural techniques such as scripting, applets, plug-ins etc.

The distinction between presentation-independent, structured markup versus presentation-oriented, visual markup is also fuzzy in the case of multimedia and hypermedia because many of the spatio-temporal relations that determine the way the document content is presented, may affect the "message" of the presentation and should thus remain constant over multiple presentations. These relations should hence not be part of a separate and replaceable style sheet, but are considered to be an intrinsic, non-variable part of the document structure itself. Note that not all spatio-temporal relations do indeed affect the contents, so some of them may indeed vary across presentations and as such be good candidates for inclusion in a style sheet. This implies that hypermedia requires support for spatio-temporal relations on all levels in the document processing chain. For hypertext, separating semantic relationships from navigational hyperlinks is not common, most links are either purely navigational or mix both aspects.

The last distinction is the one between a presentation and the realization of this specification in a specific output format. This distinction is seldom made in hypermedia, mainly because the lack of a sufficiently abstract and standardized presentation format (that is, there is no hypermedia equivalent of the flow object model used for text). The only exceptions are perhaps the strongly text-based hypertext applications, where linking is limited to traditional textual references such as footnotes, bibliographical references, etc. These structures can usually be mapped to the simple navigational structures supported by the flow object models that are currently in use.

The next chapter illustrates the models described in this chapter by discussing their application on the World Wide Web. We assess the pros and cons of the original Web protocols by revisiting the research issues discussed in this chapter, and use the more recent Web developments related to XML to illustrate many of the techniques described so far.

Chapter 3

Hypermedia on the World Wide Web

The World Wide Web is — by far — the world's largest and most successful hypermedia application. Compared to the models discussed in the previous chapter, however, the document model of the Web is surprisingly simple. From the very beginning, the Web has been criticized for its simplistic approach to document markup, especially by SGML advocates. The hypertext community criticized the simple link model supported by the Web — a far cry from the linking capabilities of the systems hypertext researchers were used to. Synchronized multimedia was initially not supported at all.

Although the underlying models were simple, the Web had something all the other systems discussed so far did not: it provided its users with a uniform interface to the Internet, which gave them easy access to a virtually unlimited amount of information. Additionally, the Web's client/server architecture effectively separated document storage from document presentation. This separation of concerns allowed for easy experimentation and new developments at the client side, without the need to modify servers or the underlying protocols.

In this chapter, we evaluate the first generation Web protocols and document models in the context of the research issues discussed in the previous chapter. Additionally, we discuss the recent XML-related developments on the Web to illustrate the most relevant issues related to structured hypermedia processing.

3.1 Overview: Basic Web Protocols

The World Wide Web originates from a project initiated in 1989 by Tim Berners-Lee at the European Laboratory for Particle Physics (CERN). Researchers at CERN needed access to the articles and reports of their colleagues at laboratories located around the world. While available over the Internet, researchers found it difficult to access these documents for various reasons[26]:

- differences among (client) desktops in use at CERN, running different operating systems with different user interfaces,
- differences among the (server) platforms used at the other laboratories,

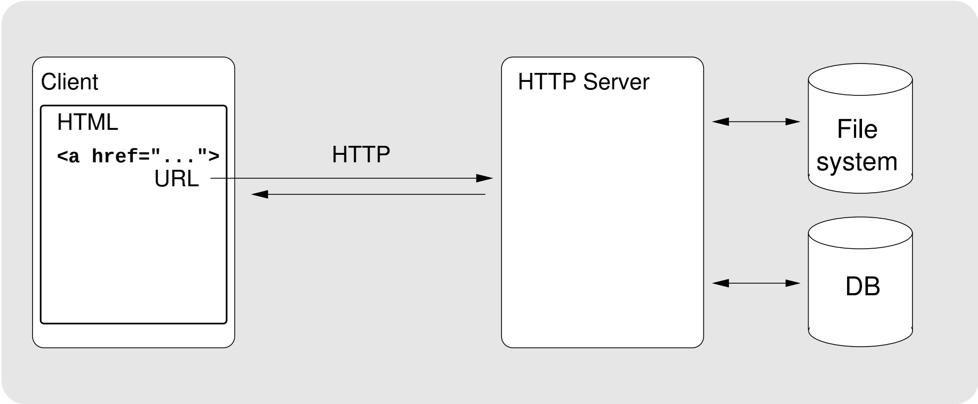

Figure 3.1: The first generation Web specifications: URL, HTTP and HTML.

- differences among document file formats; often document formats used by the author could not be processed on the platform of the reader,
- differences among the applications needed to retrieve the documents, and
- difficulties related to remembering the host names of the servers and locations of the files they were looking for.

Berners-Lee proposed to solve these problems by developing a single application that provided its users with a uniform interface to all applications and protocols relevant to document access in use at CERN. Naturally, this new application itself should run on multiple platforms. By using a hypertext interface that provided access to remote documents, users were no longer forced to remember host names, file locations, or the command syntax of the transfer protocols.

To realize such an application, two main issues needed to be addressed. First, and perhaps most important, the application needed a uniform scheme for locating files on the Internet. Second, there was a need for a simple syntax that could be used to encode documents that contained the hyperlinks that used the new location scheme.

Berners-Lee proposed to solve the location problem by combining the local filename of a document, the host name of the server and the name of the protocol being used to transfer the file from the server to the client. This combination formed the basis of the concept of the URL (discussed in Section 3.1.1 below). For the second problem he proposed a simple, text-based document format, with a syntax that was inspired by SGML. This format formed the basis for HTML (Section 3.1.3). While the resulting system worked, it was relatively slow. This was largely caused by the existing protocols (such as FTP), which were designed for long interactive sessions with human users. The overhead required to keep track of the state of the client program and to perform extensive error checking was prevented by developing a new, light-weight, stateless protocol: HTTP (Section 3.1.2). The result is sketched in Figure 3.1.

Soon, the Web was also used outside CERN. While support for a graphical user interface was an explicit non-goal in Berners-Lee's original project proposal [26], usage of the Web literally exploded after the introduction of Mosaic, the first commonly available browser with a graphical user interface[172]. Since then both research and commercial organizations extended and improved the existing Web applications. A major part of the relevant development and standardization work is carried out under the umbrella of the World Wide Web Consortium (W3C). Many new protocols and extensions have been proposed — some of them have become a success, but many of them are already history. Against this background, the three basic specifications (URL, HTTP and HTML) have been remarkably stable. Before evaluating the Web against the context of the electronic publishing, hypertext and multimedia requirements formulated in the previous chapter, we first briefly discuss these three basic specifications.

3.1.1 Uniform Resource Locators (URL)

An important aspect of any hypertext system is the method used to specify the endpoint of a link. First, a system needs to specify the components that form the endpoints of the link (i.e. *component specification* in Dexter). Second, a system needs to specify the names or locations of portions of those components that define the "hot spots", the active source (or target) areas of the link (i.e. *anchoring* in Dexter). Anchoring mechanisms typically depend on the way the component is encoded[1]. Therefore, we discuss anchoring on the Web in our description of HTML in Section 3.1.3.

Here we only deal with the first issue: specification of the name or location of the target component. In Web terminology, every component that can be addressed and accessed as a single unit is called a *resource*. Examples of resources on the Web include HTML pages, image files, style sheets, scripts, applets, etc. Currently, virtually all resources on the Web are addressed by means of a Uniform Resource Locator (URL) [24, 25].

The idea behind a URL is very simple — one can think of it as an Internet extension of a file's pathname. A URL describes the *location* of a document, i.e. it specifies the hostname of the server, the protocol used and the name of the resource relative to that server. Consequently, renaming or moving a resource is likely to obsolete *all* URLs pointing to it. Additionally, it is difficult — if not impossible — for a client to take advantage of the fact that there are sites that mirror (part of) the data available from other sites. At the time of writing, it is common practice to retrieve a resource from the other side of the world, because both user, browser and server are not aware of the fact that there is a copy of that particular resource locally available.

To resolve these problems, the concept of a *Uniform Resource Name* (URN [170, 212]) has been proposed[2]. While the URL can be thought of being a resource's *address*, the URN should play the role of the resource's *name*. To complicate the terminology, both URNs and URLs are members of a more general family, called *Uniform Resource Identifiers* (URI) [24].

[1] see also Chapter 2, page 41.

[2] At the time of writing, the discussion about URNs has been going on for more than five years, but as yet no concrete results of this discussion have been widely implemented.

Despite its disadvantages, the URL is simple and generally applicable. The URL is still the only commonly used addressing scheme of the Web. Being the glue that connects HTTP, HTML and many other Web protocols, the URL can be considered to be one of the fundamental concepts underlying the Web's infrastructure.

3.1.2 Hypertext Transfer Protocol (HTTP)

The basic communication protocol used by Web servers and clients is HTTP, the *Hypertext Transfer Protocol*. Early prototype Web-clients used existing transfer protocols, such as the FTP protocol [182]. The overhead involved in establishing, maintaining and releasing a single FTP connection triggered the development of HTTP: a faster, lightweight, stateless protocol, virtually without error checking, built on top of TCP/IP. Implementation of the protocol proved to be extremely simple, both for the client and the server, and this simplicity certainly helped in the rapid adoption of HTTP. While HTTP has been widely used since the early days of the Web, the first official version, HTTP 1.0 [23] was standardized only in 1996.

An HTTP 1.0 client merely establishes a connection and requests a single document. The connection is terminated as soon as the document has been received. This has the obvious disadvantage that for a document with ten inline graphics, one TCP connection is needed for the document itself, and ten connections for the graphics. To avoid the overhead related to setting up repeated connections to the same server, HTTP 1.1 allows a client to keep a connection open to request several documents from the same server in a single TCP connection [99]. Current research is focused on the use of distributed object technology as the basis of the next generation (extensible) HTTP protocols [234]. Other HTTP related research relates to caching protocols, content type negotiation and quality of service negotiation.

Most of the traffic on the Web uses HTTP. The Web is, however, by no means bound to the use of HTTP. All popular Web browsers can also retrieve files by other protocols: Gopher, WAIS, FTP and NNTP access is supported by most browsers, as well as access to files on the local file system. In the early days of the Web, when the number of HTTP servers was still low, support for these protocols contributed significantly to the early adoption of Web technology. Later, the simplicity of HTTP and the quality of HTTP servers that were freely available, rapidly made HTTP the most common transfer protocol. The Web's ability to integrate other protocols has again become relevant with the introduction of time-based media on the Web. This requires real-time alternatives to HTTP, such as the Real-time Streaming Protocol (RTSP [206]), which are now gaining popularity.

3.1.3 Hypertext Markup Language (HTML)

HTML (Hypertext Markup Language) was originally developed as a simple markup language for Web-documents, with a syntax that was inspired by SGML. It was primarily designed to encode simple hyperlinks, and, additionally, to provide some basic layout and structuring facilities.

The most generally applicable link in HTML is the `<a>` element, which is further discussed below. Additional link-related elements provided by HTML include the `area` element for defining image maps (which make specific portions of an inline image behave as the source anchor of a link), the `base` element that defines a URI value that overrides the default value used to expand relative URIs to absolute URIs, and the `link` element (only allowed in the document header), which defines general link relations between its containing document and other documents.

Links encoded by the `<a>` tag are one-directional, embedded links. Both the link itself and the anchors are encoded in a single element. HTML links point to a target resource by employing the concept of the URI. The destination of the link is typically the complete resource that is associated with that URI, though a portion of that resource can be explicitly addressed by extending the URI with a *fragment identifier*. For example, the following encodes the text `Chapter 3` as the source anchor of a link to a fragment in a page called `book.html`, where the fragment is named `chapter3`:

```
See <a href="http://www/book.html#chapter3">Chapter 3</a> for
more information.
```

This works only if the page `book.html` contains a destination anchor with a matching `name` attribute:

```
<a name="chapter3"><h1>Chapter 3</h1></a>
<p>...
```

While simple and extremely scalable, HTML linking has the drawback that adding or modifying outgoing links always requires modification of the source document. This makes basic hypertext features such as personal annotations hard to implement. Additionally, linking to a portion of another document requires that that portion is already marked as a named anchor by the author of the document. To overcome these limitations, new specifications are being developed, which are discussed in Section 3.3 below.

Besides its link-related elements, HTML provides a larger number of elements that are used for the encoding of the text layout and document structure. While the initial versions of HTML contained only a few elements, subsequent versions included more and more elements. In addition, browser vendors added their own elements. HTML 3.0 defined many elements that were neither used nor implemented [184]. To enhance interoperability there was a need for a specification that better reflected the everyday use of HTML. This was the main reason for the development of the HTML 3.2 [185] specification, published in January 1997.

As mentioned above, HTML's syntax was initially inspired by SGML. But even HTML 3.2 could hardly be classified as a true application of SGML. Most browsers did not (and still do not) use SGML parsers, and the majority of the HTML documents on the Web was (and still is) syntactically incorrect, which makes it hard to process them with most existing SGML-based software. A more fundamental problem is that HTML markup is typically not strictly presentation-independent: most HTML documents contain a mixture of structured and visual markup. W3C addressed the critique of the SGML community, and moved its focus from adding more and more (presentation-oriented) elements to other techniques. This resulted in the HTML 4.0 specification [186] in December 1997, standardizing, among other things, the use of CSS style sheets in HTML. To a large extent, HTML can be said to have come back to its roots: virtually all presentation-oriented elements are deprecated in HTML 4.0, which ironically makes it rather similar to the initial versions of HTML.

3.1.4 Summary

The three specifications that formed the basis of the early Web architecture (URLs, HTTP and HTML) are in many ways still the fundamental protocols of the Web as it exists today. One of the strengths of the original Web design is that although many extensions and new protocols have been added, these additions have hardly required major changes to the underlying model. New document formats now provide alternatives to HTML, and new transfer protocols have been introduced as an alternative to HTTP, without substantially changing the Web's original client/server architecture.

To a certain extent, the only absolute technical requirement on the Web is conformance to the generic URI syntax, since URIs form the glue that connects all Web protocols and URI parsers are part of virtually all Web-related software. As long as data formats or network protocols use URI-based addressing or naming schemes, they seem to fit within the overall infrastructure of the Web.

To fully exploit the advantages of Web-based hypermedia, however, a much closer, document-level integration is required. In the following sections, we explore the needs and the requirements for this type of integration. We evaluate the Web's initial and current protocols with respect to the hypermedia research issues presented in the previous chapter. In Section 3.2, we revisit the general issues and requirements related to structured documents and electronic publishing presented at the beginning of Chapter 2. In Section 3.3, we evaluate the hypertext-related capabilities of the Web. We conclude with an evaluation of the Web's support for multimedia in Section 3.4.

3.2 Structured Documents on the Web

While the lack of structure in most documents found on the Web has been frequently criticized in the literature, certainly not all applications on the Web require document formats with rich structured semantics: "...data transmitted across the Web is largely throw-away data that looks good but has little structure" [63]. Therefore, Web protocols should not enforce structure on applications that do not require it. On the other hand, there are applications that depend on a sophisticated document structure, such as the more content-driven applications discussed in the previous chapter. For those applications, standard protocols should be available to facilitate dissemination of structured documents over the Web.

In this section, we evaluate the Web against the electronic publishing requirements discussed in Chapter 2. Additionally, we discuss two recent Web specifications that address many of the Web's limitations that relate to structured document processing: CSS and XML.

3.2.1 Requirements

As a transport protocol, we consider HTTP to be neutral with respect to the document-level electronic publishing requirements discussed so far. The impact of URLs on these requirements is discussed in the next section. Below, we focus on the functionality offered by HTML in the context of three important issues in electronic publishing: longevity, reusability and tailorability.

- **Longevity** — HTML addresses this issue primarily by offering platform independence, which is a basic requirement to protect documents from changes in the hard- or software environment. On the Web, platform independence has been an important requirement, from the very first drafts [26]. Despite a lack of backward compatibility, vendor-specific extensions and other differences among different browser implementations, the HTML specification in itself defines a document format that is highly platform-independent. In practice, however, the longevity of the content of many (generated) Web-pages is often limited, and this makes the longevity characteristics of the document format a less important issue. In addition, documents with content that is valid for a longer time frame, often require frequent updates due to the pressure to keep up with the rapid changes in browser technology. This makes HTML less attractive as a document format for longer term storage.

- **Reusability** — Support for the single source, multiple delivery publishing model facilitates the reuse of documents and document fragments in another context, and is also another way to support longevity. This, however, was not part of the original Web requirements: HTML was explicitly designed as a markup scheme for presentation in a single, uniform browser environment. Additionally, early versions of HTML mixed structure and presentation information, several non-standard presentation-oriented extensions to HTML were frequently used, and, until recently, there was no generally accepted style sheet format. These limitations made HTML unsuitable as a source format in the single source, multiple delivery publishing model. As an output format, however, HTML has been extremely successful as a means to make SGML and other non-HTML sources easily accessible on the Web.

- **Tailorability** — As HTML's main objective is to provide a commonly useful, domain-independent document format, it does not support document markup tailored to the specific needs of a particular application. As stated above, HTML has proven to be useful as an output format to disseminate domain-specific information originally encoded in SGML. This requires a server side conversion from the domain-specific SGML encoding into HTML. The main drawback of this conversion process is the loss of structuring information, information that is no longer available for client-side applications. Applications that need access to this type of information require a Web-based document format that, like SGML, allows markup to be domain or application specific.

The main problems related to the HTML document format discussed above can be summarized as a lack of separation between structure and presentation, and a lack of support for domain-specific markup. The issues related to mixing of structural and visual markup have been primarily addressed (apart from the deprecation of visual markup in HTML 4.0) by the introduction of Cascading Style Sheets (CSS) in 1996. Support for domain-specific markup was one of the main issues addressed by the Extensible Markup Language (XML) in 1998.

3.2.2 Cascading Style Sheets (CSS)

CSS level 1 (CSS1 [156]) defines how multiple style sheets can be applied to determine the visual formatting of an HTML document. A number of extensions to CSS1 have been proposed, including support for high-quality printing, absolute positioning of HTML elements on the screen and speech-synthesis of Web documents (e.g. to read HTML pages out loud for visually impaired users). Most of these proposals have been incorporated in CSS level 2 (CSS2 [33]). HTML documents that previously needed to mix structure and presentation to achieve a certain visual effect, can now use CSS to separate these two issues. For example, in the following HTML fragment:

```
<h3 align="center">
  <font color="black">
     The Need for Style Sheets
 </font>
</h3>
```

the HTML can be simplified by removing all visual markup:

```
<h3>The Need for Style Sheets</h3>
```

where the presentation information is encoded separately, and only once for all third-level headings, by a single CSS style rule:

```
h3 { text-align: center; color: black }
```

HTML is quite flexible with respect to the location of the CSS style rules. Style rules such as the one above could be part of a separate style sheet which is explicitly referred to by the HTML document. Alternatively, when the author prefers to combine all information in a single file, style rules can be included in the style section of the heading of the HTML document. While ignoring the advantages of separate markup, the right hand side of a style rule can also be included directly as the value of a `style` attribute on the target element. Finally, style rules can be part of a style sheet which is not explicitly linked to the HTML document, but is nevertheless applied to the document by the processing software. The style rule could, for example, be part of the default style sheet of the user's browser. A key aspect of CSS is that multiple style sheets can be combined (cascaded), and that the CSS specification defines the strategy for resolving conflicting style rules.

CSS is, at the time of writing, not fully implemented by most browsers. The introduction of CSS has nevertheless been quite successful. This success is not only based on the (longer term) advantages CSS has when it comes to maintenance, reuse and longevity as sketched above. The application of CSS turned out to have some other (short term) advantages as well. First of all, CSS was, in contrast to HTML, designed from the beginning to specify the document's visual appearance. It gave authors control over the appearance of their HTML documents, even more control than they used to have with (non-standard) HTML extensions. Secondly, because all style information is localized, HTML documents employing CSS proved to be, on average, smaller than similar documents using many (redundant) inline style attributes. The smaller size resulted in a significant decrease in down-load and response times [174]. Additionally, users often request a sequence of HTML pages that share a common style sheet. Browsers can use this to further reduce the download times by caching style sheets.

3.2.3 Extensible Markup Language (XML)

While HTML 4.0 and CSS facilitate the separation of structure and presentation, they do not support the second requirement of domain-specific structuring facilities. Organizations that needed to disseminate documents with domain-specific markup over the Web, have historically approached this problem in several ways:

- **Down-conversion** — This process involves using SGML or another language that provides the appropriate domain-specific structures as the source format. By converting to standard HTML at the server side, information becomes easily accessible for all users with a mainstream Web-browser. The disadvantage is the loss of domain-specific information, which is no longer available for the client.

- **SGML-based dissemination** — The second solution is dissemination of information in the original SGML source format. This applies in particular to organizations that are already using SGML in other applications, and want to start to use the Web for the dissemination of documents which are encoded using a domain-specific document format (e.g. a format conforming to a particular industry standard). Their intended audience is typically small and is already using the required client-applications to process SGML documents. The advantage of this approach is that all structuring information is available for the client. The drawback is that is difficult to reach a larger audience, because this audience does typically not have access to the (expensive) SGML client software. Additionally, it is hard to keep up with (and benefit from) more mainstream Web software, because the market for SGML applications is so much smaller.

- **HTML-based extensions** — Many domain-specific extensions to HTML, and new document formats with an HTML-like syntax, have been developed to overcome the limitations of HTML for their particular application. This approach suffers from the same drawbacks as SGML-based dissemination: it is difficult to reach a large audience because of the special client software that is required. Additionally, ad hoc extensions are often hard to maintain and are not easily reusable in other applications.

In theory, the drawbacks of all three approaches could be overcome by making support for SGML a standard feature in mainstream Web-clients and other Web-related software, as already advocated by Sperberg-McQueen and Goldstein [213] and implemented by Hyper-G (now HyperWave) [165] in 1994. However, this solution has been regarded as overkill by many Web developers. SGML is often considered too complex to implement, too complex to use, and offering too many features that are not needed for the average Web application. Additionally, standard SGML tools are generally geared towards more traditional document types, with makes it hard to fully employ the user interaction and multimedia facilities of today's browsers (see also Chapter 8).

While an SGML-based solution has considerable drawbacks, an HTML-based solution also has major disadvantages. Uncontrolled extension of HTML would endanger interoperability on the Web. On the other hand, developing standard extensions to HTML for the wide range of applications for which HTML did not suffice, is also unlikely to succeed, and would certainly jeopardize one of the strong points of HTML: its simplicity. To overcome the limitations of both SGML and HTML, the Extensible Markup Language (XML [38]) has been developed.

XML versus HTML and SGML

XML bridges the gap between (overly complex) SGML and (overly simplistic) HTML by defining a streamlined subset of SGML. By inheriting SGML's mechanisms for defining domain and application specific markup, XML overcomes the limitations of the "one size fits all" approach of HTML. On the other hand, by removing all of the more exotic and complex features of SGML, it also overcomes the limitations associated with the traditional SGML systems. While most SGML tools are complex and expensive, free XML parser implementations are available for Java, C, C++ and most scripting languages, and XML support is already a standard part of the current generation Web-browsers.

One of the major differences between XML and SGML is the role played by the document type definition (DTD) that defines the document structure. Retrieving the DTD, parsing, validation and other DTD-related processes, are traditionally considered to be necessary steps in an SGML processing model. The fact that these steps are indeed mandatory in SGML was not considered a serious drawback, because they are relatively cheap when compared to the other steps necessary to process the large and complex documents that are typical for most SGML environments.

The same steps, however, are considered by many Web-developers as expensive and often unnecessary overhead, especially for small and relatively simple Web-documents. An important issue here is that XML has not only been developed as a means for exchanging structured documents, but also as a means for exchanging structured data over the Web, for instance in Web-based EDI applications. Especially in these data-oriented domains, the use of XML has many advantages over SGML.

Although making the DTD optional simplifies the development of XML processing software, authors of XML documents are required to assure that their documents can indeed be processed without requiring a DTD. All kinds of keystroke minimization features of SGML (allowing authors, for example, to omit tags that can be derived from the DTD) are not supported by XML. Another consequence is the need to define two levels of conformance for XML documents. At the first level, XML defines *well-formed* documents. Documents that comply to the basic XML rules are well-formed documents. Additionally, XML uses the notion of *valid* documents. Valid documents are well-formed documents, that in addition contain (a reference to) a DTD, and have a document structure that conforms to that DTD. It is expected that many Web-applications will not need to validate their documents against the DTD. For these applications, the requirements associated with well-formedness are sufficient, and these applications do not need to disseminate documents with a DTD, nor do they have to be able to parse DTDs of documents they receive. (See Appendix B for other differences between SGML and XML.)

XML and related specifications

While XML syntactically supports domain-specific markup, it does not provide a means for attaching semantics to such markup. As is the case for SGML (and, to a lesser extent, HTML) there are several other specifications, closely related to XML, that can be used in combination to specify these semantics.

In this section, we focus on the specifications needed to attach hypermedia semantics to XML syntax. These specifications are often influenced by earlier, HTML or SGML-based specifications, as depicted in Table 3.1. The first column summarizes the use of

	HTML Family	**XML Family**	**SGML Family**
Markup	HTML	XML	SGML
Presentation	CSS	CSS, XSL	DSSSL
Linking & Anchoring	HTML	XLink, XPointer	HyTime, TEI
Document model	DOM	DOM	Grove
Extensions	Ad hoc	XML Namespaces	Architectures

Table 3.1: Structured document standards on the World Wide Web.

the HTML family of Web document specifications. HTML is used to define the markup of Web documents, CSS is specifically designed to describe the presentation semantics of a document. Linking and anchoring semantics are, as discussed before, part of the HTML markup. The Document Object Model provides a standardized API to HTML documents for scripting and other applications. Application-specific extensions to the HTML syntax are typically handled in a rather ad hoc manner.

The third column contains standards that are all related to SGML. SGML is used to define the overall markup of the document, and SGML documents can additionally make use of SGML-related standards such as HyTime or TEI to define their link structure. These documents typically use DSSSL or a proprietary style sheet language to define their presentation semantics. The concept of the SGML grove provides a formal document model used by parser APIs, but also for defining link anchors and style sheet selectors. Applications can extend their document types by using the concept of an SGML *architectural form*, as defined by the HyTime standard (See also Appendix A).

For all five rows (markup, presentation, linking, document model and extensibility), the problems are basically the same. For a growing number of Web applications, the HTML solutions in the first column are too limited, while the SGML solutions in the third column are too complex. W3C is developing a set of XML-related specifications to bridge this gap between the HTML and SGML world. These specifications are listed in the middle column of the table[3], and discussed below.

XML presentation semantics As discussed above, XML itself provides only the syntactic facilities needed for defining domain-specific markup. Presentation semantics of a document are not covered by XML. An important difference between XML and HTML is that even when an HTML document uses no visual markup at all, the HTML browser is still able to render the document because it has a set of reasonable default style rules for HTML markup. This is not the case for generic XML browsers, which cannot know in advance what document types they are expected to render. XML documents with a logical structure that is similar to the presentation structure (as is the case with HTML documents) are able to use CSS style sheets to define their presentation semantics in terms of the HTML/CSS output model.

For documents that need a more complex transformation, or for presentations that go beyond the CSS output model, W3C is currently developing the Extensible Style Language (XSL [73]). XSL uses, unlike CSS and DSSSL, an XML compliant syntax to define the presentation semantics of another XML document. An XSL style sheet is thus a well-formed XML document, and can as such be processed by XML software. XSL's transformation language (XSLT [58]) is being developed to specify a transformation process that can convert XML documents into HTML, plain text or other XML documents.

XSL formatting (in DSSSL terminology: construction of a flow object tree) is defined as a transformation to a special XML document type that defines the formatting objects expected by the rendering engine. Such a set of formatting objects is called an *output vocabulary*. One of the most common conversions is expected to be the transformation of XML documents to the HTML/CSS output vocabulary. More complex vocabularies are currently under development to support presentations that go beyond the HTML/CSS presentation model.

[3]At the time of writing, the XSL, XLink and XPointer specifications are still work in progress.

XML linking semantics To provide a general link mechanism across all XML document types on the Web, the XML Linking Language (XLink [75]) is being developed. For many XML documents, HTML's linking and anchoring syntax is not sufficient, because it is based on the single <a> tag, and other XML document types should not be forced to use HTML's tag and attribute names for their linking. HyTime's link syntax was especially designed to overcome this problem. It uses the DTD to define fixed attribute values which are used to identify elements as being links, thereby renaming attributes to the proper HyTime names. However, the necessary syntax relied heavily on the existence of a DTD, and is not suitable in an XML environment where DTDs are optional.

In the current XLink draft a hybrid solution is used: XLink defines a specific XML namespace in which the (qualified) element names of the links can be used to identify a link. At the same time, XML elements with other element names can be recognized as defining an XLink by using (qualified) attributes. This allows XML applications to use XLink without the need to conform to any pre-defined element names. Note that in a DTD-less XML document, this mechanism requires several XLink attributes to be present on all link elements, a drawback for manually authored documents. In addition to the syntactical differences, XLink also provides extra hyperlink functionality, to be discussed in Section 3.3.

XLink is designed to be used in combination with the XML Pointer Language (XPointer [76]), which provides a general anchoring mechanism for XML documents. XPointer overcomes the limitations of HTML's anchoring mechanism, where every anchor needs to be encoded inline using a named <a> element. Note that pointing to an element for anchoring purposes is similar to selecting an element for applying a style rule. This is why both XSL and XPointer share a common underlying language, the XML Path Language (XPath [59]).

XML scripting and extensibility The Document Object Model (DOM [14]) provides scripts and other applications with a general standardized API to XML documents. Specific XML document types may extend the core interface defined by the DOM, tailored to the specific needs of their applications.

Despite the name, XML in itself does not provide explicit support for extending XML-document types with other, existing, document types. A major part of the problem is related to potential name clashes, which can be avoided by using the concept of XML Namespaces [37]. XML namespaces allow an application to qualify identifiers by using a prefix on tag and attribute names. The prefix can be chosen by the author of the document. The author needs to associate the prefix with the URI that serves as a globally unique identifier for the target namespace. The prefix can than be used as a convenient shorthand for the URI. A major drawback of XML namespaces is that they can only be used to avoid name clashes. Validation, for example, is not supported: documents combining markup from more than one DTD cannot be (automatically) validated, even if they use XML namespaces. Note that the concept of an "architectural form" as defined by HyTime has also been used to combine document type definitions in a single document. This approach, however, depends heavily on DTDs, and is thus not a suitable option for many XML applications.

Implications for HTML

Since HTML's syntax is based on SGML, the differences between XML and SGML will also have implications for HTML. HTML's syntax will need some minor modifications to make it conform to XML. Additionally, for many XML applications it would be useful to build upon the markup already provided by HTML. An XML document containing table definitions, for example, could benefit from reusing HTML's table model. Syntactic conformance [232] and modularization [233] are the two main reasons for the development of the Extensible Hypertext Markup Language (XHTML). XHTML basically provides the same functionality as HTML 4.0, reformulated as a set of XML namespaces, to facilitate reuse by other XML document types. Modularization of HTML also allows alternative browser platforms that cannot support the complete HTML 4.0 specification (as is common for PDAs or mobile phones) to explicitly define which subset is supported. Protocols for explicitly describing profiles of the platform capabilities and user preferences are currently under development [178].

3.2.4 Summary

Since the introduction of CSS style sheets and XML structured markup, many of problems related to HTML are overcome, and the basic requirements for Web-based structured markup are met. To realize a full document processing environment, however, these techniques are not sufficient, and a complete family of XML related specifications is currently under development. In the next section, where we focus on the Web's hypertext functionality, we will further discuss two of these XML-related specifications: XLink and XPointer.

3.3 Hyperlinking on the Web

In the previous section we evaluated HTML in terms of requirements related to structured document processing, and pointed out how style sheets and XML can be used to overcome some major drawbacks associated with HTML. In this section, we evaluate the functionality of the Web from a more hypertext-oriented perspective.

3.3.1 Requirements

The hypertext research issues presented in the previous chapter give a good indication of the typical features hypertext researchers expect from a hypertext system. This applies particularly to Halasz's seven issues, Engelbart's open hypermedia requirements, and the Dexter model, which all have been discussed in Section 2.3. We will use these issues for an evaluation of the Web from a hypertext point of view. To simplify referencing in the discussion below, Halasz's seven issues and Engelbart's twelve requirements have been summarized in Table 3.2 and Table 3.3[4].

H1	Search and Query	H5	Versioning
H2	Composition	H6	CSCW
H3	Virtual structures	H7	Extensibility & tailorability
H4	Computation		

Table 3.2: Halasz's 7 issues [112].

This section combines the requirements of Halasz, Engelbart and the Dexter model into four categories, which are used to evaluate the hypertext functionality of the Web in terms of its underlying node/link model, information retrieval facilities, support for computer supported collaborative work, and integration and interfacing functionality.

E1	Mixed object documents	E7	Hyperdocument mail
E2	Explicitly structured documents	E8	Personal Signature Encryption
E3	View control of object's form, sequence and content	E9	Access Control
E4	The basic "hyperdocument"	E10	Link addresses that are readable and interpretable by humans
E5	Hyperdocument "Back-Link" capability	E11	Every object addressable
E6	The hyperdocument "library system"	E12	Hard-copy print options to show address of objects and address specification of links

Table 3.3: Engelbart's 12 requirements for hypertext systems [96].

The node/link model

The Web's node/link model has been frequently subjected to critique in the hypertext literature. Before we discuss the Web's link model, we first evaluate the typical "node", i.e. the HTML Web page. When evaluated against the more node-oriented requirements, the typical Web page scores surprisingly well.

Web pages meet, for example, to a large extent the first four requirements listed in Table 3.3. Since the addition of inline images to HTML and the use of MIME (Multipurpose Internet Mail Extensions [32]) as the common envelope technique, mixed-media documents (requirement E1) have become the *de facto* standard on the Web. Being based on document formats such as HTML, and later XML, the Web also supports explicitly

[4]For a discussion of the Web with respect to Engelbart's requirements, see also [63].

structured documents (requirement E2). Additionally, with the advent of WYSIWYG HTML editors and style sheets the user has direct control over the appearance of his documents (requirement E3). The use of hyperlinks has become very common on the Web, meeting Engelbart's basic hyperdocument requirement (requirement E4). Many link addresses on the Web are human readable (requirement 10), and many organizations nowadays even use the address of their Web site in off-line advertisements. Requirement E12 is, at the time of writing, not commonly supported but could be easily added as an extra print option in most Web browsers.

Web pages also meet Halasz's requirement for virtual structures, i.e. structures that are created at runtime, which do not necessarily require explicit authoring or persistent storage (requirement H3). On the Web, generated pages have become common practice, especially since the introduction of the Common Gateway Interface (CGI [61]). CGI not only standardizes the interface between the HTTP-server and the application generating the page, but also encodes queries using URIs, an approach that facilitates a closer integration of query and navigation based interfaces.

While explicitly modeled by Dexter, composition is a feature that is still not adequately supported on the Web. Combining several separate HTML documents into one larger document is accomplished by using referential hypertext links for composition purposes (typically using "next", "previous" or "top" anchors). Even after the introduction of composition based on HTML's frame concept, users still suffer from the often counterintuitive difference in behavior between the document's "previous" link and the browser's "back" button. This problem is precisely the problem Halasz addressed in issue H2, where he discussed the lack of an adequate composition model in NoteCards. On the Web, the W3C is currently looking into alternatives for the current practice of using referential links or frames for composition. HTML 4 has, for example, a recommended list of typed links of which several have explicit composition semantics. These link types are, however, not widely used or supported.

While the Web's node model may meet most of the requirements hypertext researchers have, this does not apply to its link model. Even when compared to the hypertext systems of the eighties, the Web's link functionality is, in many aspects, very limited. Typical hyperlink features that HTML lacks include:

- **Bi-directional links** — In several hypertext link models, including the Dexter model, links have a bi-directional nature. This makes it, for example, very easy to find links referring to a particular node (Engelbart's back-link, requirement E5), a task which is virtually impossible on the Web. While a general (scalable) model for bi-directional links cannot be realized in open and distributed systems as the Web, such links can be effectively defined among specific sets of documents (e.g. among a set of related documents on an organization's intranet).

- **Multiple source/destination links** — This type of link is also supported by the Dexter model, and is particularly useful when links are used to model semantic relations that are not necessarily binary. But even for navigational purposes, this type of link can be useful, for instance to define menu-like navigation structures (using multiple destination links) or for navigation structures where multiple locations in the document can be used to traverse to the same destination (using multiple source links).

- **Virtual linking** — As discussed above, links whose destination component is defined as the result of a (CGI) script have become commonplace on the Web. But many other virtual link structures are not supported. In a system supporting bidirectional links, an obvious extension allows the (address of) the source components of the link to be defined by a query (as supported by the Dexter model). An even more common hypertext concept is the *virtual anchor*. Virtual anchors are not statically encoded but defined by a query that is evaluated at runtime. While virtual anchors are not explicitly modeled by the Dexter model, both virtual anchors and virtual links have been a characterizing feature of open hypermedia systems in general, and the Microcosm system [69] in particular.

- **Transclusion and stretchtext** — Hyperlinking is used as a flexible means for inclusion of material into a document by hypertext systems such as Guide and Xanadu. Such use of linking is however, not supported on the Web. Note that it is neither supported by most other hypertext systems, nor is it explicitly modeled by the Dexter model. Using links for this purpose has many advantages though, because it provides an intuitive model for interactive inclusion (as provided by Guide's replace buttons) and allows easy reuse of the link's anchoring mechanism to include fragments of one document into another one. In Xanadu, links are even the primary means of composition in the concept of the transclusion mechanism.

Note that several features listed above require links to be encoded out-of-line, a feature which is commonly supported in (open) hypertext systems, but not on the Web. In the next section, we discuss the out-of-line links proposed by XLink, which enables a Web-based realization of many of the features listed above.

Information retrieval

The hypertext literature has been significantly influenced by research on information retrieval. The need for a query-based interface in addition to the purely navigational link-based interface is identified by issue H1[5]. Halasz discriminated *content search* from *structure search*. A third type of searching that has recently become fashionable lies somewhere in between content and structure search and is based on annotations associated with the document, better known as *metadata*.

- **Content search** — For textual components, content search can be efficiently implemented by employing standard (distributed) database and information retrieval techniques. For components containing other media, content-based retrieval is

[5] In later work [115] Halasz notes that query-based navigation can not only be seen as complementing traditional link-based navigation, but as a true alternative to traditional link-based access.

much more difficult. Typically, approaches range from manually annotating multimedia data along with text-based query interfaces, to advanced content-based image retrieval techniques in combination with query-by-example interfaces [100].

Facilities for text-based content-search are now an indispensable part of the Web's infrastructure. Efficient support for global searches on the Web, with its vast, dynamically changing and distributed document base, has become a favorite research issue, and has even become the core business of several companies. In addition to global search facilities, many Web-sites provide a local query interface to the information on their site, and, on the component level, a function to search the content of the current page is part of virtually every browser. HTML's *forms* provide markup to embed query interfaces in Web documents and have a been standard feature since HTML 2.0 [22]. On the server side, protocols such as the Common Gateway Interface (CGI [61]) have standardized the interface between the HTTP-server and the application carrying out the search.

- **Metadata search** — Improving upon the results of content-based retrieval is an important objective underlying several initiatives to add *metadata* to Web pages. Examples include RDF [155], which aims at providing a general metadata framework, and the Dublin Core [55], which standardizes a relatively small set of generally applicable metadata attributes. Metadata formats also play an important role in annotation and retrieval of multimedia data, the main topic of standards such as MPEG-7 [138].

- **Structure search** — In addition to referring to the content, queries in a structure search are allowed to refer explicitly to the structure of a document or the link structure of a set of documents. In a system with typed hyperlinks, for example, one could search for information supporting a claim in component A by searching for all components with a "supports" link to A. Unlike content search, structure search is still quite rare on the Web. The design of an appropriate query language — combining content and structure search in a single query, the design of an effective user interface and an efficient implementation of the search engine proves to be non-trivial. Some search engines, however, make use of the link structure as a basis for ranking the results of a content query [107]. Additionally, with the Web's untyped links and ill-structured HTML pages, structure-based searching is less attractive. Structure-based queries are likely to be more relevant for document formats that employ richer markup schemes and for data-oriented XML applications. HyTime defined HyQ as a structured query langage [45, 46], and query languages for XML are currently under development [77]. Being able to query on the structure of a given document is also necessary for effectively defining link anchors or style sheet rules. This relationship is further discussed in Section 3.3.2 on XML linking below.

Note that all three query mechanisms can be used in two directions: to find relevant information, but also to filter out — or censor — irrelevant or unwanted information. This gives the subject an interesting political dimension, which explains the large amount of discussion around the content selection protocol for the Web, the Platform for Internet Content Selection (PICS [168]). PICS provides a general framework for rating services, where the user typically selects a (trusted) third party to rate the documents of a content provider.

A specific application for which effective information retrieval is essential is a digital library (requirement E6). A digital library is a general service that catalogs and archives documents and guarantees long-term access to these documents. Apart from providing good retrieval facilities, a Web-based digital library has to address issues related to longevity. Proposals that explicitly address the problems related to obsolete URLs include the URN as an alternative to the URL (see page 75) and Web sites offering indirect, but persistent URLs [237]. Digital libraries have recently grown into an active research topic in their own right [4].

Computer supported collaborative work

For early hypertext visionaries, such as Bush, Engelbart and Nelson, support for collaboration was an important objective of a hypertext system. While the Web is currently mainly focused on (read-only) browsing, the need for collaborative writing support increases and has become the topic of several new standardization efforts. Note that issues related to CSCW have not been addressed by the Dexter model, although Halasz did explicitly list CSCW as issue H6. While CSCW can also be regarded as a research topic in its own right, specific aspects that have been covered in the hypertext literature include collaborative annotation, access control and versioning, and document authentication.

- **Annotation** — The ability to annotate the work of other people — even if you have no write-access to their documents, and the ability to share these annotations with other users are important features of many hypertext systems. It was also one of Berners-Lee's early requirements for the World Wide Web [26], and even (partially) supported by early Web browsers, including Mosaic [152]. Full support for collaborative annotation requires out-of-line link encoding, which is not yet supported in HTML. In addition, annotation requires a more flexible anchor mechanism. Currently, both source and target elements of a link need explicit HTML markup to indicate that these elements are being used as an anchor. This makes it hard to link into a specific fragment of a document, unless its author has anticipated the use of the fragment as a link anchor.

- **Access control and versioning** — An essential feature of a collaborative environment is the ability to control access to shared documents (requirement E9). Access control should prevent both unauthorized access and overwrite problems due to simultaneous edits etc. On the Web, access control is typically addressed on the HTTP level. HTTP's basic user name/password authentication scheme, for example, is currently supported by most browsers and servers. Since it was considered insufficiently robust, it was extended with a (slightly) more advanced digest-based mechanism in HTTP 1.1 [99, 101]. More extensive HTTP-level support for (asynchronous) collaborative authoring is currently developed by IETF's WebDAV (Word Wide Web Distributed Authoring and Versioning) working group [85]. At the time of writing, WebDAV's features include: overwrite prevention, based on shared and exclusive write locks; a general mechanism for associating metadata, based on URI/XML name/value pairs and compatible with related efforts such as RDF and Dublin Core; and collection support, dividing the server's realm into different namespaces, with direct (local) and referential (remote) containment relationships. Planned extensions include version management (issue H5) and authentication-based access control.

- **Digital signatures** — Another type of authentication support is the ability to prove the authenticity of documents (requirement E8). On the Web, signing documents is not yet common practice, although techniques such as Pretty Good Privacy [105] are sometimes used to sign Web pages. Protocols for signing PICS labels have already been standardized [128], and currently the joint W3C/IETF XML-Signature working group [110] is focusing on a more general, XML-based, mechanism for signing pages and other resources on the Web [18].

Hypertext integration and interfacing

The integration and interface of hypertext systems with the rest of the user's environment is another frequently recurring theme in the hypertext literature. This applies in particular to ubiquitous linking, integrated computation and system extensibility.

- **Ubiquitous linking** — Early hypertext researchers envisioned hyperlink functionality that was not only available in a specific hypertext application (e.g. a browser), but in all (desktop) applications, so users could create and traverse hyperlinks from one document to another, irrespective of the document's origin (requirement E11). Based on the observation that extending the wide range of existing applications with hypermedia functionality by re-implementation was not feasible, open hypermedia systems were designed in order to add hypermedia functionality to these applications without requiring re-implementation of these systems (see also Chapter 7, page 169).

  While a general link service as provided by open hypermedia systems has not become commonly adopted, Web-based hyperlinking has become a standard feature in many applications other than the typical Web-browser. HTML is still gaining popularity, not only as a document language for Web-pages, but also as a language for encoding email or Usenet news messages. Hyperlinking is now a common feature in most mail applications (requirement E7). Additionally, many desktop applications support URI-based links and are able to convert their native file format to HTML. While less sophisticated as many hypertext researchers would have liked, ubiquitous support for hyperlinking is almost reality today.

- **Integrated computation** — Another way of integrating hypertext and the user's general computing environment is by embedding executable code into hypertext documents. Halasz already proposed to support (embedded) executable code to perform computations over the contents of a hypertext system, and the NoteCards system allowed users to embed fragments of (Lisp) code, which were to be executed at runtime (issue H4, see also [112]).

On the Web, many forms of integrated computation are used, both at the client and server side. At the client, the most common examples are applets and scripting. Applets are small programs, usually written in Java, of which the byte-code can be referenced from within a Web-page. During their execution, applets can make use of the browser's window real estate. Due to security reasons, many features available to ordinary programs (such as file and network IO) are not, or only in a very restricted form, available to applets. Scripts are commonly applied to directly manipulate the presentation of an HTML document, often to provide more interactive behavior.

At the server-side, CGI-scripts are used to provide a Web-based interface to databases and other applications. CGI-scripts are executed on behalf of the server, but run as separate processes. While this is often a disadvantage in terms of performance, it is a significant advantage in terms of security, because errors in the CGI script do not crash the server. In contrast to CGI-scripts, servlets are small, trusted programs that are, for performance reasons, executed within the process space of the server. The softbots used by search engines to index (part of) the Web can also be seen as an example of server-side computation.

- **Extensibility** — Halasz notes that the generic nature of the hypertext node link model is both a blessing and a curse, and this also applies to the page/link model of the Web. Users can use the generic primitives offered by the Web for any application they consider suitable, and can impose domain specific semantics on these primitives at will. Unless systems can be made aware of these semantics, generic systems are often less suitable for domain-specific tasks than special purpose applications. To overcome this limitation, Halasz advocates systems that are designed to be extensible and tailorable by the user (issue H7), using the Emacs editor as an example [112].

  Web clients initially addressed extensibility by allowing users to define helper applications for those document formats the browser could not handle itself. This model lacked the necessary integration between browser and helper application, and was later refined by a plug-in model, which allowed extensions to make use of the browser's window real estate, and provided in addition a more sophisticated API between the plug-in and the browser.

  Work on document-level extensibility within W3C is currently focused on XML. As discussed in the previous section, many of the outstanding issues relate to the way the semantics (including presentation, linking and other semantics) of an XML syntax is to be communicated to the processing application. Below, we focus on the issues of defining link semantics for XML, and illustrate how many of the link-related issues discussed in this section are currently addressed by the XLink and XPointer proposals.

3.3.2 XML Linking and Anchoring (XLink/XPointer)

In the previous section, we briefly discussed XLink in the context of HTML's link syntax (see page 85). HTML linking is primarily based on a specific element (`<a>`), while in an XML environment, other elements could also be links. In addition to the new syntax for HTML-like links (termed *simple link* in XLink), XLink also provides functional extensions that go beyond HTML linking. The most substantial extension from the perspective of hypertext research is XLink's *extended link*. Unlike simple links, extended links are not necessarily embedded within the source resource. Additionally, each extended link element can contain an arbitrary number of end-point elements, which are called *locators* in XLink terminology. Extended links can thus be used to encode both out-of-line and multi-ended links on the Web. Link directionality, as in the links of the Dexter model, can be specified by associating *arc* elements with the extended link. Arcs explicitly label two locators as a "to" or a "from", respectively. By associating two arcs with the same pair of locators, an explicit "bi-direct" link can also be encoded. Since any XML element can be identified as a link endpoint (see below), an out-of-line link can itself be the target of another link[6].

Processing out-of-line links

Out-of-line links require a different process model when compared to HTML's inline links. Since links associated with a particular document could be defined separately — by a set of extended link definitions encoded in other Web resources — the application processing the document needs to be made aware of the existence and location of these resources. In other words, out-of-line links require a mechanism to define an explicit relation between a document and the resources containing the relevant out-of-line link definitions. XLink defines such relations by introducing a special kind of extended link (called an *extended link group*) which contains a list of special locators (called an *extended link document*) to define the resources containing the out-of-line links. These resources can be chained, as depicted in Figure 3.2. XLink defining resources can, using the same mechanism, point to other resources containing other links, etc. See Figure 3.2 for an example. Because this chain could be of arbitrary length, the originating document may impose a limit on the length of the chain by defining a special attribute on the extended link group.

[6] When an inline link is the target of another link, there is no way to determine whether it is indeed the link that is being linked to, or the source anchor of that link.

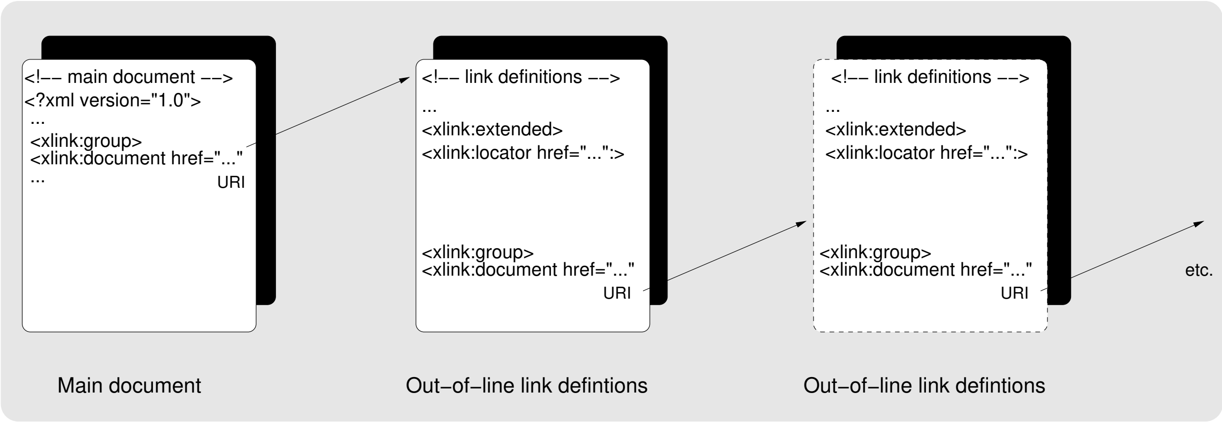

Figure 3.2: XLink chaining: Documents can refer to other documents that contains relevant links. These documents can refer to other documents, etc.

Navigation and other semantics

In the previous chapter, we explained the distinction between presentation-independent links with rich semantics, versus navigation or otherwise presentation-oriented hyperlinks (cf. page 44). XLink does not make such a distinction. Traversal behavior, presentation and other semantics can be associated with all of XML link elements by using a set of predefined attributes. Authors may specify whether link traversal will present the destination in a new window, in the current window (replacing the current presentation) or embedded in the current presentation, possibly replacing the source anchor (e.g. as in stretchtext links). Additionally, links may indicate that they should be automatically traversed by the application during the parsing process (without requiring user interaction), which potentially allows inclusion of document fragments by using anchoring.

Other presentation issues, such as the way anchors are presented, or the way links are drawn in a site map, are delegated to a style sheet language. In general, however, the interaction between XLink and a document's style sheet is not yet completely clear. A fundamental issue, for example, is the mapping from the links and anchors, which are defined by XLink in terms of the source document, to the link markers (i.e. the associated "hot spots") in the presentation. Even in CSS, the style sheet may choose, for example, not to display a specific anchor, or to render parts of the same anchor on different areas of the screen. Document transformations (using XSLT), scripting behavior (using the DOM) and time-based synchronization (using SMIL) may further complicate the mapping from document to presentation-level linking.

Anchoring

XLink only specifies the relation among Web resources — the specification of the resource's address or name is outside XLink's scope and delegated to the URI. In Dexter terms, this means that the URI contains both the specification of the target component and the anchor. Within the URI, the anchor is defined using the fragment identifier. In the context of HTML, the only valid fragment identifiers are the values of the `name` attribute of the anchor elements. XHTML is already more flexible and allows the values of `id` attributes on any element, so that arbitrary elements can be used as the endpoint of a link. The only limitation is that the author is responsible for adding `id` attributes to elements that are candidates for linking.

With the XML Pointer Language (XPointer [76]), any XML element (or text-span) can serve as the end-point of link, even if this was not anticipated by the original author of the document. In addition, it allows the definition of virtual anchors as discussed above. XPointer has been strongly influenced by the concept of extended pointers as used in the Text Encoding Initiative (TEI) and by the HyTime location module.

To point to fragments of the document which are not given an explicit identifier, XPointer provides mechanisms that use the XML tree structure to point into a specific portion of the document. Note that such anchors, especially if used by extended links defined in other documents, could easily become invalid after modifications to the document tree. Explicit identifiers are, in general, more robust, and should therefore be used if available.

Fragment identifiers are a function of the MIME type of the resource they apply to. XPointer can be seen as a general extension of HTML's fragment identifier syntax for XML documents. Although linking into non-XML document formats is outside the scope of XPointer, these formats are free to define their own fragment identifier syntax and associated semantics.

3.3.3 Summary

When compared to many of its hypertext predecessors, the Web still lacks many hyperlink features. In addition, the Web is still largely read-only, focused on (distributed) browsing instead of (distributed) editing. We discussed a number of new specifications that facilitate better distributed and collaborative editing and support typical link-related requirements such as support for inclusion relations, stretchtext and multi-headed links. Examples of such facilities include the anchoring mechanism's provided by XPointer, the out-of-line link facilities of XLink, and the HTTP extensions defined by WebDAV.

Note that all these specifications are, at the time of writing, still under development. It is too early to predict whether these will be as widely adopted as XML itself. Another point of concern is the fact that most of these efforts are focused on XML, and little attention is being paid to support linking and (especially) anchoring into non-XML documents. XLink and XPointer do not, for example, explicitly support image maps, a navigation feature that is commonly used in HTML and SMIL. Neither do they provide support for changing the base URL for links with relative URLs, another frequently used feature in HTML and SMIL.

In the next section, we look at the Web from a multimedia perspective, focusing on issues related to synchronization and multimedia layout.

3.4 Multimedia on the Web

In the previous sections we discussed the use of style sheets and XML markup in the context of the limitations of HTML as a structured document format. Additionally, we sketched some ongoing work on hyperlinking in XML, and the extent to which languages such as XLink and XPointer address HTML's hypertext-related problems. In this section, we take a look at the Web from a multimedia perspective, focusing on document formats supporting temporal synchronization and multimedia layout.

3.4.1 Requirements

In Chapter 2, we discussed typical multimedia requirements, including fast hardware, high-bandwidth networks and real-time protocols with QoS support and synchronization primitives. Since most end-users have platforms that are perfectly capable to run multimedia applications from CD-ROM, faster hardware at the client-side is no longer a real bottleneck for distributing multimedia over the Web. Although network performance still affects most distributed multimedia applications, we consider network issues beyond the scope of this thesis. Instead, we evaluate the document models that are in use on the Web in terms of the multimedia requirements discussed in Chapter 2: temporal synchronization, content adaptation and multimedia layout.

Temporal synchronization

HTML's approach to multimedia is still similar to Dexter's, in that it allows links to, and even inclusion of, non-textual media items, but does not support synchronization of these media items. There have been several approaches to adding document-level synchronization functionality to the Web. We sketch the three basic approaches: developing new multimedia document formats, extending HTML and extending CSS.

The first approach is to use a special-purpose multimedia document format as an alternative. Unfortunately, these formats are often in a platform-specific, proprietary and binary format. This not only threatens the Web's platform-independence and interoperability, it also makes it hard to integrate these documents into existing Web-technology. (For example, it is hard to index these documents using existing search engines.) To better integrate multimedia content into the Web, W3C developed the Synchronized Multimedia Integration Language (SMIL). SMIL is an open, platform independent multimedia language, with a text-based XML syntax. SMIL is discussed in Section 3.4.2 below.

Since SMIL is specifically developed for synchronized multimedia presentations, it is based on a document model that is radically different from HTML's text-flow model. SMIL 1.0 does not provide a straightforward mechanism for adding timing to existing HTML documents. Such a mechanism would be of particular interest for applications that are primarily based on static media such as text and graphics, but which need in addition a set of basic primitives to synchronize and/or animate the presentation of the text and other media objects. A typical example is a slide show presentation. While this type of presentation might not need a fully-fledged multimedia synchronization toolkit, it would certainly benefit from the standard text-based formatting primitives that are provided by HTML.

Therefore, a second approach is based on adding temporal synchronization facilities to HTML. Temporal extensions to HTML have been realized in several ways. A procedural solution that is currently in use is to employ scripting technology. Using scripts to specify the temporal behavior of HTML documents has the advantage that it does not require any extensions to the HTML syntax, and that the document's behavior can be programmatically controlled. On the other hand, it has all the disadvantages associated with scripts in terms of programming effort required from the author, maintenance, etc. A more declarative technique is to define the timing by adding an extra set of elements and attributes to the current (X)HTML specification. This is, for example, the basis of the HTML+TIME proposal [204], which adds a set of new elements and attributes to HTML. The additions can be used to define the temporal behavior of an HTML document in terms of the SMIL timing model.

A third approach models timing as a presentation property, to be specified in a style sheet. While such style sheets can also be used to add temporal behavior to HTML documents, it has the advantage that it is both a declarative solution and keeps HTML simple and free of timing-related details. It does, however, require an extension of the output model of the style sheet language[7]. Another problem is that a major advantage of style sheets — their reusability — only applies to a very small extent to style sheets defining timing information. In most cases, timing specifications are a unique aspect of a single document.

Content adaptation

As argued previously, the Web was initially built around the functionality of the browser, which offered a single, uniform interface to a wide variety of information services. This approach has several limitations. Terminals currently used to access the Web range from high-end workstations and PCs to small PDAs and mobile phones, all featuring very different rendering capabilities and interface requirements. In addition, the network resources available will vary from user to user, due to differences between network carriers (e.g. fixed-line versus mobile network), differences in modem technology, etc. With the forthcoming second and third generation Internet, these differences are likely to become even larger. Finally, there are many differences among the end-user's abilities and preferences that need to be taken into account. This variety requires protocols that are no longer tailored to a single browser environment, but are able to adapt to different situations. Such protocols need to explicitly address issues such as quality of service (QoS) specification and content negotiation.

Most Web-related research regarding QoS and content negotiation on the Web is car-

[7]CSS2 only supports some very basic timing in the context of aural style sheets, which are used in speech synthesized presentations of Web-pages.

ried out at the network (TCP/IP) level or at the transport (HTTP) level, which has the advantage that solutions are often transparent from a document-level perspective. In practice, however, lower-level approaches cannot always bridge the large differences sketched above, in which case it becomes necessary to give the author some control over the way the document adapts to a new situation. This requires document-level techniques such as those discussed in Chapter 2. A good example is the support for different output media in CSS. Within a single CSS style sheet, an author may specify a different set of style rules for each medium the document is to be rendered on. Another example (discussed in further detail below) is SMIL's *switch* mechanism which specifies alternatives for the media items used in the presentation, and attaches the necessary metadata to provide information about the resources needed for these alternatives. Note that such a mechanism could also be useful in (X)HTML and other XML document types, and is for example, likely to become part of W3C's forthcoming markup language for Scalable Vector Graphics (SVG [98]).

Multimedia spatial layout

The visual layout model of both HTML and CSS is built around a hierarchical, two-dimensional box-model which is similar to the models which are often used for multimedia layout. However, many of the notions that are built on top of this model do not apply to multimedia layout, while important multimedia layout features are missing. Examples of typographic concepts, that are fundamental in CSS but do not necessarily apply to multimedia layout, include: inline and block display, scrollable viewports, pagination, and floats. Important multimedia layout features that are missing are the definitions of the coordinates of media items relative to the coordinates of another media item[8] and non-textual box filling strategies (such as resizing with or without preservation of aspect-ratios, clipping/cropping etc.).

An additional problem is caused by inheritance of visual style properties. Inheritance in CSS is defined in terms of the document hierarchy, and not in terms of the hierarchy of the presentation structure [33]. This is problematic for multimedia documents where the document structure is used to model the temporal hierarchy (which is the case for many multimedia formats, including the SMIL language discussed below). Inheritance of visual style properties along a temporal hierarchy can yield unexpected and unwanted results. For these documents, it is often more appropriate to limit this type of inheritance to temporal properties. Inheritance of visual properties is then more appropriately defined in terms of the document's spatial layout hierarchy. In the Amsterdam Hypermedia Model (Chapter 5), for example, visual style attributes are inherited along the channel hierarchy.

Many of the issues discussed above are illustrated in the next section by discussing SMIL, W3C's Synchronized Multimedia Integration Language.

[8] CSS also uses the notion "relative positioning", but with different semantics. Relative in CSS means "relative to the object's default position in the text-flow". Positioning relative to another object is currently not supported by CSS.

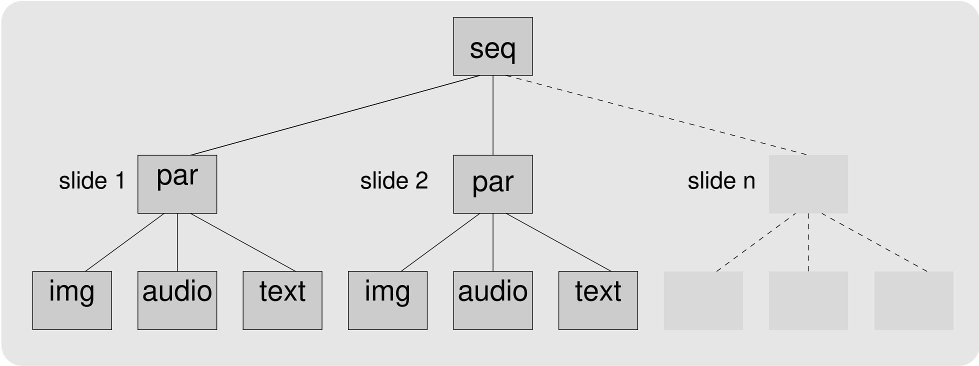

Figure 3.3: Example of a temporal hierarchy in SMIL.

3.4.2 Synchronized Multimedia Integration Language (SMIL)

To support synchronized multimedia on the Web, W3C developed the Synchronized Multimedia Integration Language (SMIL [231], pronounced "smile"). SMIL is a document format defined with an XML-based syntax, and allows integrated presentation of hyperlinked multimedia objects over the Web. Though it is expected that most SMIL documents will be generated using dedicated editors, SMIL documents can be edited by an ordinary text editor because of the text-based XML syntax.

SMIL relies on existing transport protocols such as HTTP and RTSP to retrieve media items from potentially different servers. SMIL also relies on existing media formats to describe the content of the individual media objects. Note that the SMIL specification does not require applications to support any specific media type or media format. While this allows SMIL authors to use those media formats they find to be most appropriate, it also limits the interoperability between various SMIL implementations.

The SMIL documents are built using four basic ingredients: temporal composition and synchronization, spatial layout, content adaptation and hyperlinking.

Temporal composition and synchronization

SMIL's top level structure is comparable with HTML in that it contains a `head` and a `body` section. The hierarchical structure of the body reflects, unlike HTML, the temporal structure of the presentation. SMIL's temporal model is based on the structure-based temporal models discussed in Chapter 2 (see page 53). SMIL presentations are developed by building a tree of nested parallel and sequential composition elements. The leaf-nodes of this tree contain references to Web-resources containing the actual media data. Figure 3.3, for example, illustrates how a simple slide show can be hierarchically modeled using parallel and sequential composition. The root of the tree contains a sequence of slides (the figure displays only the first two slides). Each slide is represented by a parallel composite, containing three media items: the image containing the slide's image data, and an accompanying audio and text item. Using SMIL's XML based syntax, this could be encoded as follows (the `region` attributes refer to the spatial layout definitions described below):

```
<seq>
 <par id="slide1">
  <img   src="map.gif"  region="slides-img"/>
  <audio src="map.au"/>
  <text  src="map.html" region="slide-text"/>
 </par>
 <par id="slide2">
  <img  src="palace.gif"  region="slides-img"/>
  <audio src="palace.au"/>
  <text  src="palace.html" region="slide-text"/>
 </par>
 <!-- ... slide n -->
</seq>
```

Note that there are no explicit start times, end times or durations defined. The presentation will use the defaults derived from the hierarchical structure. In this particular case, only the audio files have their own intrinsic duration, so the duration of each parallel composite will be equal to the contained audio fragment. As a result, both the accompanying text and image will be presented for the same amount of time, and end when their direct parent node ends. This coarse-grained synchronization, inferred from the parallel and sequential composition elements, can be refined by synchronization attributes. SMIL specifies attributes to delay starting times (`begin`), explicitly set an element's end time or duration (`end`, `dur`), synchronize an element's end time to another element (`endsync`) and to repeat an element (`repeat`). Larger modifications (e.g. reordering the slides, or showing two slides in parallel) will require authors to change the document structure.

Hyperlinking

An obvious extension to the slide show example is to add hyperlinks to the slides so the user can navigate from one slide to another, and from the slide show to other Web-resources.

On a syntactic level, hyperlinking in SMIL is comparable with hyperlinking in HTML. The main addition to HTML's `<a>` tag is the addition of the `show` attribute[9], with the values `replace`, `new` and `pause`. On traversal of a `show=replace` link, the current presentation is stopped and replaced by the destination presentation. A `show=new` link starts playing the destination in a new context (e.g. a new window), not affecting the current presentation. Finally, a `show=pause` plays the destination in a new context, while pausing the current presentation. Just as HTML, SMIL 1.0 only allows simple embedded links. Out-of-line or multi-headed links as defined by XLink are not supported.

[9] The attribute name and values of `show` anticipated the publication of the XLink specification.

In addition to the <a> tag, the SMIL `anchor`[10] can be used to define links from specific portions of media objects in a way similar to HTML's image maps. The `anchor` element not only defines the spatial extent of the anchor but also its temporal extent. This allows for anchors that are active for a specific period. The anchor's temporal extent is defined using the same synchronization attributes SMIL defines for defining the temporal behavior on media elements (e.g. `begin`, `end`, `dur`).

Note that in general, links in a multimedia document will interact with the document's temporal behavior (in the example, link traversal will interfere with the timing implied by the sequential composition of the slides). SMIL 1.0 defines the effect of link traversal to a specific media element to be a "seek" (that is, a fast-forward or rewind operation) to the begin time of that specific element. Note that if the user does not initiate link traversal, the slide show will be presented as before. SMIL 1.0 does not provide an alternative for the sequential composite that could define a container element that could be used to group elements without any predefined timing. Future versions of SMIL are expected to have such a container in the form of the *exclusive* element (`excl` [17]), which behaves not unlike the *choice* composite of the AHM. In the example above, replacing the sequential composite by an exclusive would allow the start and end of all slides to be triggered by hyperlink navigation or other events originating from user interactions, media streams, scripts etc.

Spatial layout

The spatial layout model of SMIL is consistent with the visual box model of CSS2 [231]. In theory, the spatial layout of a SMIL presentation can be defined by any style sheet language that has a sufficiently powerful layout mechanism, including CSS2. For reasons of interoperability, however, the SMIL specification requires implementations to implement at least one, minimal, layout mechanism, called SMIL basic layout. SMIL documents usually contain a relatively large set of media objects, which are rendered on a relatively small number of regions on the screen. Therefore, SMIL layout allows authors to define those regions once (in the layout section in the header), and then refer to these regions multiple times (from the media objects in the body). Note that this is exactly the other way round from most style languages, where the layout definitions in the style sheet refer to the elements defined in the document.

Web-based multimedia presentations often need to be presented on different platforms, so it is useful to be able to use different size windows. Additionally, a resize operation carried out by the user's window system should typically not lead to a reflow of text (as is the case in HTML browsers), but should resize the individual regions and their associated media items when appropriate. SMIL supports this kind of behavior and allows regions to be specified relative to the total size of the top-level window. For example, a layout for the slide show could be to use an initial 400x400 pixels wide top-level window, reserve 10 percent of the window for the slide text, and use 80 percent for the images, leaving room for a small (5 percent) margin. This could be defined

[10] The `anchor` element will be aligned with HTML's `area` element in the next version of SMIL.

in SMIL as:

```
<smil>
  <head>
    <layout type="text/smil-basic-layout">
      <root-layout width="400" height="400"/>
      <region id="slides-img"
         left="5%" top="5%"  width="90%" height="80%"/>
      <region id="slide-text"
         left="5%" top="85%" width="90%" height="10%"/>
    </layout>
  </head>
  <body>...
```

Note that the intrinsic size of the media elements does not need to match with the size of the regions. How the SMIL player should compensate for such differences can be specified using the region's `fit` attribute, which can be used to specify a range of common reconciliation strategies including filling, resizing with respect to aspect ratios, cropping and scrolling. To support rendering of media objects that (partially) overlay other media objects, the region's relative depth can be specified using the `z-index` attribute, that has semantics identical to the CSS property with the same name.

Content adaptation

To refer to the actual media content, SMIL has a set of predefined tags for the most common media elements, such as `img`, `audio`, `video`, `text`, etc. These tags are, however, merely provided for authoring convenience. Implementations are expected to determine the MIME type of the media object using the mechanisms commonly employed by HTML browsers (that is, using the information in the HTTP header, filename extension, explicit type attributes on the elements, etc).

For each media object or composite, alternatives can be specified by using SMIL's `switch` element. A SMIL application selects an alternative by evaluating a set of test attributes, which describe platform-dependent attributes such as screen size and available network bandwidth, or user preferences such as the preferred language. For example, in the slide show above we could have provided alternatives for the text and audio objects in another language, or alternatives for the audio and image objects with a lower sample rate and resolution for users that do not have a sufficiently fast network connection.

The switch element could also be used to provide alternative media types, e.g. we could also provide text alternatives for the audio and image objects. SMIL's synchronization facilities and the switch element play, for instance, an important role in developing accessible documents [189, 190]. Online "talking books", for example, have been developed for the visually impaired [218]. In such applications, SMIL is typically used to synchronize the display of HTML text with the spoken version of the same material.

Accessibility issues significantly influenced SMIL design, which is reflected in many attributes with text descriptions on all relevant elements, and the support for providing alternative media content.

While the switch element gives the author a large amount of control over the behavior of a document in different situations, it is in many aspects still a rather low level mechanism. Especially in larger documents, providing the necessary alternatives may result in an explosion of the number of variations in the switch element, and for each element, the right test attributes need to provided. Additionally, authors should take into account that different media elements in a switch may all have a different effect on the time structure. In the slide show example above, all timing is essentially based on the intrinsic durations of the audio fragments. Unless more explicit timing information is added, replacement of these fragments by text would result in unwanted timing behavior.

Relation to other Web specifications

SMIL 1.0 has been developed as a first step in integrating time-based multimedia into the architecture of the World Wide Web. Together with streaming protocols such as RTSP, it provides the basic primitives needed to play synchronized multimedia presentations over the Web. Many features are still missing, including the more advanced hypermedia features discussed in Section 2.5. Future versions of SMIL, or extensions to SMIL, will need to address these problems.

An important requirement is a better integration of SMIL's synchronization functionality into document formats other than SMIL. Text-oriented applications, such as slide shows, will benefit from (light-weight) time-based extensions to languages such as (X)HTML. Animated vector graphics, for example, require timing support in SVG, a topic addressed by SMIL Animation [203]. In these applications, integration at the syntax level (which can be handled using XML Namespaces) is insufficient: timing also needs to be integrated in the presentation model of the languages involved.

3.4.3 Summary

Initially, the Web was built on three basic protocols: HTML, HTTP and URLs. These three protocols have been the basis of the success of the Web, and will continue to be useful for a large number of applications. Despite the demonstrated success of these protocols, they also have some significant drawbacks. In this chapter, we focused on a number of document-level requirements, evaluating the HTML document model from different perspectives. In addition, we sketched a new generation of Web protocols and XML-based document formats.

We explained how the introduction of style sheets by means of CSS fulfilled a major electronic publishing requirement: separation of structure and presentation in HTML documents. Another important requirement, support for domain-specific markup, is satisfied on a syntactical level by the introduction of XML.

We also evaluated HTML from a hypertext perspective, and looked at applications that require advanced hyperlinking and anchoring that goes beyond simple one-way, inline encoded link of HTML's anchor tag. We discussed the multi-headed, out-of-line encoded linking proposed by XLink, and the more advanced anchoring mechanisms of XPointer.

Finally, we discussed some ways to extend the Web with multimedia functionality, focusing on temporal synchronization, adaptation and multimedia spatial layout. We illustrated these aspects of Web-based multimedia by discussing SMIL and some HTML-based alternatives.

3.5 Conclusion of Part I

In the first part of this thesis, we looked at modeling structured documents from the perspective of four different research areas: electronic publishing, hypertext, multimedia and the Web. We explained the basic terminology and concepts used in the electronic publishing literature on structured documents. Until now, the areas where structured documents have been applied successfully are mainly text-oriented applications. We identified the needs for extending the text-oriented models upon which these application are based. Important use cases include:

- Text-based applications that are currently taking advantage of the multiple delivery processing model, but need to be extended with Web-based hypermedia functionality.
- Web-based hypermedia applications that need to adapt to a variety of interfaces. The initial Web model, based on a single browser and uniform interface, can no longer accommodate the needs of different Web users, with different abilities, network access and terminals. Style sheets and structured XML documents address the need for more tailorable and adaptive documents. Additionally, this approach has the potential of protecting documents against rapidly changing technology.

Both cases require a better integration of structured documents, hypertext and multimedia technology into the current Web infrastructure.

We discussed the major differences between text-based, hypertext and multimedia documents, and the consequences these differences have on a structured hypermedia document model. For hypermedia document processing, the separation of content, structure and layout found in many text processing systems cannot be easily applied to hypermedia processing systems. First of all, multimedia documents may contain certain spatio-temporal relations that are considered to be an integral part of the document structure, and are as such not supposed to vary between different presentations of the document. Secondly, the well established logical and page description models for text documents are not suitable for describing the spatio-temporal layout of media items in hypermedia documents. Finally, many of the existing models do not account for the interactive behavior of such documents, because they neither model user interaction (from simple hyperlink traversal to the full interaction provided by an application embedded in the document) nor system interaction (such as QoS negotiation or dynamic adaptation to a changing system environment).

These issues all play an important role in the protocols and specifications that are being developed for use on the World Wide Web. From an electronic publishing perspective, the introduction of XML and style sheet technology has successfully addressed many of the problems related to HTML. Despite XML's extensible syntax, many outstanding issues need to be resolved, before a full integration of hyperlinking and multimedia functionality in XML can be obtained. Some of the issues discussed are currently being addressed in the ongoing work on XML linking, synchronized multimedia, Web-based collaborative work and many other related research topics.

Difficulties in developing hypermedia document models lie in the need to deal with a broad range of hypermedia applications, and with rapidly changing technology. The protocols and interfaces associated with these models need, to a large extent, to be mutually independent, but still allow applications to use several of these interfaces in a consistent manner. In addition, the models need to be sufficiently specific to be useful for standardization and interoperability purposes, and at the same time be sufficiently abstract to adapt to new requirements and changes in the future.

To be able to evaluate whether a hypermedia model provides the right functionality and is expressed in terms of the right abstractions, these models need to be defined in an unambiguous manner. The "light-weight" use of formal methods in the following part allows us to unambiguously define the concepts underlying the models discussed in the previous chapters. It enables the specification of these concepts at the right level of abstraction, and allows us to precisely identify the issues that need to be addressed in the interfaces of the architectures that implement these systems.

Part II

Hypermedia Modeling

Chapter 4

The Dexter Hypertext Reference Model

This part of the thesis provides a more precise treatment of the most relevant hypermedia models discussed so far. In particular, we revisit the Dexter hypertext reference model and the Amsterdam hypermedia model from a more formal perspective. In addition, we sketch a document transformation model, and explore the modeling of the temporal behavior of hypermedia presentations at runtime.

Formal specification of the behavior of a hypermedia system facilitates the definition of standards that enable hypermedia document interchange and other forms of interoperability. It also provides a basis for comparing the way documents are processed, and we will use the models described in this part to differentiate between the different hypermedia architectures discussed in Part III of the thesis.

This chapter is built around a formal specification of the Dexter model. The specification is used to formalize many issues discussed in the first part of the thesis. Furthermore, it enables a more incremental approach to the specification of the Amsterdam Hypermedia Model, of which a formal specification is given in Chapter 5. The formal aspects of hypermedia style sheets and document transformations are discussed in Chapter 6.

4.1 Introduction

The Dexter model was originally formalized using the Z specification language, but this specification [113] has never been widely published, and has thus never been accessible by a broader audience.

This chapter provides an alternative specification of the Dexter model using Object-Z. Apart from the correction of some minor flaws in the original Z specification, the Object-Z specification given here is functionally equivalent to its counterpart in Z. When compared to the original specification, however, it is both shorter and easier to understand. Additionally, the use of Object-Z facilitates an incremental approach to the specification of the Amsterdam hypermedia model and the document transformation model in the following chapters.

The main body of this chapter is devoted to the formal specification of the two upper layers of the Dexter model. We discuss the cases where the formal text deviates from the original Z specification, and highlight the parts of the Dexter model that are relevant to the research issues discussed in the first part of the thesis.

4.1.1 About Object-Z

Object-Z [80, 81, 82] is an extension to Z [214] to support formal specification in an object-oriented style [154]. An object-oriented specification describes a system as a collection of interacting objects, each of which has a prescribed structure and behavior.

A Z specification defines a number of state and operation schemas. It is the responsibility of the author of the specification to make clear which state and operation schema's belong together, for instance using informal prose or adequate layout techniques. In Object-Z, a state schema and the schemas operating on that state are grouped together forming a *class*.

A class is, however, more than a group of related schemas: it also behaves as a template for objects: each object of a class has a state conforming to the class's state schema and is subject to state transitions which conform to the class's operations. Classes can further be related by inheritance and can also be used as a type: instances of that type are identities which reference objects of that class. This allows objects to refer to other objects.

Additionally, classes can constrain their behavior by specifying an optional *history invariant*. The history invariant is a predicate over histories of objects of the class expressed in temporal logic. The most commonly used temporal operators are $\Box$ (always), $\Diamond$ (eventually) and $\bigcirc$ (next).

Appendix C gives an brief explanation of the Object-Z notations used.

4.1.2 Overview of the Dexter model

The Dexter hypertext reference model [113, 114] was developed to provide a principled basis for comparing hypertext systems as well as for developing interchange and interoperability standards. The model is the result of various meetings — the first one was in 1988 at the Dexter Inn — of experienced hypermedia system designers.

The model is divided into three layers (see Figure 4.1 on the facing page): the storage, runtime and within-component layer. The storage layer is the middle layer that describes the network of nodes and links that is the essence of hypertext. Dexter uses the more neutral term *component* for all variations of nodes, including links. The runtime layer describes the mechanics supporting user interaction. The within-component layer covers the content and structures of the data content within the components. The elaboration of the within-component layer is considered beyond the scope of the Dexter model, and is thus not formalized in this chapter. The focus of the model is on the storage layer (formalized in Section 4.2), and the runtime layer (formalized in Section 4.3).

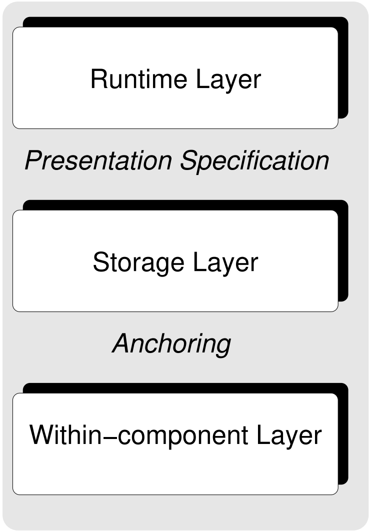

Figure 4.1: The three layers of the Dexter model.

The notion of *anchoring* is introduced to describe the main interface mechanism between the within-component layer and the storage layer. Anchoring is used to be able to address locations or items within components, without knowledge of their inner structure. This extra level of indirection allows a description of hyperlinks which is independent of the structure of the media items at the end points of the link. At the time the Dexter model was developed, not all systems allowed this media independent definition of hyperlinks. The concept of anchoring, and the clear distinction between anchoring and linking, is regarded as one of the major contributions of the Dexter model.

The interface between the storage layer and the runtime layer is accomplished using the notion of *presentation specifications*. Presentation specifications are a generic mechanism for modeling information about how a component is to be presented to the user. Presentation specifications can be stored as part of the component in the storage layer, but can also be associated with a component at runtime by the application. Since presentation specifications are typically application and/or media dependent, Dexter makes no attempt to model their inner structure.

Note that the use of presentation specifications is to a large extent comparable with the use of CSS on the Web, where style rules can be an intrinsic part of the Web page, but can also be assigned at runtime by the user application.

The following two sections will further discuss and formalize the Dexter storage layer and the runtime layer.

4.2 The Storage Layer

The storage layer focuses on the mechanisms enabling the data-containing and other components to be interconnected by link components. Figure 4.2 on the next page de-

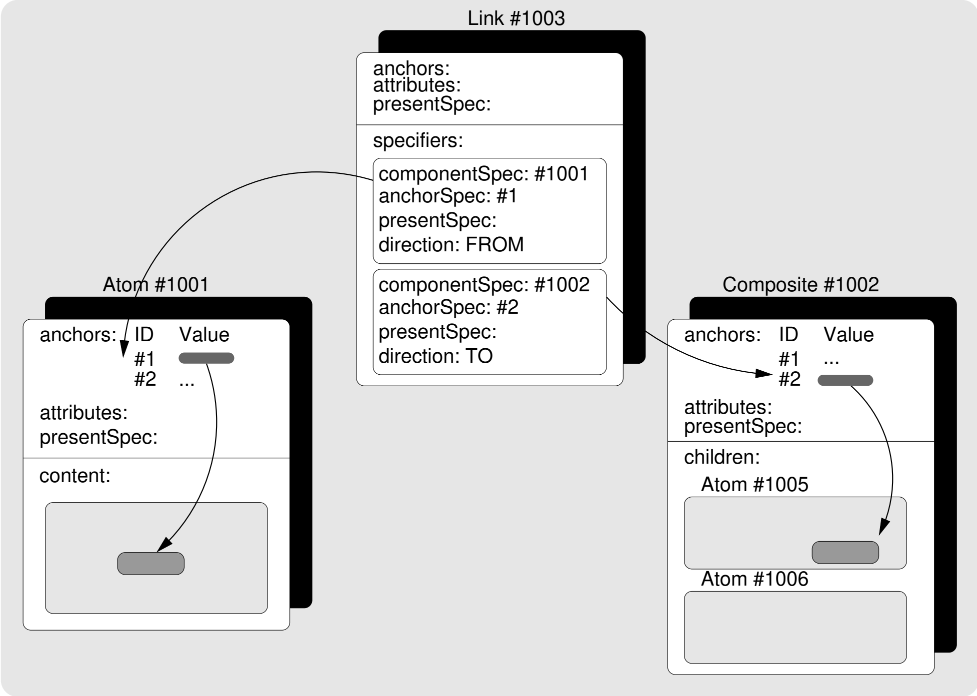

Figure 4.2: Dexter linking from an atomic to a composite component.

picts the three types of components of the Dexter model: atoms, links and composites.

Atomic components are generic containers of data. *Composite* components are recursive structures constructed out of other components. The composite in the figure contains (references to) two other components, which happen to be two atomic components. The *Link* component models relations between other components. Links contain a sequence of *endpoint specifiers* — the link in the figure is a binary link that contains one specifier for the source of the link and one for the destination. These specifiers point to (part of) a component by specifying the associated anchor identifier and a component specification.

The addressing of components involves a two-step process. First, a given component specification (e.g. a query) is mapped to the unique identifier (Uid) of the component matching the specification. This mapping is modeled by the *resolver* function. Second, the Uid is mapped to the component itself by the *accessor function*. The explicit use of the accessor and resolver function was introduced to address the issue of support for search and query facilities, and the issue of support for virtual structures, as advocated by Halasz ([112], see also page 36).

A *Hypertext* is modeled as a set of atomic, link and composite components with an associated accessor and resolver function. All these concepts will be formalized in the remainder of this section, using the bottom-up approach implied by the "declare before use" conventions of the Object-Z notation.

4.2.1 Identifiers and Anchors

We start by defining the lower level building blocks needed by all components: component identifiers and hyperlink anchors. Two sets from which identifiers can be chosen are introduced. A set of unique identifiers *Uid* is needed to identify the components of the hypertext. Additionally, we need a set of anchor identifiers *AnchorId* to identify an anchor within a given component.

$$[Uid, AnchorId]$$

Note that the constraint, stating that anchor identifiers have to be unique for a given component, is typically easy to satisfy. Ensuring uniqueness for *Uids* is much harder, especially in open systems such as the Web. We return to this issue when discussing the *accessor* function, which maps *Uids* to their associated components.

All anchors in Dexter are divided in two parts. The specific fragment of the component which plays the role of the anchor is specified in an anchor value, from the given set *AnchorValue*.

$$[AnchorValue]$$

Anchors consist of a media-independent anchor identifier and an associated value, which is media dependent.

$$Anchor == AnchorId \times AnchorValue$$

The notion of anchoring is regarded as one of the major contributions of the Dexter model. Although the clear distinction between links and anchors has been lost in HTML, it is again clearly visible in the current XLink and XPointer specifications. The anchor also plays distinct roles in each of the three layers.

In the storage layer, only the *AnchorId* plays a prominent role, since links (which will be defined in Section 4.2.3) can only refer to anchors by means of the *AnchorId*. This implies that all anchors need to be explicitly present at the level of the storage layer. It is, for example, not possible to use an anchor that is embedded in tagged video data in the within-component layer directly. Instead, a storage level anchor needs to be created, of which the *AnchorValue* can be used to refer to the tag name in the video. The link can then use the *AnchorId* to refer to the specific anchor.

The obvious advantage of this indirection is that it isolates the link from the lower-level details needed to interpreted the *AnchorValue*. This represents the role of the anchor in the within-component layer, and is typically media-specific. Its value can consist of a character range, a particular shape specified by a set of coordinates, a range of video frames, etc. It is interpretable only by the end-user application, and not by the "hypertext engine".

The application will typically use the *AnchorValue* to determine the characteristics of the "link marker", the object that represents the anchor at runtime (the link marker is defined in Section 4.3). It models the associated "sensitive region" or "hot-spot" in the user interface. Note that the term "link marker" was also used in this way in Brown's Intermedia system [167].

4.2.2 Specifiers and Specifications

Below, we define three similar sounding, but functionally very distinct concepts: component specifications, presentation specifications and (link end) specifiers.

Component specifications To address the need for a query interface in addition to the traditional navigation interface, Dexter introduces component specifications to model search and query operations. Instead of using a component's Uid directly, component specifications (i.e. queries) are used to refer to a component. Later, we will define the *resolver* function which is used to map a component specification to a Uid. Component specifications are modeled by the basic type *ComponentSpec*.

Presentation specifications The runtime layer uses presentation specifications from the basic type *PresentSpec*. Presentation specifications model all layout and style information needed to present a component. While they are heavily used, their inner structure is not constrained by the Dexter model. The Amsterdam hypermedia model described in Chapter 5 provides a more detailed definition of the inner structure of presentation specifications.

$$[\mathit{ComponentSpec}, \mathit{PresentSpec}]$$

Specifiers Not to be confused with the component and presentation specifications, a *specifier* describes a single link end. The *Direction* type is used to model a link-end specifier as source, destination, both or as neither.

$$\mathit{Direction} ::= \mathit{FROM} \mid \mathit{TO} \mid \mathit{BIDIRECT} \mid \mathit{NONE}$$

The last option seems to be included for historical reasons, without a clear practical use. In other models, *NONE* is sometimes used to model invalid link ends explicitly [108] or to attach meta-information to anchors. Both uses are, however, explicitly forbidden by Dexter because of its strict linking constraints (discussed below). A link-end specifier contains a *direction* which intially has the value *NONE*. In addition it contains a reference to the target component, anchor and a presentation specification.

Specifier

$$
\begin{array}{l}
\mathit{componentSpec} : \mathit{ComponentSpec}; \\
\mathit{anchorSpec} : \mathit{AnchorId}; \\
\mathit{presentSpec} : \mathit{PresentSpec}; \\
\mathit{direction} : \mathit{Direction}
\end{array}
$$

INIT

$$\mathit{direction} = \mathit{NONE}$$

Note the use of a component specification (*componentSpec*) instead of a Uid to refer to the target component. This allows for so called "virtual links", hyperlinks whose destination or source component will not be resolved until runtime. The Uid of the destination can be the result of a database query, and can be dependent on information only available at runtime (e.g. to specify a link to the most frequently visited component). The same indirection is, however, *not* used for anchors. Instead, anchor identifiers are used directly by the specifier (*anchorSpec*). This makes it hard to model more indirect anchor specifications, as used by the *local link* and *generic link* of Microcosm [69] or the anchors specified by XPointer as discussed in the previous chapter.

In the context of the Web's client/server model, the distinction between the component specifier (specified by the first part of a URI) and the anchor (the fragment identifier in the latter part of the URI) allows for an effective separation of concerns. The client passes the component specifier directly to the server, which is responsible for its interpretation. Within the boundaries of the URI syntax specifications, it may contain any query that the server is able to resolve. It allows the server to use arbitrary search engines and associated query interfaces, independent from the capabilities of the client. In contrast, the anchor is never sent to the server, which gives the client the freedom to implement new fragment identifier encodings such as XPointer. Note that fragment identifiers are comparable with Dexter's anchor values to the extent that they are media-dependent and thus generally only interpretable by the client, and not by a generic link processing application.

In Dexter, the presentation of a component can be dependent on the link that was used to reach that component, so every link-end specifier includes a presentation specification (*presentSpec*). Traditionally, the hypertext community has adopted a broad interpretation of the term presentation specification. In [114], for example, the presentation specification is used to model security issues by including "access presentation information" into the link specifiers.

While Halasz already mentioned the potential problems of linking into composites, the Dexter link end does not explicitly address this issue. In the case that a link end refers to a component that is part of a composite, the link end does not specify to what extent the composite is considered to be part of the link end. Although this information could be modeled implicitly by using the presentation specification, we model the notion of link context more explicitly in the specification of the AHM in the next chapter.

4.2.3 Components

An central role in the storage layer is played by the *Component*, which provides an abstract base class from which the three actual component types are derived: the *Atom*, *Link*, and *Composite* components. The fact that the specification of the Dexter model is more concise in Object-Z than in plain Z is, to a large extent, the result of the significantly more compact and straightforward specification of the *Component* class and its subclasses. Readers familiar with the Dexter model will note that, due to the use of inheritance, the artificial distinction between the Dexter concepts COMPONENT, BASE_COMPONENT and COMP_INFO is no longer necessary in the Object-Z version. The same applies to the plethora of schemas that were necessary in the plain Z version to ensure type consistency of the various components.

Note that, in contrast to links, which are modeled as "first class" components, an anchor is modeled as part of the component it is associated with. In addition to the initially empty list of anchors, all components have a presentation specification and, optionally, a number of attribute/value pairs to store ancillary information.

$[Attribute, Value]$

Component

$$anchors : \mathrm{seq}\, Anchor$$
$$presentSpec : PresentSpec$$
$$attributes : Attribute \rightarrowtail\!\!\!\!+ Value$$

$$\#anchors = \#(first(\!|\ \mathrm{ran}\, anchors\ |\!))$$

INIT

$$attributes = \varnothing \land anchors = \langle\,\rangle$$

The invariant states that the number of anchors ($\#anchors$) must be equal to the number of different anchor identifiers ($\#(first(\!|\ \mathrm{ran}\, anchors\ |\!))$). This invariant is needed to ensure that anchor identifiers are unique within a component. In the model, we will never use an instance of *Component* directly, since all components will be instances of one of the three derived classes. We will use the Object-Z notation $c : \downarrow Component$ to define c to be an instance of class *Component* or an instance of a class derived from *Component*.

Atomic component

The original Dexter specification is ambiguous when it defines the way atomic components are to be included in a composite component. The informal text and the explanatory figures ([113], page 10) allow both atomic components and direct content to be part of the composite, while the formal specification allows only the latter. We further discuss this ambiguity when defining the composite component.

In the specification given here, we have chosen to make a clear separation between the roles of the three component types: link and composite components now have a purely structural role, that is, links and composites do not contain any content information directly. All the content information stored in the hypertext is ultimately modeled by atomic components. The *Atom* is derived from *Component*, from which it inherits information shared by all components: anchors, attributes and a presentation specification. In addition, it contains a *content* variable which represents the data. These contents are modeled by the given set *Atomic* and have no internal detail from the viewpoint of the storage layer.

$$[Atomic]$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Atom\ }}\\
Component\\
\quad content : Atomic\\
\hline
\end{array}
$$

Link component

A link is a so-called "first class object": it is a descendant of *Component* and as such addressable just as any other component. This allows links to links to be modeled in the same way as links to other components. Links inherit anchors, attributes and a presentation specification from the *Component* base class. The presentation specification typically models the presentation of the link itself, which can be, for example, useful in systems that display the link in a navigation map. Presentation of the anchor or the component the link refers to can be modeled by the presentation specification in the link-end specifier.

In addition to the information it inherits from component, a link consists of a sequence of link-end specifiers. This allows the modeling of multi-headed and multi-source links.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Link\ }}\\
Component\\
\quad specifiers : \operatorname{seq} Specifier\\
\quad \Delta\\
\quad componentSpecs : \mathbb{F}\, ComponentSpec;\\
\quad anchorSpecs : \mathbb{F}\, AnchorId\\
\hline
\quad \# specifiers \geq 2\\
\quad \exists s : \operatorname{ran} specifiers \bullet s.direction \in \{TO, BIDIRECT\}\\
\quad componentSpecs = \{cs : ComponentSpec \mid\\
\qquad \exists s : \operatorname{ran} specifiers \bullet cs = s.componentSpec\}\\
\quad anchorSpecs = \{as : AnchorId \mid\\
\qquad \exists s : \operatorname{ran} specifiers \bullet as = s.anchorSpec\}\\
\hline
\end{array}
$$

All links should have at least two specifiers ($\# specifiers \geq 2$), and at least one destination. The latter invariant is ensured by stating that there should be at least one specifier with a *TO* or *BIDIRECT* direction ($\exists s : \operatorname{ran} specifiers \bullet s.direction \in \{TO, BIDIRECT\}$). Dexter is severely criticized for its strict link correctness invariants. For instance, it is reasonable to allow incomplete links, with less than two end points and no destination, during the authoring phase. Note that XLink also allows (extended) links with only one link end [75]. To overcome these problems, we simply drop both preconditions in the specification of the AHM in the next chapter.

Object-Z has the notion of *secondary variables*, which are preceded by a Δ in the declaration part of the state schema. These variables are secondary in the sence that their values are fully determined by the value of the primary variables. They add no new information and are only used to increase the readability of the specification. Note that normally, the Δ is used in operation schemas to indicate what state variables are modified by the operation. Because the value of a secondary variable is fully determined by other variables, there is no need to include them in the in the Δ-list of operation schemas [80].

To enhance readability of classes that need to query the anchors and components of the endpoints of a link, the *Link* class has two secondary variables, the finite sets *componentSpecs* and *anchorSpecs*. Their values are defined in terms of the primary *specifiers* variable by the two last constraints.

Composite component

Halasz's composition issue (see page 37) is addressed in the Dexter model by providing an explicit composition structure in addition to link-based structuring. The *composite component* is characterized by its children attribute, which is (recursively) defined as a sequence of components. Additionally, the secondary variable *descendants* is defined by the transitive closure of the *subcomp* relation. It is used to constrain the composition structure to a directed acyclic graph (DAG) by stating that no component may contain itself as a subcomponent: $self \notin descendants$.

Composite

Component

$subcomp : \downarrow Composite \leftrightarrow \downarrow Component$

$\forall c_1 : Composite;\ c_2 : \downarrow Component \bullet$
$\quad c_1 \ \underline{subcomp}\ c_2 \Leftrightarrow c_2 \in \operatorname{ran} c_1.children$

$children : \operatorname{seq} \downarrow Component$
Δ
$descendants : \mathbb{F} \downarrow Component$

$descendants = \{c : \downarrow Component \mid (self \mapsto c) \in subcomp^{+}\}$
$self \notin descendants$

INIT

$children = \langle\, \rangle$

An advantage of Object-Z is that recursive types can be described without using a free type definition, which would have been needed in plain Z. The use of recursive class schemas is described in more detail in [81].

The semantics of the recursive definition given above differs on two, related, aspects from the functionality of the original Dexter composite specification. The first difference is an extension of the original model, which is a direct result from the fact that the children of our composite component inherit from the abstract *Component* class. This means that each child can have its own anchors, attributes and presentation specification. In the original formal specification, this information can only be associated with the root element of a composition hierarchy. The original model does not mention this omission, and the informal text and figures even suggest that the composite can include both "complete" components (that is, components that include anchors, attributes and a presentation specification) as its children, as well as "base" components (that is, components without anchors, attributes or presentation specification). This brings us to the second difference, which is not an extension but a restriction to the original Dexter model and was already mentioned when defining the atomic component. In our model, composites can only have other components as their children, and cannot directly include raw content.

Note that while the specification given here allows anchor definitions at all levels of the composition hierarchy, it does *not* address the critique of Grønbæck and Trigg on Dexter's underspecified anchor values [108]. For example, the model does not specify how to construct composite anchors by grouping anchors defined by the children of a composite, a construct that is explicitly modeled by the AHM described in the next chapter.

Dexter's composition mechanism does not have any associated semantics. As discussed before, Dexter does not model the effect of composition on link traversal, nor does it explicitly model the presentation of composites, which is an import aspect of multimedia systems. In addition, the flexible way to refer to content by using component specifications and anchors as used by Dexter's links, is not used to model composition. This prevents a straightforward means of modeling "virtual" composites.

Dexter also fails to model an adequate integration of composition and linking, that is needed to describe systems supporting stretchtext (Guide) or transclusions (Xanadu)[1].

4.2.4 The Hypertext Class

The main concept of the storage layer is the *Hypertext* class. While a hypertext only consists of a set of *components* and an associated *resolver* and *accessor* function, it is still rather complex due the many constraints that ensure an "intuitively" sound system. The constraints ensure composability of the resolver and accessor function, accessibility of the components and link and anchor consistency. They are discussed in more detail below.

[1]Neither the developers of Guide nor those of Xanadu where involved in the development of the Dexter model.

In addition to the hypertext's state and initialization schema, there are three schemas that model operations to modify a hypertext: *addComponent*, *deleteComponent* and *modifyComponent*. Additionally, two inspection operations are provided which can be used to determine the set of links resolving to a specific component (*linksToComponent*) and the set of links connected to a specific anchor (*linksToAnchor*).

Hypertext

$$
\begin{array}{l}
components : \mathbb{F} \downarrow Component \\
resolver : ComponentSpec \rightarrowtail\!\!\!\!\!\!\!+\; Uid \\
accessor : Uid \rightarrowtail\!\!\!\!\!\!+\; \downarrow Component \\
\hline
\operatorname{ran} resolver = \operatorname{dom} accessor \\
\forall c : components \bullet c \in \operatorname{ran} accessor \\
\forall l : Link \mid l \in components \bullet (\forall s : l.componentSpecs \bullet \\
\quad (\exists c : components \bullet (accessor \circ resolver)(s) = c)) \\
\forall c : components \bullet \\
\quad \exists s_1, s_2 : \mathbb{F} AnchorId;\ lids! : \mathbb{F} Uid;\ cuid : Uid \mid c = accessor(cuid) \bullet \\
\qquad linksToComponent[cuid/uid?] \wedge \\
\qquad s_1 = first (\!| \operatorname{ran} c.anchors |\!) \wedge \\
\qquad s_2 = \{aid : AnchorId \mid \exists lid : lids!;\ link : Link \bullet \\
\qquad\qquad link = accessor(lid) \wedge aid \in link.anchorSpecs\} \wedge \\
\qquad s_1 = s_2
\end{array}
$$

INIT

$$
components = \varnothing \wedge resolver = \varnothing \wedge accessor = \varnothing
$$

addComponent

$$
\begin{array}{l}
\Delta(components, resolver, accessor) \\
c? : \downarrow Component \\
\hline
components' = components \cup \{c?\} \\
\exists_1\, uid : Uid \bullet (\exists cs : ComponentSpec \bullet \\
\quad accessor' = accessor \cup \{uid \mapsto c?\} \wedge \\
\quad resolver' = resolver \cup \{cs \mapsto uid\})
\end{array}
$$

$$
\begin{array}{l}
\textit{deleteComponent} \\
\hline
\Delta(\textit{components}, \textit{resolver}, \textit{accessor}) \\
\textit{uid}? : \textit{Uid} \\
\hline
\exists\, \textit{lids}! : \mathbb{F}\, \textit{Uid} \bullet \\
\textit{linksToComponent} \wedge \\
(\exists\, \textit{uids} : \mathbb{F}\, \textit{Uid} \mid \textit{uids} = \{\textit{uid}?\} \cup \textit{lids}! \bullet \\
\textit{components}' = \textit{components} \setminus \textit{accessor}(\!|\ \textit{uids}\ |\!) \wedge \\
\textit{accessor}' = \textit{uids} \mathbin{\lhd\!\!\!-} \textit{accessor} \wedge \\
\textit{resolver}' = \textit{resolver} \mathbin{-\!\!\!\rhd} \textit{uids})
\end{array}
$$

$$
\begin{array}{l}
\textit{modifyComponent} \\
\hline
\Delta(\textit{components}, \textit{accessor}) \\
\textit{uid}? : \textit{Uid} \\
\textit{new}? : \downarrow\!\textit{Component} \\
\hline
\exists\, \textit{old} : \textit{components} \bullet \textit{old} = \textit{accessor}(\textit{uid}?) \wedge \\
\qquad \textit{components}' = \textit{components} \setminus \{\textit{old}\} \cup \{\textit{new}?\} \\
\textit{accessor}' = \textit{accessor} \oplus \{\textit{uid}? \mapsto \textit{new}?\}
\end{array}
$$

$$
\begin{array}{l}
\textit{linksToComponent} \\
\hline
\textit{uid}? : \textit{Uid} \\
\textit{lids}! : \mathbb{F}\, \textit{Uid} \\
\hline
\textit{lids}! = \{\textit{lid} : \textit{Uid} \mid (\exists\, \textit{link} : \textit{Link} \mid \textit{link} \in \textit{components} \bullet \\
\qquad \textit{lid} = \textit{accessor}^{\sim}(\textit{link}) \wedge \\
\qquad (\exists\, s : \textit{link.componentSpecs} \bullet \textit{uid}? = \textit{resolver}(s)))\}
\end{array}
$$

$$
\begin{array}{l}
\textit{linksToAnchor} \\
\hline
\textit{uid}? : \textit{Uid} \\
\textit{aid}? : \textit{AnchorId} \\
\textit{links}! : \mathbb{F}\, \textit{Link} \\
\hline
\exists\, \textit{lids}! : \mathbb{F}\, \textit{Uid} \bullet \\
\textit{linksToComponent} \wedge \textit{links}! = \\
\qquad \{\textit{link} : \textit{Link} \mid \textit{link} \in \textit{accessor}(\!|\ \textit{lids}!\ |\!) \wedge \textit{aid}? \in \textit{link.anchorSpecs}\}
\end{array}
$$

Composability of the resolver and accessor The first constraint of the state schema is straightforward. It ensures that all the Uids from the resolver's range may be used to retrieve a valid document with the accessor function: $\operatorname{ran} \textit{resolver} = \operatorname{dom} \textit{accessor}$. Note that $\operatorname{ran} \textit{resolver} \supseteq \operatorname{dom} \textit{accessor}$ would allow the existence of component specifications that are mapped correctly to a *Uid* by the resolver, but cannot be mapped to a *Component* by the accessor. On the other hand, $\operatorname{ran} \textit{resolver} \subseteq \operatorname{dom} \textit{accessor}$ allows the existence of components which do have an associated *Uid*, but no associated component specification which maps to that *Uid*. This would make the document inaccessible in practice, which is not allowed in the Dexter model.

A more questionable constraint arises from the fact that Dexter models the resolver as a partial function ($\nrightarrow$). The problem lies not in the fact that it is a *partial* function: it would indeed make no sense to model it as a *total* function, since not all component specifications will be resolvable. The problem lies, however, in modeling the resolver as a mathematical *function*, and not as a *relation*: the resolver function enforces a query to resolve into exactly one single Uid, a highly unrealistic constraint for a general hypertext system.

Accessibility of the components All of the hypertext's components should be accessible by means of the accessor function, which is ensured by $\forall\, c : components \bullet c \in \operatorname{ran} accessor$. The accessor is modeled as a partial injective function ($\rightarrowtail\!\!\!\!\!\!+\,$). It is a *partial* function, because not all Uids are mapped on a component: some Uids will remain unused. It is an *injective* function because every component has only one Uid. It also ensures that the accessor has an inverse function($accessor^{\sim}(c)$), a property that is used in the next constraint. Note that in most of the second generation hypertext systems upon which the Dexter model is based, the functionality of a resolver function could be implemented by maintaining a central table mapping Uids to components. In open, distributed systems such as the Web, a different approach is needed. In this respect, the issue of assigning and resolving globally unique IDs is closely related to the URL versus URN discussion on page 75.

Link consistency The remaining constraints are more complex. The third constraint is needed to ensure consistency of the hyperlinks.

$$\begin{aligned}&\forall\, l : Link \mid l \in components \bullet (\forall\, s : l.componentSpecs \bullet \\ &\quad (\exists\, c : components \bullet (accessor \circ resolver)(s) = c))\end{aligned}$$

Literally, it states that for every link that is a member of *components*, all of the link's component specifiers should resolve to an existing component. The constraint ensures that all links must resolve to an existing component within the system. In this way, dangling links are explicitly excluded. This constraint is another example of Dexter's (over)emphasis on link consistency: most hypertext systems are more flexible, especially during the authoring phase. Note that in open, distributed hypertext systems such as the Web, links can also point to components outside the system. So in these systems, this constraint can never be realized.

Anchor consistency The last constraint makes sure that all anchors used by the link ends are defined by the proper component. In addition, it prevents the existence of anchors that are not used by any link end, which seems overly restrictive. Literally, it states that for all components, the set of anchor identifiers (s_1) is equal to the set of anchor identifiers of the component specifiers of the links resolving to that component (s_2).

$$\begin{array}{l}
\forall\, c : components \bullet \\
\quad \exists\, s_1, s_2 : \mathbb{F}\, AnchorId;\ lids! : \mathbb{F}\, Uid;\ cuid : Uid \mid c = accessor(cuid) \bullet \\
\qquad linksToComponent[cuid/uid?] \land \\
\qquad s_1 = first(\!|\ \mathrm{ran}\, c.anchors\ |\!) \land \\
\qquad s_2 = \{aid : AnchorId \mid \exists\, lid : lids!;\ link : Link \bullet \\
\qquad\qquad link = accessor(lid) \land aid \in link.anchorSpecs\} \land \\
\qquad s_1 = s_2
\end{array}$$

Note that the value of *lids*!, the set of UIDs of the links that resolve to the target component *c*, is bounded by using the linksToComponent schema. Schema renaming (*SchemaName*[*new*/*old*]) is needed to set up the proper context for the *linksToComponent* schema.

Operations on the hypertext In addition to the constraints, the following operations are defined to modify or inspect the state of a *Hypertext*:

- *addComponent* — Adding a component requires a unique ($\exists_1$ *uid*) identifier to properly extend the *accessor* function. The resolver function is also extended so that there is at least one component specification for the new uid.

- *deleteComponent* — Deleting a component also requires the deletion of all links resolving to the component. Note that this also seems too rigorous. Even when the link consistency constraints are taken into account, deleting only the relevant specifiers from the link will usually be sufficient. Note the use of domain anti-restriction and range anti-restriction (resp. denoted by $\lhd$ and $\rhd$) to remove the associated identifiers from the accessor's domain and resolver's range.

- *modifyComponent* — This operation allows modification without the need for a combined delete/add operation. Function overriding ($\oplus$) is used to override the accessor function with the new component. Note that the resolver function is not modified, while it seems reasonable to assume that the resolver needs modification every time a component is modified.

- *linksToComponent* — This operation is used to determine which links have an end-point specification resolving to a specific uid.

- *linksToAnchor* — In the same way, *linksToAnchor* is used by the runtime layer to determine which links are related to a specific anchor.

Note that the last two schemas do not contain a Δ-list. This implies that they can only be used for inspection. Other operations related to link traversal are described in the runtime layer in Section 4.3.

4.2.5 Summary

This section provided a formalization of the Dexter storage layer. It models a hypertext as a set of atomic, link and composite components, with a resolver and accessor function. The most interesting aspects of the model include the clear separation between anchors and links, the addressable, n-ary relationships modeled by the link component and the explicit support for composition as modeled by the composite component. Critique on the model in the hypertext literature has addressed, among other issues, the lack of support for versioning, security, stretch text, transclusions, and multimedia synchronization. In addition, especially Dexter's strict link constraints have been subjected to critique from the developers of hypertext systems that are based on distributed link processing, such as the World Wide Web.

4.3 Runtime Layer

The Dexter runtime layer describes the mechanisms supporting the user's interaction with the hypertext. The fundamental concept in this layer is the *instantiation*, which represents the object on the user's screen that is used to interact with the components stored in the storage layer. Using structured document terminology, the instantiation models the presentation of the underlying (source) component. In Dexter, all user interaction is modeled via the instantiation: the user sees and edits a component's instantiation, not the component itself. The focus is not on presentation of the components contents, but on the manipulation of components via instantiations (creating, opening, modifying, etc.) and on link-based navigation.

4.3.1 Instantiation

Each instantiation has a unique instantiation id from the given set *Iid*. Halasz et al. describe the instantiation as:

> An instantiation consists of a *base instantiation* which "represents" a component, a sequence of *link markers* which "represents" the anchors of the component, and a function mapping link markers to anchor identifiers.
>
> Halasz et al. [113]

Consequently, we define the following sets:

$$[Iid, BaseInstantiation, LinkMarker]$$

and the *Instantiation* class:

Instantiation

$base : BaseInstantiation$
$links : \mathrm{seq}\, LinkMarker$
$linkAnchor : LinkMarker \pfun AnchorId$

$\mathrm{dom}\, linkAnchor = \mathrm{ran}\, links$

INIT

$links = \langle\,\rangle \land linkAnchor = \varnothing$

The *base* can be seen as an abstraction of the object that actually takes care of the component's display and user interaction, i.e. it could model the "widget" of a window toolkit that is used to render the object. Note that an instantiation does *not* contain any references to the component it represents. All administration that keeps track of the mappings between instantiations and their components is centralized in the *Session* class described in Section 4.3.2.

The link markers (in the sequence *links*) represent the active "hotspots" on the screen that can be used to trigger link traversal. To be able to find which anchor identifier is associated with the marker, the *linkAnchor* function is defined. Note that in practice, the various presentation specifications will affect the way anchors are mapped to link markers. When complex style sheets or transformations are used, this mapping may be non trivial: a single anchor may correspond to multiple sensitive regions on the screen, etc. Unfortunately, this effect of presentation specifications on link markers is not captured by the Dexter model.

4.3.2 Session

Session management is provided by the *session* class It models the user's interaction with a single *hypertext*, that is, to interact with more than one hypertext, a user has to start multiple sessions. The type of operations that a user enables to interact with the hypertext are listed by the *Operation* declaration.

$$\begin{aligned} Operation ::=\ & OPEN \mid CLOSE \\ & \mid PRESENT \mid UNPRESENT \\ & \mid CREATE \mid EDIT \mid SAVE \mid DELETE \end{aligned}$$

The various operations a user performs during a session are recorded in the session's *history* variable. Because the session is always opened during initialization, the first operation in the history list is always *OPEN* ($head(history) = OPEN$).

As discussed above, user interaction itself is modeled via the runtime instantiations of the components in the storage layer. For each component the user wants to interact with, a corresponding runtime instantiation needs to be created. This mapping is modeled by the *instantiator* function, which returns an instantiation given the component's Uid and a presentation specification.

Modifications made to the instantiation are made permanent by explicitly writing them back to the storage layer. This process is called "realizing" the edits and requires a function that maps the instantiation onto the new component. This is modeled by the *realizer* function. Note that an instantiation which is immediately followed by a realize operation should not change the component, this is ensured by the last state constraint:

$$\begin{array}{l} \forall\, uid : Uid;\ ps : PresentSpec \mid \\ \quad uid \in \operatorname{dom} hypertext.accessor \bullet \\ \qquad realizer(instantiator(uid, ps)) = hypertext.accessor(uid) \end{array}$$

To record the necessary information about which instantiation belongs to which component, the variable *instants* provides a mapping between instantiations and components: given the instantiation id, it returns both the instantiation and the Uid of the corresponding component.

In order to be able to resolve component specifications that require runtime knowledge ("the last component visited"), the notion of a runtime resolver is introduced. The *runTimeResolver* extends the resolver function from the storage layer (defined by the *Hypertext* class on page 120), a constraint ensured by $hypertext.resolver \subseteq runTimeResolver$.

In the following schema, we have only included the definitions of the *openComponents*, *followLink* and *editInstantation* operations. An informal description of these three operations will be given after the schema definition. We deleted the long definitions of the other operations, since they contribute little to the understanding of the model. A complete formalization of the session class can be found in Appendix C.

Session

$$\begin{array}{l} hypertext : Hypertext \\ history : \operatorname{seq} Operation \\ instants : Iid \rightarrowtail\!\!\!\!\!\!\rightarrow Instantiation \times Uid) \\ instantiator : Uid \times PresentSpec \nrightarrow Instantiation \\ realizer : Instantiation \rightarrow Component \\ runTimeResolver : ComponentSpec \nrightarrow Uid \\ \hline head(history) = OPEN \\ hypertext.resolver \subseteq runTimeResolver \\ \forall\, uid : Uid;\ ps : PresentSpec \mid \\ \quad uid \in \operatorname{dom} hypertext.accessor \bullet \\ \qquad realizer(instantiator(uid, ps)) = hypertext.accessor(uid) \end{array}$$

INIT

$$\begin{array}{l} hypertext.\mathit{INIT} \wedge history = \langle OPEN \rangle \wedge instants = \varnothing \\ instantiator = \varnothing \wedge realizer = \varnothing \wedge runTimeResolver = \varnothing \end{array}$$

openComponents

$$
\begin{array}{l}
\Delta(history, instants) \\
specs? : \mathbb{F}(Specifier \times PresentSpec) \\
\hline
history' = history \frown \langle PRESENT \rangle \\
\exists\, iids : \mathbb{F}\, Iid;\ newInstants : Iid \rightarrowtail\!\!\!\!\!\!\rightarrow (Instantiation \times Uid) \bullet \\
\quad \# newInstants = \# specs? \land \\
\quad iids = \operatorname{dom} newInstants \land iids \cap \operatorname{dom} instants = \varnothing \land \\
\quad (\forall\, s : specs? \bullet \exists\, iid : iids;\ uid : Uid; \\
\quad\quad cs : ComponentSpec;\ ps : PresentSpec;\ inst : Instantiation \mid \\
\quad\quad\quad cs = (first(s)).componentSpec \land \\
\quad\quad\quad ps = second(s) \land \\
\quad\quad\quad uid = runTimeResolver(cs) \land \\
\quad\quad\quad inst = instantiator(uid, ps) \bullet \\
\quad\quad\quad\quad newInstants(iid) = (inst, uid)) \land \\
\quad instants' = instants \oplus newInstants
\end{array}
$$

followLink

$$
\begin{array}{l}
\Delta(history, instants) \\
iid? : Iid \\
linkMarker? : LinkMarker \\
\hline
\exists\, aid? : AnchorId;\ uid? : Uid;\ links! : \mathbb{F}\, Link \bullet \\
\quad aid? = (first(instants(iid?))).linkAnchor(linkMarker?) \land \\
\quad uid? = second(instants(iid?)) \land \\
\quad hypertext.linksToAnchor \land \\
\quad (\exists\, specs? : \mathbb{F}(Specifier \times PresentSpec) \mid \forall\, s : specs? \bullet \\
\quad\quad (\exists\, link : links! \bullet first(s) \in \operatorname{ran}(link.specifiers)) \land \\
\quad\quad (first(s)).direction \in \{TO, BIDIRECT\} \land \\
\quad\quad second(s) = (first(s)).presentSpec \bullet \\
\quad\quad openComponents)
\end{array}
$$

editInstantiation

$$
\begin{array}{l}
\Delta(history, instants) \\
iid? : Iid \\
inst? : Instantiation \\
\hline
iid? \in \operatorname{dom} instants \\
history' = history \frown \langle EDIT \rangle \\
\exists\, uid : Uid \mid uid = second(instants(iid?)) \bullet \\
\quad instants' = instants \oplus \{iid? \mapsto (inst?, uid)\}
\end{array}
$$

Session operations The following *Session* operations have been defined above:

- *openComponents* This operation models the process of instantiating a set of new components. It is primarily designed for presenting components as a result of link traversal, which explains why the target components are specified by a link-end specifier, and not by a component specification or UID. In addition to the link-end specifier, a presentation specification to define the runtime presentation-specific information is also required. So for each component to be opened, the operation expects a $(Specifier \times PresentSpec)$ pair. A set of such pairs is given as input ($specs?$).

  The operation's behavior can then be summarized by the first and last line of the lower part of the schema. It first adds the *PRESENT* operation to the session's history ($history' = history \frown \langle PRESENT \rangle$). The last line adds the new instantiations to the session's *instants* function (using function overriding: $instants' = instants \oplus newInstants$).

  The code in between defines the *newInstants* function, which maps the identifiers of the new instantiations to their instantiations and Uid. It needs a new mapping for each link-end specifier/presentation specification pair ($\#newInstants = \#specs?$). Its domain is a new set of instantiation identifiers ($iids = \mathrm{dom}\, newInstants$) which are not already in use ($iids \cap \mathrm{dom}\, instants = \varnothing$).

  For each pair in *specs*, there needs to be a mapping ($newInstants(iid) = (inst, uid)$) such that *inst* is an instantiation of *uid* using the proper presentation specification ($inst = instantiator(uid, ps)$). Note that the component's *uid* is found by resolving the component specification in the link-end specifier ($cs = (first(s)).componentSpec$) using the the runtime version of the resolver function ($uid = runTimeResolver(cs)$).

- *followLink* This operation is relatively simple, since opening the link targets is done by *openComponents* described above. Recall that *openComponents* takes a set of pairs as its input, where each pair contains a link-end specifier and a presentation specification. So all we have to do here is define this set of pair (named $specs?$ in the schema).

  The operation assumes that when a link is selected for traversal, the ID of the associated instantiation (*iid*) and the selected link marker ($linkMarker?$) are known by the runtime layer and given as input. The first task of *followLink* is to map these runtime concepts to their associated storage layer concepts, that is, to the ID of the source component (*uid*) and the anchor identifier ($aid?$) of the corresponding anchor.

  The session's *instants* function is used to map the $iid?$ to the corresponding $(Instantiation, Uid)$ pair, which directly yields the *uid* we need. Additionally, the instantiation's *linkAnchor* function is used to retrieve the id of the source anchor: $aid? = (first(instants(iid?))).linkAnchor(linkMarker?)$. Note again that Dexter does not take into account the role of presentation specifications when mapping from run-time level link markers to document level anchor identifiers. In practice, style sheets and document transformations typically are a complicating factor for implementing this mapping.

The *uid*? and *aid*? can then be used by the hypertext's *linksToAnchor* operation to find the set of links (*links*!) that link from, or to, this anchor. This set is needed to retrieve the set of link-end specifiers and presentation specification pairs (*specs*?) for the *openComponents* operation defined above. This set is defined by requiring that for every pair *s* in *specs*?, the following conditions hold:

- The first element of the pair *s* is a link-end specifier of one of the links that link from (or to) the source anchor: $\exists\, link : links! \bullet first(s) \in \mathrm{ran}(link.specifiers)$.
- That link-end specifier is also an outgoing link: $first(s)).direction \in \{TO, BIDIRECT\}$.
- The second element of the pair *s* is the presentation specification of the corresponding link-end specifier: $second(s) = (first(s)).presentSpec$.

Note that the Dexter *followLink* operation does not use the feature of the *openComponents* operation to supply run-time presentation information, it simply supplies the presentation specification stored with the link-end specifier.

- *editInstantiation* This operation nicely illustrates the clear distinction the Dexter model makes between the instantiation object and the underlying storage-level component associated with the instantiation.

  An edit operation is only possible on objects that are already instantiated ($iid? \in \mathrm{dom}\, instants$). The operation updates the history ($history' = history \frown \langle EDIT \rangle$) and overrides the *instants* function with the a new mapping ($instants' = instants \oplus \{iid? \mapsto (inst?, uid))\}$). The new mapping consists of the original *iid*? and *uid*, and the new instantiation *inst*?. Note that these modifications to the instantiation will only affect the run-time layer. Committing these changes to the storage layer requires an explicit *realizeEdits* operation (see Appendix C for a definition of *realizeEdits* and the other remaining operations of the session class).

4.4 Discussion

The model presented offers more or less the same functionality as the original model described in [113]. However, by using Object-Z instead of plain Z, we have been able to provide a far more compact notation for the Dexter hypertext reference model[2]. For larger specifications, i.e. specifications that go beyond the typical toy examples found in textbooks, we think that Object-Z is generally better suited than plain Z. In the specification given in this chapter, the structuring facilities and other object-oriented features

[2]The formal part of the original specification contains more than 40 (mostly generic) Z schemas which require about 20 pages of text.

of Object-Z were especially useful for modeling the Dexter concepts of the *Component* and its *Atom*, *Link* and *Composite* subclasses. In the corresponding schemas of the original specification, several (artificial and hard to read) concepts and Z schemas had to be introduced to model this essentially object-oriented relationship in the Z notation (see also the discussion in Section 4.2.3).

Many aspects of the Dexter model have been criticized in the hypertext literature. Some of the disadvantages of the Dexter model can be addressed by removing constraints from the specification, without further implications for the model. For instance, the claim that Dexter only models closed systems can be (partially) addressed by removing the link consistency constraints. Other claims, such as the fact that Dexter does not model versioning, security and multi-user aspects of hypertext systems would require a more significant change to the model. The problems related to the lack of support for multimedia synchronization are addressed in the following chapter, in which we will use the specification given in this chapter as the basis for a formalization of the Amsterdam Hypermedia Model.

Chapter 5

The Amsterdam Hypermedia Model

The Dexter model, described in the previous chapter, models content data, link, anchor and composition structures in hypertext documents. This provided the hypertext community with a way of describing and comparing their documents and systems. As the use of synchronized hypermedia documents increases, and as linking to and from parts of these documents becomes more common, an updated version of the Dexter model is required to compare these more complex hypermedia structures.

The Amsterdam hypermedia model (AHM) describes a hypermedia document model which includes temporal and spatial relationships among constituent media elements, and also pays attention to defining the traversal behavior of links among groups of dynamic media items. Many of the concepts that are central to the model have found their way into the SMIL language discussed in Chapter 3.

The description of the AHM has mainly relied on informal descriptions and its (partial) implementation in the CMIFed system [228]. This chapter provides a formal specification of the AHM.[1]. The formal specification process helped to abstract from the implementation details of CWI's authoring and play-out environment CMIFed, and to keep the model as generic as possible. Furthermore, the specification process helped to find inconsistencies and design flaws in earlier versions of the model. Parts of the model have not yet been implemented in the CMIFed system, and especially for checking these parts of the model, a formal approach proved to be very useful.

The objective of developing the formal specification was not, however, to prove formal correctness of the model, nor to prove the correct behavior of systems implementing the model. Instead, the specification gives a concise description of the model, which helps to develop a deeper understanding of the more complex concepts of the AHM, and provides a good basis for comparison with other hypermedia models. The informal prose accompanying the formal Object-Z schemas is used to stress the model's important concepts, to illustrate the differences with respect to the Dexter model, and to point

[1]While the formalization of the AHM was developed by the author of this thesis, the formalization process initiated many discussions with one of the original developers of the AHM at CWI, Lynda Hardman. The formal model itself and the results of the discussions are reflected in this chapter, as well as in a previous version that was included as Appendix 1 of Hardman's PhD thesis "Modelling and Authoring Hypermedia Documents" [121].

out the relevant relationships with the Web-oriented languages discussed in Chapter 3.

Note that the specification given in this chapter is based upon the description of the AHM as described in [121] and differs from the AHM as originally described in [119].

5.1 Introduction

The Amsterdam Hypermedia Model provides an abstraction of, and an extension to, the hypermedia model implemented by CWI's CMIFed hypermedia authoring environment [228].

The Web, and many of the hypertext systems which formed the basis of the Dexter model, started off as text-oriented systems, to which other media types were added later. In contrast, CMIFed was originally designed as a multimedia system, extended by hyperlink support in a relatively late stage in its development. This multimedia background explains the central role of temporal relationships in the composition mechanisms of the model, the way the behavior of hyperlinks is related to these structures and the explicit modeling of spatial layout.

When compared to the Dexter model, the main extensions of the AHM are its multimedia specific semantics for composite components, the notion of link context and explicitly defined spatial layout. Since the formal specification given in the following sections will focus on these three topics, we first provide a brief, informal, introduction below.

5.1.1 Composition structures

In contrast to the generic composition facilities of the Dexter model, the AHM defines two specific composition mechanisms: temporal and atemporal composition.

Temporal composition provides synchronization of media items by placing them on the same time axis. This type of composition is common in multimedia systems. Examples include the parallel and sequential composition elements of SMIL, discussed in Chapter 3. By using temporal composition exclusively, the resulting hypermedia document represents a multimedia presentation with a single, strictly linear time-axis. As a consequence, hyperlinks with a destination inside the same document can only be used to jump back and forward to positions on a single, linear timeline. The only escape from this linearity is by linking to another document. Traversal of such links typically launches the presentation of the target document.

A more refined hypermedia model would allow links that affect only part of the presentation, while other parts continue playing in parallel. It would also need to provide non-linearity within a single document. A non-linear document is, from a temporal perspective, a document that does not have a completely predefined timeline. Intuitively, the document can be regarded as having a timeline with one or more "holes". Each of these holes forms a slot that can be used to play a specific part of the document. Which parts of the document would be used to fill these slots, and in what order, is determined interactively at runtime. When compared to the traditionally linear multimedia model, this non-linear model supports presentations with interactive behavior that more closely follows that of traditional hypertext documents. It allows, for example, the user to follow different paths through parts of the presentation, while other parts play in parallel in a predefined schedule. It also allows presentations containing hyperlinks to parts that are optional, and that would not be played if the user never selected the corresponding link.

In the AHM, a document's timeline can indeed contain "slots" such as described above. In fact, it features a recursively structured timeline, because each slot can be used to play a part of the document that contains media items and other "holes" in its timeline. Since the structure of each part that can be played in a slot is essentially the same as that of the overall document, the terms "root document" and "sub-document" are sometimes used. Which sub-documents are played in a specific slot is determined at runtime, by means of hyperlink activation. As discussed above, temporal composition alone does not allow the author to create a non-linear document. To model non-linear documents, the AHM introduces the notion of atemporal composition. Atemporal composition allows grouping of elements that represent alternative sub-documents that are accessible by means of hyperlinking.

For example, imagine a hypermedia presentation which includes a glossary with many text entries. There are at least two obvious reasons for not including the entries within a temporal composite, and to use an atemporal composite instead. Typically, the entries should only be presented after an explicit request from the user, in the AHM this implies that each entry is only accessible by means of link traversal. If they were included in the document by using temporal composition only, they would *always* be presented at some predefined point in the presentation. Note that this is the situation in SMIL 1.0 presentations, where each media object included in the document will also be included in the presentation. As a consequence, the example document could not be modeled in SMIL 1.0 without resorting to scripting. Another reason for not using a temporal composite is the absence of an obvious temporal relation. While there is a clear structural relation between the entry components and the glossary component, there neither is a temporal relation between the glossary and the entry components, nor are there any temporal relations among the entry components.

In the AHM, the document sketched above could be modeled by a top-level temporal composite (e.g. a `par` in SMIL) with two children. The first child could be another temporal composite containing the main flow of the presentation, the other child would be an atemporal composite containing the text entries of the glossary.

The temporal and atemporal composites are defined in Section 5.5.

5.1.2 Spatio-temporal layout

The Dexter model does not model space and time explicitly but rather assumes all spatio-temporal relationships among components to be hidden in the presentation specifications (from the given set *PresentSpec*). However, the ability to express temporal and spatial relationships between components is too important to be omitted from a hypermedia reference model. As the use of synchronized media increases, comparing the techniques used to express these relationships becomes an essential part of comparing different hypermedia systems. Additionally, it is useful to have commonly accepted abstraction mechanisms and terminology addressing exactly these topics. For example, one of the main objectives for developing the HyTime standard was the need within the SGML-community for standardized methods of (spatio-temporal) alignment. As such, we see common abstractions for spatio-temporal alignment as a requirement for interoperable hypermedia systems.

As discussed in Chapter 2, spatial layout of multimedia documents differs from traditional text layout. The spatial layout of multimedia components requires more explicit specification because hardly any layout information can be derived from the (lexical) position of the component in the document. In addition, within a single multimedia document, each media item can have different properties with respect to resizing (whether or not aspect ratios should be preserved), positioning (specified in absolute coordinates or as a percentage of the parent window), the window it should be displayed in (when multiple windows are open at the same time), etc. Since such layout properties are often shared by more than one media component, the AHM models these properties as a shared resource, the *channel*. The channel is formalized in Section 5.3.1.

Another characterizing feature of multimedia discussed in Chapter 2 is temporal layout. The AHM's model of temporal layout differs radically from its spatial layout model. Temporal layout is tightly bound to the overall document structure as modeled by the temporal composite component discussed above. All temporal layout is defined by binary relations between the children of temporal composites. These relations are modeled using the concept of the *synchronization arc*, formalized in Section 5.3.2. Note that this implies a limitation in terms of reuse: while the spatial layout specifications defined in the channels can be easily shared, the temporal layout specifications defined in the synchronization arcs cannot. In practice, the drawbacks of this limitation can to a large extent be overcome by providing adequate authoring support (for example, in the GRiNS authoring tool, many of the binary relations are automatically generated from higher level specifications [51]).

5.1.3 Link context

The Dexter hypertext navigation model is insufficient for hypermedia. In an environment with dynamic media, it is common to have presentations with multiple active streams of media. In such an environment, it becomes more important to define the scope of the link, that is, which part of the presentation is affected by link traversal. To be able to define the scope of a hyperlink in a declarative way, the AHM introduces the notion of link context [120]. Link contexts make explicit which part of the current presentation stops or pauses and how much of the link destination is presented.

For instance, the hypermedia presentation in Figure 5.1 on the facing page contains an audio and video track, together with some text components. In one of the texts,

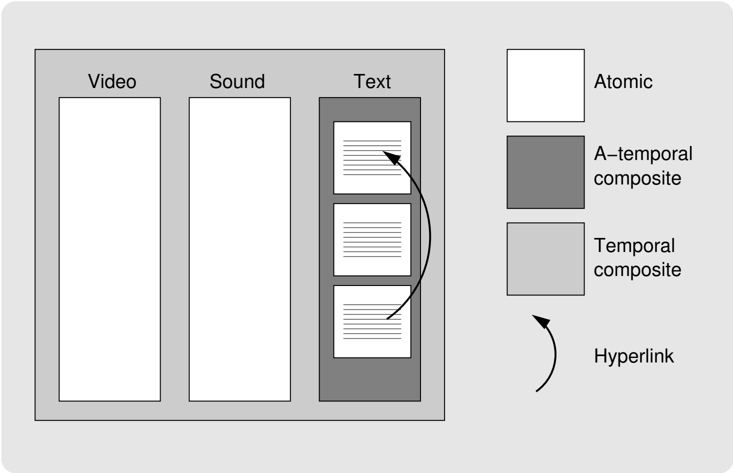

Figure 5.1: The link between the text atoms may affect the sound and video atoms.

there is a link to another text. The effect that traversal of this link will have on the sound and video presentation depends on the definition of the contexts of the link. If the source context is limited to the third text item, and the destination context to the first text, the destination text will simply replace the source text, and the sound and video presentation will continue with no interruption. However, if the source context was defined to be the entire temporal composite, the presentation of both the sound and video node could be stopped or paused upon link traversal. Which traversal behavior is the most appropriate will depend on the document. Link contexts give the author of the document the ability to fully control this type of link behavior. The notion of link context is formalized in Section 5.4.

The remainder of this chapter is devoted to the formal specification of the AHM storage layer, focusing on the three concepts discussed in this section: composition structures, spatio-temporal layout and link context. In the next section, we formalize some low-level differences between the Dexter and AHM model. In Section 5.3 defines the AHM's layout concepts in terms of channels and synchronization arcs. Section 5.4 formalizes the notion of link context, while Section 5.5 defines the AHM atom, link and composite components. Finally, Section 5.6 defines the AHM counterpart of the Dexter *Hypertext*, the *Hypermedia* class.

5.2 Preliminaries

Before we start with the specification of (a)temporal composition, spatio-temporal layout and link context, we first need to introduce some basic classes that reflect a number of differences between the AHM and the Dexter model.

Presentation Specifications The AHM explicitly discriminates the information describing temporal and spatial relationships from other, stylistic presentation information. The styles are modeled by a *PresentSpec*, and as in Dexter, we consider the inner structure of a *PresentSpec* beyond the scope of the model.

AhmPresentSpec
$style : PresentSpec$

In the following sections, we will provide additional structures to specify spatio-temporal layout. Since all objects that need spatio-temporal layout also need to store presentation information, these structure will be modeled by subclassing *AhmPresentSpec*.

Object References Dexter uses the *ComponentSpec* to indirectly refer to a component. The AHM also applies the advantages of this indirect addressing mechanism to other objects, including media items, anchors and channels. In this way, one can refer to these objects by means of a database query as is already possible for components in the Dexter model. Note that this allows components to refer to an anchor by querying over an existing anchor database, which is different than the concept of a runtime-computed or virtual anchor value, which is for example used by Microcosm's generic link mechanism [69]. In Dexter, this can be modeled by using queries in the anchor value, which is part of the target component. In the AHM, the query can also be modeled as an anchor reference, which is part of the link-end specifier. Anchor references could, for example, query the contents of (a set of) documents to compute the desired anchors. The specifications arise from the following sets:

$$[MediaItemSpec, AnchorSpec, ChannelSpec]$$

Anchoring In the AHM, anchors are modeled as first class objects, and not as attributes of a component, as in the Dexter model[2]. This better models the situation in systems that externalize anchors in the same way as they externalize links. It also models the more advanced virtual anchors that use an *AnchorSpec* query in the link-end to define the actual anchor.

The Dexter anchor is further extended by adding a style specification and semantic attributes. In Dexter, anchor style can be modeled in the link-end *Specifier*, but by locating the style information in the anchor, multiple link-ends can share the same anchor style, while a link-end may still override the style specification provided by the anchor. Adding semantic attributes to anchors allows knowledge-oriented applications to store meta data associated with the anchors (as is used, for example, in MacWeb [171]), and can also be used for querying and other information retrieval processing.

[2]The role of the anchor reference was defined somewhat ambiguously in [121]. We follow the text on page 51 of [121], which suggests that the anchor reference semantics is similar to Dexter's indirect component referencing.

AhmAnchor

$$
\begin{array}{l}
\mathit{anchorStyle} : \mathit{PresentSpec} \\
\mathit{attributes} : \mathit{Attribute} \pfun \mathit{Value} \\
\mathit{anchorValue} : \mathit{AnchorValue}
\end{array}
$$

Note that *PresentSpec*, *Attribute*, *Value* and *AnchorValue* are the Dexter types as defined in the previous chapter.

Components We introduce the *AhmComponent* as a base class for the AHM atom, link and composite components described in section 5.5. Subclasses of the component class differ from their Dexter counterparts in containing a list of *AnchorSpec*s referencing the extended *Anchor* type described above.

AhmComponent

Component

$$
\mathit{anchors} : \mathrm{seq}\, \mathit{AnchorSpec}
$$

Resolver functions are needed to map the specifications to the specified objects. The definition of the channel resolver needs to be deferred and is given after the definition of the *Channel* object.

$$
\begin{array}{l}
\mathit{mediaResolver} : \mathit{MediaItemSpec} \pfun \mathit{Atomic} \\
\mathit{compResolver} : \mathit{ComponentSpec} \pfun \downarrow\!\mathit{AhmComponent} \\
\mathit{anchorResolver} : \mathit{AnchorSpec} \pfun \downarrow\!\mathit{AhmAnchor}
\end{array}
$$

Note that the resolvers no longer return identifiers (such as Dexter's *Uid* and *AnchorId*). While the value semantics of Z made the use of identifiers necessary in the original Dexter specification, we consider the use of explicit identifiers and accessor functions superfluous and use the implicit object identity built-in the Object-Z language.

5.3 Spatio-Temporal Layout

Spatial layout is modeled differently from temporal layout. Spatial layout definitions are defined by *Channel*s. Components can share the same spatial layout by using the same *Channel*. Temporal relations are modeled by synchronization arcs (*SyncArcs*).

5.3.1 Spatial relationships: channels

The spatial layout of a component is defined by associating that component with a particular *channel*. In this respect, channels are similar to the region and the root-layout in

SMIL (discussed on page 103). Each channel specifies its spatial *extent* and its *position*. To facilitate managing complex documents that need many channels, channels contain an optional reference to another *parent* channel. This allows related channels to be grouped in a hierarchical structure[3]. A channel's extent may overlap with another channel, but the channel hierarchy forms a strict spatial containment hierarchy similar to the temporal *during* hierarchy formed by the temporal composites. Channels that do not contain a reference to a parent are typically used to model top level windows (cf. the `root-layout` in SMIL), in which case the *position* and *extent* model the position and size of the window. For channels that have a parent, the *position* and the spatial *extent* of the channel is to be interpreted relatively to the dimensions of the channel's parent.

[*ResourceSpec*]

Channel

AhmPresentSpec

$parent : ChannelSpec$
$position : \mathbb{R} \times \mathbb{R}$
$extent : \mathbb{R} \times \mathbb{R}$
$resourceSpec : ResourceSpec$
$attribute : Attribute \pfun Value$

$channelResolver : ChannelSpec \pfun Channel$

Where SMIL regions are only used for spatial layout, channels in the AHM may additionally define *style* properties that model shared style defaults for the components that are associated with the channel. For example, an author may change the default font of all captions in the document, by changing the font of the "caption-channel", instead of changing the presentation specification of all individual caption components. Although not explicitly defined by the AHM, visual style properties of channels could easily be subjected to inheritance rules, as is common in style sheet languages such as CSS. Note that the inheritance of visual properties in the temporal hierarchy typically results in unwanted results. In contrast, the channel hierarchy forms a spatial hierarchy that is well-suited for this type of inheritance.

There are some other subtle differences between SMIL regions and AHM channels. SMIL regions are media independent. The same region can first be used for an image and later in the presentation reused for a video. In contrast, AHM channels are media dependent, and can only be used for a single media type. Whereas in SMIL only visual media objects have regions, in the AHM all atomics (including audio components) have an associated channel.

[3]Although SMIL 1.0 regions cannot be nested, hierarchical region structures similar to the AHM channel hierarchy are expected to be included in the second version of SMIL [17].

In the CMIFed implementation, channels are also used for resource allocation. They are regarded as abstract output devices for playing the contents of components. At run-time, channels are mapped onto physical output devices. To be able to specify resource dependent information, AHM channels include a *resourceSpec* attribute.

Finally, to facilitate easy selection (by means of the *channelResolver*), channels have a set of semantic *attribute*s. By selecting or de-selecting specific channels, the document can be easily adapted to individual preferences (e.g. choosing between an English or Dutch sound channel) or hardware resources (e.g. turning off a video channel where video is not supported).

5.3.2 Temporal relationships: synchronization arcs

Temporal relationships among the descendants of a temporal composite are defined using synchronization arcs. Such temporal constraints indicate a preferred delay and allowable deviations. The constraints can be *ADVISORY*, meaning that realization of the constraint at runtime is desirable, but not strictly necessary, or *HARD*, meaning that violating the constraint would be an error. The initial values all default to (advisory) zero delays.

$$SyncType ::= HARD \mid ADVISORY$$

TemporalConstraint

$$\begin{array}{l} preferredTime : \mathbb{R} \\ minimimBeforeTime : \mathbb{R} \\ maximumAfterTime : \mathbb{R} \\ type : SyncType \end{array}$$

INIT

$$\begin{array}{l} preferredTime = minimimBeforeTime = maximumAfterTime = 0 \\ type = ADVISORY \end{array}$$

A synchronization arc is defined by references to the anchors of the arc's source and destination, followed by the temporal constraint between these components. Synchronization arcs are used to denote temporal constraints among descendants of a temporal composite, and are considered to be part of the composite's presentation specification (see also Section 5.5.3). Depending on the anchor, constraints can be defined between intervals (when the anchor has an explicit or implicit duration) and/or between specific points (when the anchor refers to the begin or end time of a component, or to a specific frame or sample). When intervals are used, any of the thirteen possible temporal interval relations defined by Allen ([11], see also page 52) can be described by at most two synchronization arcs [118, 121]. Additionally, constraints may be defined on specific points in the presentation. For instance, a synchronization arc may be used to synchronize a video frame with an audio sample. In that case the anchors resolve to an individual frame or sample. Because synchronization arcs define temporal relations, the source and destination component of the arc should be different.

$$
\begin{array}{|l}
\textit{SyncArc} \\
\hline
\textit{source} : (\textit{ComponentSpec} \times \textit{AnchorSpec}) \\
\textit{destination} : (\textit{ComponentSpec} \times \textit{AnchorSpec}) \\
\textit{constraint} : \textit{TemporalConstraint} \\
\hline
\textit{compResolver}(\textit{first}(\textit{source})) \neq \textit{compResolver}(\textit{first}(\textit{destination})) \\
\hline
\textit{INIT} \\
\textit{constraint}.\textit{INIT}
\end{array}
$$

Note that while synchronization arcs are structurally very similar to binary hyperlinks, we explicitly do not overload the hyperlink for specifying temporal constraints. Hyperlinks are considered to primarily describe semantic relationships and navigation structures. In contrast, synchronization arcs cannot be used for describing semantic relationships, nor for navigation, but are only used for describing temporal presentation information. Both links and synchronization arcs, however, use the same media-independent anchoring mechanism to address their end points.

5.4 Link Context

The declarative aspects of the concept of link context can be easily formalized by deriving a new class from the Dexter link-end *Specifier*. We need a flag which indicates whether the source context needs to be continued[4], paused or deactivated.

$$\textit{SourceActivation} ::= \textit{CONTINUE} \mid \textit{PAUSE} \mid \textit{DEACTIVATE}$$

While the destination context is normally activated in a playing state, authors can also specify a paused state. The latter can be useful, for instance, to link to a particular detail in a single frame in a video fragment. Paused destination contexts are assumed to be started by user interaction.

$$\textit{DestinationActivation} ::= \textit{PLAY} \mid \textit{PAUSED}$$

A specific link-end can be part of the source context in one link, and part of the destination in another link. The presentation specification for the link-end specifier contains

[4]The need for *CONTINUE* is debatable, because a similar effect can always be achieved by removing the components that need to continue from the source context. From an implementation perspective, *CONTINUE* can be used in systems where the author cannot control which components are part of a link's source context. In SMIL, for example, the source context is always considered to be the entire presentation that contains the link. The use of the `new` value for SMIL's `show` attribute (see page 101) can be considered to be equivalent to the use of *CONTINUE* in the AHM.

both flags, and inherits a *style* attribute from the *AhmPresentSpec*. The styles of the link-ends can be used, for instance, to model the transition effect of the link (e.g. the source context may dissolve into the destination context). Although SMIL 1.0 does not support transition effects, they are expected to be included in the second version of SMIL [17].

SpecifierPresentSpec

AhmPresentSpec

$srcActivation : SourceActivation$
$dstActivation : DestinationActivation$

INIT

$srcActivation = CONTINUE$
$dstActivation = PLAY$

The context itself is modeled as attribute on the link-end specifier. Note that a link can have many anchors, so modeling context by adding source and destination contexts attributes to the link itself would not suffice. On the other hand, modeling context on the anchor level, as is the case in MacWeb [171], would make it hard to reuse an anchor in another link (which might need to specifiy another context).

So for each link-end specifier, the component that is considered to be the context of the link is specified by a *context* attribute of type *ComponentSpec*. The context is typically a reference to a composite containing the link-end component. For example, if the source of a link is an anchor in a subtitle component, the associated context is likely to be the composite containing the subtitle along with the video and sound track component.

LinkSpecifier

Specifier

$presentSpec : SpecifierPresentSpec$
$anchor : ComponentSpec \times AnchorSpec$
$context : ComponentSpec$

$direction \neq NONE$

INIT

$context = first(anchor)$
$presentSpec.\mathit{INIT}$
$direction = FROM$

By default, the specifier is initialized to represent the most simple case, i.e. where the context component equals the component containing the anchor.

The context's role (whether it is a source or destination context) depends on the direction of the associated specifier (*FROM* or *TO* respectively). Actually, the context can act as both source and destination context (if *direction* = *BIDIRECT*) so in general, the contexts of a link can only be determined at runtime. Note that Dexter's use of *NONE* has been criticized [108] because of its undefined semantics. The AHM simply disallows a *NONE* direction.

The fact that the context is not a part of the presentation specification, reflects the fact that context is considered to add structural (and indeed semantic) information to a hyperlink. It other words, it is considered to represent more than "just" presentation information. The structural role of contexts is further emphasized by disallowing the context to consist of an arbitrary set of components. Instead, it is required to be a single (composite) component, and thus closely related to the existing document structure.

5.5 Components in the AHM

Keeping the above descriptions of spatio-temporal relationships in mind, we can now formalize the various components of the AHM. We describe the AHM atomic, link and composite component, focusing on the structure of the spatio-temporal information within the components. Additionally, we discuss the difference between the Dexter and AHM components and their specifications.

5.5.1 Atoms

The AHM atomic component mirrors its Dexter counterpart, but makes its spatio-temporal characteristics explicit. We expect all spatio-temporal arithmetic to be carried out using real numbers (the associated unit used, e.g. seconds, milliseconds, pixels or centimeters, is outside the scope of the model). The duration and layout information of the atomic component is described in its presentation specification, since it provides layout information which needs to be interpreted in the runtime layer.

Presentation Specification of Atoms The AHM atoms' temporal characteristics are reflected by their *duration* (stated by the author or as an intrinsic property of the media items). Their spatial layout is defined by the associated *channel*. In contrast to SMIL 1.0, atoms in the AHM may define a specific *position* and *extent* within the extent defined by their channel.

AtomPresentSpec

AhmPresentSpec

$duration : \mathbb{R}$
$channel : ChannelSpec$
$position : \mathbb{R} \times \mathbb{R}$
$extent : \mathbb{R} \times \mathbb{R}$

Anchoring in Atoms The concept of anchoring for atomic components needs to be extended to include a duration. For continuous media items, the media dependent *AnchorValue* can be expected to define the duration of an anchor (for example, by defining a range of frames for a video fragment). But for static media such as text, we need to define the interval in which the anchor is active. Note that the atomic anchor inherits attributes to store semantic information (these can be used for information retrieval or knowledge representation purposes) and an anchor style presentation specification from *AhmAnchor*.

```
AtomicAnchor
AhmAnchor
  startTime : ℝ
  duration : ℝ
```

$$\begin{array}{l} \textit{AtomicAnchor} \\ \hline \textit{AhmAnchor} \\ \quad \textit{startTime} : \mathbb{R} \\ \quad \textit{duration} : \mathbb{R} \end{array}$$

The Atomic Component The AHM atomic component is a subclass of the *AhmComponent* base class. It contains a *presentSpec* and *content* attribute. The media content is referred to by a system dependent media item reference (this can be a filename, URL or database query) and a media dependent anchor value denoting which part of the media item is used. In this way, the model is independent of the granularity of the server providing the media items (e.g. an author does not need to create a new image file if only a part of that image needs to be included in the presentation).

$$\begin{array}{l} \textit{AhmAtom} \\ \hline \textit{AhmComponent} \\ \quad \textit{presentSpec} : \textit{AtomPresentSpec} \\ \quad \textit{content} : \textit{MediaItemSpec} \times \textit{AnchorValue} \\ \hline \quad \forall\, \textit{aspec} : \operatorname{ran} \textit{anchors} \bullet \textit{anchorResolver}(\textit{aspec}) \in \textit{AtomicAnchor} \end{array}$$

Note the list of *anchors* that is inherited from *AhmComponent*. These anchor references should resolve to the *AtomicAnchor*s defined above ($\forall\, aspec : \operatorname{ran} anchors \bullet anchorResolver(aspec) \in AtomicAnchor$).

5.5.2 Links

Following the Dexter model, the link is a component with a sequence of link-end specifiers. It can be used to define links of arbitrary arity. Because the AHM does not associate spatio-temporal information with the link component itself, there is no need to specify a new presentation specification class for link components. Style information, for instance to display the link in a link browser, is specified using a *style* attribute of type *PresentSpec*. Being a component, links can be end points of other links, so a link needs anchors just as the other components do. The link component inherits its attributes and anchors from *AhmComponent*.

In order to model transitions, a link may specify the transition's *duration* and a *position* (which models the position of the destination relative to the source anchor or mouse click) and a list of specifiers. Note that the link contains only one duration and position attribute, which is not sufficient for links containing multiple destinations. This problem needs to be solved in a future version of the model.

AhmLink
AhmComponent

$specifiers : \mathrm{seq}\, LinkSpecifier$
$style : PresentSpec$
$duration : \mathbb{R}$
$position : \mathbb{R} \times \mathbb{R}$

5.5.3 Composites

One of the most fundamental differences between the Dexter and AHM composite is that the AHM discriminates between two types of composition: temporal and atemporal composition. These differences are primarily reflected by the different presentation specifications below. Another difference with the Dexter composite lies in the AHM notion of a composite anchor.

Anchoring in Composites The AHM addresses Dexter's under-specification of anchors for composite components [108] by defining the composite anchor value to be a list of references to anchors defined by the descendants of the composite. See Figure 5.2 on the next page. This can be used to group anchors, and allows one to build a DAG structure of composite anchors[5]. Semantic attributes and style information can be attached to each anchor, and are inherited from the *AhmAnchor*. Note that in SMIL, anchors can only be defined at the level of the atomic components, and the notion of a composite anchor does not exist.

CompositeAnchor
AhmAnchor

$anchorValue : \mathrm{seq}(ComponentSpec \times AnchorSpec)$

The Composite Component The *AhmComposite* serves as an abstract base class for the two composites.

[5] Thanks to Randy Trigg who noticed that limiting the composite anchor structure to a strict hierarchy as described in [121] was over-restrictive and unnecessary.

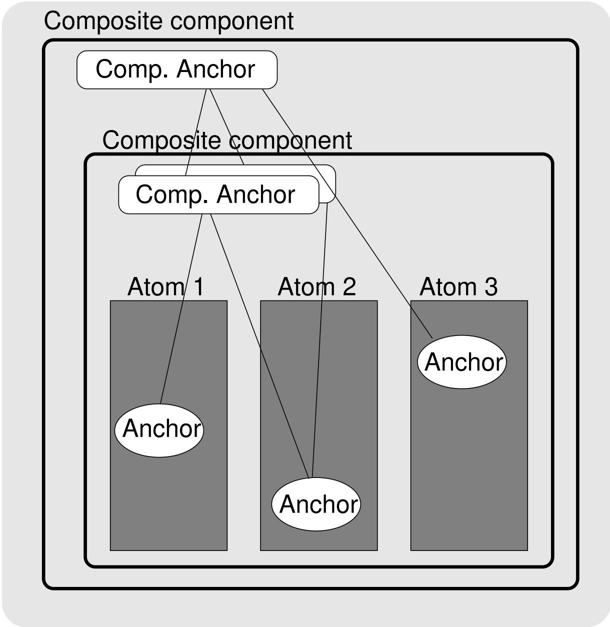

Figure 5.2: Composite anchoring hierarchy.

$$
\begin{array}{l}
\textit{AhmComposite} \\
\hline
\textit{AhmComponent} \\
\hline
\textit{anchors} : \mathrm{seq}\, \textit{CompositeAnchor} \\
\textit{children} : \mathrm{seq}\, \textit{ComponentSpec} \\
\Delta \textit{descendants} : \mathbb{F} \downarrow \textit{AhmComponent} \\
\hline
\forall a : \mathrm{ran}\, \textit{anchors} \bullet \forall \textit{avalue} : \mathrm{ran}\, a.\textit{anchorValue} \bullet \\
\quad \exists d : \textit{descendants};\ \textit{aspec} : \textit{AnchorSpec} \bullet \\
\qquad \textit{compResolver}(\textit{first}(\textit{avalue})) = d \land \\
\qquad \textit{aspec} \in \mathrm{ran}\, d.\textit{anchors} \land \\
\qquad \textit{anchorResolver}(\textit{second}(\textit{avalue})) = \textit{anchorResolver}(\textit{aspec}) \\
\hline
\end{array}
$$

The second state invariant requires that the anchors of the composite refer to existing anchors of the composite's descendants (the formal definition of *descendants* mirrors that of the Dexter composite and is left out for reasons of brevity).

The Temporal Composite The temporal composite component is used to define a strict *during* hierarchy, i.e. a temporal hierarchy where each child can only be active when its parent is also active (see page 52). The children of a temporal composite share the same timeline. Within the constraints imposed by the during hierarchy, the position of the children on the timeline can be further contrained by using synchronization arcs.

The presentation specification of the temporal composite contains a list of synchronization arcs defining the temporal relations among its children. By specifying an explicit duration a user can override the intrinsic duration of the composite (the playing environment may scale or clip the children in order to achieve this):

$$
\begin{array}{l}
\textit{TemporalPresentSpec} \\
\hline
\textit{AhmPresentSpec} \\
\quad \textit{duration} : \mathbb{R} \\
\quad \textit{syncArcs} : \operatorname{seq} \textit{SyncArc}
\end{array}
$$

The temporal composite extends the *AhmComposite* base class with the temporal presentation specification:

$$
\begin{array}{l}
\textit{TemporalComposite} \\
\hline
\textit{AhmComposite} \\
\quad \textit{presentSpec} : \textit{TemporalPresentSpec} \\
\hline
\quad \forall\, \textit{syncarc} : \operatorname{ran} \textit{presentSpec}.\textit{syncArcs} \bullet \\
\qquad \exists\, \textit{source}, \textit{dest} : \textit{descendants};\ \textit{aspec}_1, \textit{aspec}_2 : \textit{AnchorSpec} \bullet \\
\qquad\quad \textit{aspec}_1 \in \operatorname{ran} \textit{source}.\textit{anchors} \land \\
\qquad\quad \textit{aspec}_2 \in \operatorname{ran} \textit{dest}.\textit{anchors} \land \\
\qquad\quad \textit{compResolver}(\textit{first}(\textit{syncarc}.\textit{source})) = \textit{source} \land \\
\qquad\quad \textit{compResolver}(\textit{first}(\textit{syncarc}.\textit{destination})) = \textit{dest} \land \\
\qquad\quad \textit{anchorResolver}(\textit{second}(\textit{syncarc}.\textit{source})) = \textit{anchorResolver}(\textit{aspec}_1) \land \\
\qquad\quad \textit{anchorResolver}(\textit{second}(\textit{syncarc}.\textit{destination})) = \textit{anchorResolver}(\textit{aspec}_2)
\end{array}
$$

The state constraint ensures that all synchronization arcs refer to valid anchors of descendants of the composite.

The Atemporal Composite In contrast to the temporal composite, the children of an atemporal composite are scheduled on different timelines, and their temporal relationships (if any) can only be determined at runtime. Since there are no temporal relationships between the children of an atemporal composite, the atemporal presentation specification contains no synchronization arcs. Instead, it specifies the initial activation state of each of the children. This state can be play, pause or inactive. By default, all children of an atemporal composite are inactive, and inactive children can only be made active as a result of link traversal:

$$\textit{ActivationState} ::= \textit{PLAYING} \mid \textit{PAUSING} \mid \textit{INACTIVE}$$

$$
\begin{array}{l}
\textbf{AtemporalPresentSpec} \\
\quad AhmPresentSpec \\
\quad \boxed{initialStateOfChildren : \mathrm{seq}\, ActivationState} \\
\quad \mathit{INIT} \\
\quad \forall as : \mathrm{ran}\, initialStateOfChildren \bullet as = INACTIVE
\end{array}
$$

The specification of the atemporal composite mirrors the definition of its temporal counterpart. It requires the specification of an initial state for all its children:

$$
\begin{array}{l}
\textbf{AtemporalComposite} \\
\quad AhmComposite \\
\quad presentSpec : AtemporalPresentSpec \\
\quad \hline
\quad \#children = \#presentSpec.initialStateOfChildren
\end{array}
$$

5.6 The Hypermedia Class

Finally we define the *Hypermedia* class. The composition structure in AHM, as in Dexter, specifies a directed, acyclic graph. The AHM differs, however, by requiring that all components are descendants of a single root component. Note that documents with multiple potential root elements can always be combined together into a single root element using atemporal composition. The hypermedia state schema contains only one primary variable: the root composite. The set of components is directly dependent on the descendants of the root, and thus defined as a secondary variable.

$$
\begin{array}{l}
\textbf{Hypermedia} \\
\quad root : \downarrow AhmComponent \\
\quad \Delta \\
\quad components : \mathbb{F} \downarrow AhmComponent \\
\quad \hline
\quad \mathrm{ran}\, compResolver = components \\
\quad components = \varnothing \\
\quad \quad \vee \\
\quad (\exists_1\, r : \downarrow AhmComposite \mid r = root \bullet components = r.descendants \cup \{r\}) \\
\quad \mathit{INIT} \\
\quad components = \varnothing
\end{array}
$$

The first invariant mirrors the constraint of the Dexter model that ensures accessibility of the components. It states that every component should be accessible by the external component resolver (ran *compResolver* = *components*). The second invariant is specific to the AHM. It states that the set of components is either empty, or, if it is not empty, there needs to exist a unique root such that every other component is a descendant of the unique *root* component ($components = r.descendants \cup \{r\}$).

The Dexter hypertext class specification on page 120 contains many more consistency constraints which can only be ensured in a "closed" hypermedia system. In particular, Dexter requires all links to refer to existing components and anchors, *within* the system. As a result, deleting a component involves deleting all links resolving to the deleted component. Since these constraints can never be ensured in an open environment (such as the WWW), they have been left out in the specification above.

5.7 Discussion

The main differences between the Dexter and Amsterdam Hypermedia Model are related to spatial and temporal layout. In the Dexter model, all spatial and temporal relationships among components are assumed to be hidden in the presentation specification, arising from the given set *PresentSpec*. The presentation specification is the main interface between the storage and the runtime layer. While the internal structure of the *PresentSpec* is considered to be beyond the scope of the model, Dexter makes heavy use of this concept. Each component in the storage layer has a *PresentSpec* to store presentation information local to the component. Additionally, each link-end specifier uses a *PresentSpec*, for instance to store information on how the target should be displayed in case a link is followed. Note that the link as a whole — being a component itself — also has a *PresentSpec*. To make the situation even more complex, the runtime layer may add an extra presentation specification in order to be able to reflect runtime knowledge in the specification of a component. Yet, even this large number of presentation specifications proves to be insufficient in some cases [119].

More importantly, however, we argued that temporal and spatial relationships need to be explicitly modeled by a hypermedia reference model. The Amsterdam hypermedia model provides extensions to the Dexter model that address these issues. We have expressed these abstractions as part of a formal description of the model using the specification language Object-Z. The formalization process helped to refine the model that was originally presented in [119]. Most of the flaws that were found have been corrected in the full description of the AHM as given in [121]. Some others were found after the publication of [121] and have been included in this chapter (including the role of the *AnchorSpec*, the DAG-structure of composite anchors, the role of the *CONTINUE* value in source contexts and the importance of the root component in the during hierarchy of the overall hypermedia document structure).

In general, the formal specification process enforced clear decisions about which concepts of the Dexter model could be reused within the AHM, which should be extended and which should be dropped. The advantages of the formalization are, however, limited to the structural aspects of the AHM as described by the AHM storage layer. In order to formalize the operational behavior of the additions to the storage layer as presented in this chapter, the formalization of the Dexter runtime layer needs to be extended. The current specification of the Dexter runtime layer focused primarily on the mechanics of link traversal. A similar specification of the AHM runtime layer should include the presentation effects of contexts upon link traversal, and model the temporal behavior of, and requirements on, the possible state transitions within the model. As described in the next chapter, specification languages such as Z and Object-Z are not especially useful for modeling temporal behavior.

In addition, neither Dexter nor the AHM presentation specifications provide explicit support for the single source, multiple delivery processing model presented in Chapter 2. The during hierarchy of the main document structure in the AHM severely limits the variation style sheets have in presenting the document. Changes in the order of the presentation that change the duration hierarchy require changes in the main document structure. The same applies to changes in link behavior, that might require structural changes in order to define the right context. Transition effects, in many aspects a stylistic presentation issue, are nevertheless part of the core link structure.

Another limitation is that the AHM models the media content of atomic components as black boxes. This prevents the scheduling of objects *inside* atomic components. This is, for example, an explicit goal in the specification of the second version of SMIL [17], which not only specifies the synchronization of other Web-resources (as the AHM and SMIL 1.0 do), but also the synchronization of objects inside such resources (e.g. the synchronization and animation of specific text fragments in an HTML page or graphic objects in an SVG presentation). These issues need to be addressed in future versions of the model. In the next chapter, we take a closer look at the modeling of document transformation and temporal behavior.

Chapter 6

Modeling Transformations and Temporal Behavior

In the previous two chapters, we used the Dexter and Amsterdam Hypermedia Model to illustrate many hypermedia-related topics, including: complex linking, anchoring, composition, spatio-temporal layout and link context. Two important topics, however, have not been addressed in the formalizations given so far: document transformation processes and the temporal behavior of hypermedia documents at runtime.

Document transformations played an important role in the first part of the thesis, when we discussed the multiple delivery publishing model (in Chapter 2) and its application on the Web (in Chapter 3). The formal models discussed so far, however, do not properly reflect the role document transformations and style sheets play in many of today's multiple delivery publishing hypermedia systems. Section 6.1 uses a formal description of a (stateless) document transformation to illustrate the basic aspects of today's style sheet languages.

The other aspect that has played an important role in Chapters 2 and 3, and has received little attention in the formalizations so far, is the temporal behavior of hypermedia presentations at runtime. While the formalization of the AHM given in Chapter 5 describes time-related data-structures at the storage level, it does not formalize how they are mapped to a specific temporal behavior at runtime. Section 6.2 investigates methods that can be used to formalize the specification of the real-time behavior of hypermedia systems.

6.1 Document Transformations and Style Sheets

This section formalizes a typical document transformation system, where the transformation is defined by a finite number of rules in a style sheet. As most style sheet models, it takes a (hierarchically) structured document with no inherent presentation-oriented semantics as input (the input tree) and produces a (hierarchical) data structure describing the presentation as output (the result tree). For this purpose, we reuse the hierarchical data-structure of the Dexter hypertext as defined in Chapter 4.

Note that we only model *stateless* transition systems. In such systems, all rules are side-effect free, i.e. a rule may not change variables that influence the effect of other rules. This is, for example, the case in CSS [33], XSLT [58] and DSSSL [136] specifications[1].

We define a rule as a function which maps a component to a set of output components. To avoid conflicts in the style sheet in the cases where multiple rules match a given input component, we assign a priority (a natural number) to each rule:

$$Rule == (Component \nrightarrow \mathbb{F}\, Component) \times \mathbb{N}$$

Note that this definition abstracts from many details:

- The definition does not describe how to decide, for a specific component in the source document, whether the rule matches or not. In most style sheet languages, the part of the rule that defines to which components the rule applies is called a *selector*. Different style sheet languages have different selector models. The expressiveness of the syntax used to define these selectors typically has a large impact on the overall power and ease-of-use of the style sheet language. The same applies to situations in which more than one rule matches. CSS in particular has elaborate mechanisms to deal with potential conflicts between rules from different style sheets.

- The definition does not describe how components are created, and how these components are added to the structure of the target tree. The components that are created are often called *formatting objects*. The mechanism for creating formatting objects also varies among different style languages. In DSSSL, for instance, the set of target formatting objects is pre-defined, while XSLT allows transformations to a set of formatting objects of any given XML Namespace. In CSS, the transformation process is less explicit. Instead, CSS focuses on the style properties of a component, and the structure of the input and result tree are similar. This simplifies cascading of multiple CSS style sheets, since they can all operate on the style properties of a similar structure.

These properties make the definition above sufficiently abstract to specify a wide variety of style rules. It models rules that are specific for a single input component (e.g. rules for setting the font of the title of the document, or removing a particular component with a specific *UID*). On the other hand, it also models more general rules, that provide mappings for sets of components (e.g. rules for setting the font of all section headers, or reordering a list of items).

Typically, rules do not provide mappings for all possible input components, so the mapping is a partial function ($\nrightarrow$), and not a total function ($\rightarrow$). Additionally, transformations are typically irreversible. This applies in particular to formatting transformations — you can generate the formatted presentation from the source document, but not

[1] In contrast, the DejaVu document framework discussed in Chapter 8 allows modification of global variables.

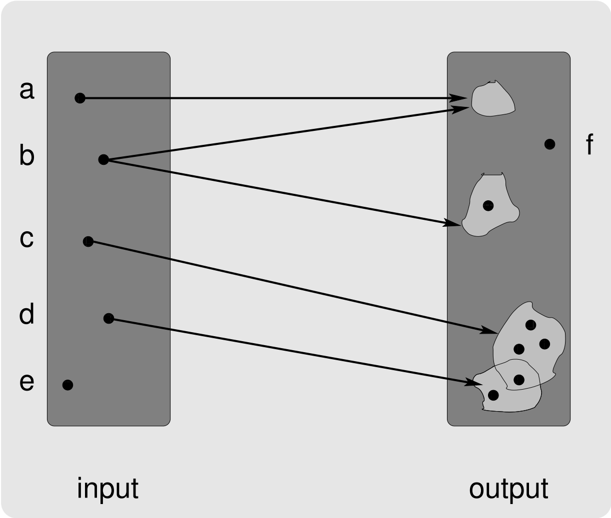

Figure 6.1: Example mappings between input and sets of output components.

vice versa. This explains why the rule is not modeled by an injective function (⤔). Note that this property is a potential problem for interactive applications, where there is often a need to map user interactions on the presentation level (e.g. editing or selecting an anchor) back to the underlying components of the source document. Such applications typically need to maintain extra information to enable such backward transformations.

Given a set of style rules, these can be grouped together to form a *Mode*

$$Mode == \mathrm{seq}\, Rule$$

A mode typically consists of style rules that together perform a specific formatting task. In XSLT, for example, one mode may be used for creating an index, while another mode generates a table of contents. Modes can also be used to group style rules for a specific media type, as is common in CSS. Note that the rules are put in a sequence, because in many style languages, the order in which the rules are defined is significant.

A style sheet is modeled simply by a sequence of one or more modes:

StyleSheet

$modes : \mathrm{seq}\, Mode$

INIT

$\# modes \geq 1$

Before specifying the *Transformation* schema, we discuss some example mappings (see Figure 6.1). In the figure, input component a is correctly mapped, but it is explicitly mapped to an empty set of output components: a will be ignored and not end up in the output document. Such mappings are, for example, common in data-oriented applications where only a small part of the input tree needs to be presented. They are also common in document-oriented applications with style sheets that use different modes, e.g. one mode is used to add a table of contents to the result tree, and a second mode is used to format and add the main body of text to the result tree. In the first mode, all components in the input tree that do not contribute to the table of contents could be mapped to an empty set in the result tree.

Two mappings exist for component b. The two rules could be part of separate modes, and in this case there is no conflict: the style engine will choose the right rule according to the mode. If the rules belong to the same mode, the transformation should ensure the use of the one with the highest priority — a requirement we will explicitly enforce in the following schema.

Component c and d are mapped to overlapping sets of output components. Note that in practice, sets of output components are typically disjoint. However, we will allow overlaps since they could be useful for modeling the generation of components in the result tree from multiple style rules.

No mapping is defined for component e, a situation which we explicitly disallow in the *Transformation* schema below. The same applies to the spurious f component in the output document, which is not the result of any mapping in the style sheet. Note that this follows current practice to the extent that all components in the result tree need to originate from an associated style rule and source component. This also applies to components in the result tree that contain only literally generated text (e.g. a date stamp or page number): the generated text needs to be specified by some style rule, and even such a style rule needs to have an associated source component.

Finally, we define the *Transformation* schema. Note that it does not have a state to model variables that can be modified by the transformation. It simply produces an output document from an input document and a style sheet:

Transformation

$$
\begin{array}{l}
in?, out! : Hypertext \\
stylesheet? : StyleSheet \\
\hline
\forall mode : \operatorname{ran} stylesheet?.modes \bullet \forall cin : in?.components \bullet \\
\quad \exists priority : \mathbb{N} \bullet \\
\quad\quad (\exists mapping : Component \mathrel{+\!\!\!\rightarrow} \mathbb{F}\, Component;\ couts : \mathbb{P}\, out!.components \bullet \\
\quad\quad\quad\quad (mapping, priority) \in \operatorname{ran} mode \land cin \mapsto couts \in mapping) \land \\
\quad\quad (\nexists higherpriority : \mathbb{N};\ alternative : Component \mathrel{+\!\!\!\rightarrow} \mathbb{F}\, Component; \\
\quad\quad\quad otherouts : \mathbb{P}\, out!.components \bullet \\
\quad\quad\quad\quad (alternative, higherpriority) \in \operatorname{ran} mode \land \\
\quad\quad\quad\quad cin \mapsto otherouts \in alternative \land higherpriority > priority) \\
\forall cout : out!.components \bullet \exists mode : \operatorname{ran} stylesheet?.modes; \\
\quad cin : in?.components;\ couts : \mathbb{P}\, out!.components; \\
\quad mapping : Component \mathrel{+\!\!\!\rightarrow} \mathbb{F}\, Component;\ priority : \mathbb{N} \bullet \\
\quad\quad (mapping, priority) \in \operatorname{ran} mode \land cin \mapsto couts \in mapping \land \\
\quad\quad cout \in couts
\end{array}
$$

The first constraint ensures "completeness" by requiring that for each mode, all input components are used in the transformation. That is, it disallows unused components such as component *e* in Figure 6.1 on page 153. It also ensures that for each input component the rule with the highest priority is applied. Literally, it states that for all input components in all modes, there is a rule which maps that component to a set of output components and there are no other matching rules with a higher priority. Note that in the case there are multiple rules with the same "highest" priority, the above specification does not describe which one is chosen. Since the rules are ordered, one could easily add the requirement that in such cases, the first or the last rule is applied.

The second constraint ensures "soundness" by requiring all output components to be the result of applying a valid rule to an input component, i.e. components may not "magically" appear in the output out of the blue, as is the case for component *f* in Figure 6.1 on page 153. Literally, it states that for all components in the output, there is a corresponding input component and matching rule in some mode of the style sheet.

Note that for a typical style language, the "completeness requirement" is reflected explicitly in the definition of the language. In XSLT, for example, this requirement is always met by XSLT's built-in default rules, that ensure that the a style rule exists for every element in the input tree. Similar defaults exist for other style languages. On the other hand, the "soundness requirement" is typically the implicit result of a correct (procedural) implementation of the style engine, but is needed in the formal text because of the declarative nature of the specification.

The specification given above abstracts from many details. Nevertheless, it still adequately illustrates some important aspects of multiple delivery publishing that have not been modeled by Dexter and the Amsterdam Hypermedia Model. The specification models fundamental properties of style languages such as CSS, DSSSL and XSLT, including the notions of stateless document transformations, prioritized style rules and multiple style modes.

The final fundamental property missing in the formal specifications given so far is the temporal aspect of the run-time behavior of hypermedia presentations. This is the topic of the following section.

6.2 Modeling Temporal Behavior

While the specifications given in Chapter 5 formalize the data-structures that underlie the temporal aspects of the Amsterdam Hypermedia Model at the storage level, the temporal behavior of a hypermedia document during run-time has not been specified.

Note that the specifications discussed so far all use the Object-Z language. This is the main reason why these specifications do not address the temporal run-time behavior of hypermedia presentations: as argued in [28, 79], formal modeling techniques such as Z and Object-Z are ill-suited for modeling the dynamic, real-time behavior of multimedia systems. There are however, a few examples of the use of Z and Object-Z for the specification of real-time systems. Fridge, for instance, augmented Z with notations found in RTL (real-time logic) [102], Coombes developed an interval logic for modeling time in Z [64], and Smith and Hayes investigated a real-time version of Object-Z based on timed refinement calculus [210].

But in general, notations such as Z [214] or VDM [144] are not suited for modeling the real-time behavior of hypermedia systems. Consequently, alternative techniques have been proposed for modeling this important aspect of hypermedia systems. This section provides an overview of techniques other than Z and Object-Z and their application to the domain of multimedia and hypermedia. It is partially based on Blair et al. [28], who discuss a broad spectrum of specification techniques, including:

- specification logics,
- process algebras,
- finite state machines, and,
- Petri nets

Before we take a closer look at these specification techniques, we discuss an important property that applies to all of these techniques: the extent to which they support qualitative and quantitative modeling of time.

Modeling qualitative and quantitative time Specification languages such as Z are well suited for the specification of properties such as *safety* (i.e. some desired property is always true). For other properties, such as *liveness* (i.e. some desired property will eventually become true) the specification technique needs to incorporate a notion of qualitative time.

In addition to qualitative time, quantitative modeling of time is also regarded as a requirement for specifying the dynamic behavior of hypermedia systems. Quantitative timing allows the specification of *timeliness* properties, e.g. stating that some desired property is true at a specific time instance or during a specific temporal interval. Typical examples include "the display of this image will start at time t_1 and end at time t_2" or "the play-out of this video will start with a delay that is less than 500 milliseconds".

Unfortunately, most popular specification techniques do not model (quantitative) time, so this needs to be added by means of extension mechanisms. Frequently, such timed extensions have unwanted side-effects. For example, most of these techniques have a mathematically well-defined semantics. These semantics form the basics underlying the tools that are developed to support the use of the modeling technique involved. Many of the timed extensions proposed in the literature lack such well-defined semantics. This makes it hard or even impossible to extend the existing tools to support the extension. Other side effects with practical implications address the decidability properties of the technique involved. Several timed extensions make verification questions undecidable, where they were decidable in the original, unextended version of the technique. Unfortunately, the more expressive timed extensions are usually undecidable.

Additionally, when introducing timed extensions to a formal modeling technique, one has to deal with a common trade-off in formal modeling: the conflict between usability and rigor. For example, Duce et al. [79] did not use a rigorous approach in formalizing the temporal aspects of the specification of the multimedia standard PREMO [215]. While developing PREMO, they used a combination of two untimed modeling techniques (Object-Z and LOTOS) to specify PREMO's object model and dynamic behavior. They followed Jones in his use of "formal methods light" and focused on the advantages of thinking about PREMO in terms of abstract, mathematical defined states, and paid no attention to formal proof or refinement. Duce et al. claim that, even with the limited amount of time and other resources they had available, this lightweight approach allowed them to gain insight in PREMO's object model and find several flaws in the specification text. However, their pragmatic approach had also some practical drawbacks: the possibilities of using tools were severely limited by the lack of rigor.

In general, one of the main goals of formal modeling techniques (describing the system in a highly abstract way) often contradicts with the need to incorporate quantitative timing (which requires the specification to deal with the lowest level of performance details) [28].

Below, we summarize the pros and cons of the most common formal specification techniques that deal with timing issues: temporal logics, process algebras, finite state machines and Petri nets.

6.2.1 Temporal logics

Temporal logics [181] are a subset of modal logics. The modal operators in most temporal logics only deal with qualitative time. A typical temporal logic includes the operators $\bigcirc$ (next), $\Box$ (henceforth), $\Diamond$ (eventually) and $\cup$ (until).

Temporal logics that are extended to model quantitative aspects are usually referred to as real-time temporal logics. Quantitative time can be introduced by bounding these operators. In MTL [151] for example, $\Box_{>2}p$ means: after time 2, p always holds. Other techniques used are freeze quantification, (where quantifiers bind their variable to a specific time) and the introduction of explicit clock variables (where a special variable is bound to the value of a global clock).

Temporal logics usually have an interleaving semantics, but whether the extra expressiveness of branching time outweighs the advantages of the simplicity of linear time is still a matter of debate. Additionally, Blair et al. [28] note that the declarative style of logic based specification techniques is well-suited for specifing time constraints, but makes it hard to specify the interactive behavior of a hypermedia system. They claim that the "algorithmic" character of techniques based on process algebra leads to more natural specification of interactive behavior.

6.2.2 Process algebras

Process algebra is rooted in the concurrency theory. Bergstra and Klop's Algebra of Communicating Processes (ACP [21]), Milner's Calculus of Communicating Systems (CCS [169]) and Hoare's Communicating Sequential Processes (CSP [40, 126]) are regarded as the classical process algebra based languages. Central to these languages is a (small and elegant) set of operators. This set typically includes operators for synchronous communication, deterministic and non-deterministic choice, concurrency, restriction (information hiding) and relabeling (to avoid naming conflicts). Their rich and tractable semantics made process algebras not only a favorite object of research, but also allowed the development of a number of validation techniques and supporting tools. Additionally, the composition properties of process algebras provide the natural support for refinement and substitution that is lacking from methods based on finite state machines or Petri-Nets.

While ACP, CCS and CSP are regarded as the prime languages from a theoretical perspective, LOTOS [133, 30, 160] is one of the more popular specification tools in practice. LOTOS is standardized by ISO and is often used in combination with the abstract data typing language ACT-ONE. It is supported by several toolkits.

The support for real-time in the methods based on process algebra is comparable with the situation for temporal logics discussed above. Most languages model only qualitative time, but numerous extensions have been developed to model quantitative time. In particular, several timed extensions to CSS, CSP and LOTOS have been proposed, see [28] for an overview. Most of these extensions model so called Zeno-processes, i.e. processes that are allowed to process an infinite number of instantaneous events in a finite time interval. While this may be unrealistic in terms of executability, this disadvantage does generally not outweigh the advantages of the significantly reduced theoretical complexity of Zeno-processes. Another counter-intuitive property found in some algebras is the fact that time no longer proceeds when the system is in a deadlock. Most algebras however, do allow the passage of time, even in the case of a deadlocked process.

While the use of process algebras leads to a more natural specification of behavior when compared to logic-based approaches, specifications based on quantitative timed extensions of process algebras tend to be extremely low level. They do not benefit from the advantages of high-level abstraction and often suffer from over-specification and related problems [28].

6.2.3 Finite state machines

A finite state machine (FSM) defines a set of states (which can have a duration) and a set of instantaneous events, which define a change of state. Most modeling techniques that are based on finite state machines have a number of common extensions to the pure FSM model. *Hierarchical composition* is needed to overcome the limitations of the flat FSM model which makes specifications of even moderately complex systems unmanageable. *Parallel composition and inter-machine communication* allows for modeling concurrent behavior in the otherwise strict sequential FSM model. New mechanisms are needed to allow communication between parallel FSMs. Common communication extensions include synchronous, asynchronous and broadcast communication. Finally, most modeling techniques extend the control flow paradigm of FSMs with a data model. Together, these additions are sufficient to make the extended finite state machines (EFSM) Turing-complete. EFSM have been frequently applied by the telecom industry as a means of describing communication protocols, and both ISO and CCITT have standardized modeling techniques (Estelle[132] and SDL [129], respectively) based on EFSMs.

As for temporal logic and process algebra, real time extensions have also been developed for EFSM-based models. Common approaches are based on adding an extra delay or time-out operator. Both Estelle and SDL support such a quantitative timing model. Again, it proves to be hard to incorporate these timed extensions into the formal semantics of EFSMs.

6.2.4 Petri nets

Petri nets are similar to the methods discussed above, to the extent that various extensions to the basic Petri net model have been developed. *Colored* Petri nets, for instance, have a more elaborate data part (e.g. typed tokens). Other extensions add extra composition power which allows the construction of larger Petri nets from smaller ones, overcoming the disadvantages of the flat structure of original nets. Like the EFSM-based models, Petri nets have been extensively used to describe communication protocols. Additionally, there has been a large amount of literature about using timed Petri nets for modeling multimedia systems (see, for example, [78, 235, 187, 31, 158]).

But unlike most of the methods discussed above, Petri nets model true concurrency. Additionally, Petri nets are regarded as one of the first formal models of concurrency for which real-time extensions have been developed. Timed Petri nets generally come in three flavors: timed place, timed arc or timed transition Petri nets. Again, a trade-off between expressive power and analytical power has to be made: timed transition Petri nets have the most expressive power, but are notoriously hard to analyze.

6.3 Discussion

This chapter addressed the two important issues that were not modeled by the Dexter and Amsterdam Hypermedia Model: document transformations and runtime temporal behavior.

Document transformations and style sheet processing are not modeled by the Dexter and AHM. This is not because these models fail to address presentation or style-oriented issues. The Dexter model of Chapter 4 uses opaque *presentation specifications* to model presentation and style-related properties of all components of a hypertext. The Amsterdam Hypermedia Model described in Chapter 5 explicitly models the data structures describing temporal layout (*temporal composites* and *synchronization arcs*) and spatial layout (*channels*), and leaves other style properties to Dexter-like presentation specifications, of which the inner structure is not specified. The major limitation of Dexter and the AHM is — from the perspective of the multiple delivery publishing model — that these models do not differentiate between the structure of the source document and that of the final presentation. Consequently, they are unable to model the transformation process that maps the source to the destination structure.

In this chapter, we gave a formal specification of a simple, stateless document transformation that is sufficiently expressive to illustrate many aspects of today's style sheet languages. It does not, however, specify the interaction between the spatio-temporal aspects that are an intrinsic part of the document, and those aspects that are presentation-specific and are controlled by the style sheet. In many cases, these interactions will be complex and application-specific. In addition, such a specification will require a more abstract document model than is currently provided by the AHM. It should be able to discriminate between the various types of spatio-temporal information and should allow a more abstract composition scheme than the current one based on (a)temporal composition.

In addition to document transformations, we discussed the difficulties related to modeling real-time behavior in formal specifications in general, and in languages such as Z and Object-Z in particular. We also discussed some formal specification techniques other than Z and Object-Z. Unfortunately, a commonly used, standardized specification technique that is well-suited for modeling real-time behavior does, at the time of writing, not exist.

Still, we think that there is a practical need for such a modeling technique. For example, there is currently still no formal model describing the SMIL timing model. This is by many considered a serious issue, since many of the errors and ambiguities[2] in the current SMIL-1.0 specification give rise to practical problems that could have been prevented by an adequate formal model.

6.4 Conclusion of Part II

This second part of the thesis revisited many of the hypermedia issues discussed in Part I, but now from a more formal perspective. It particular, it described the Dexter and AHM models. In addition, it sketched a simple document transformation system and provided a short overview of the role of quantitative time in formal specification techniques other that Object-Z. While the specifications given in the previous chapters helped us to better describe the systems involved, there are some drawbacks as well. Some of these drawbacks relate to these specifications in particular, some others apply to formal specifications in general.

[2]As an example, SMIL 1.0 fails to specify whether a media element with a `repeat` value of `2` and a `dur` value of `10s` should repeat two times within ten seconds, or that the repeat should multiply the maximum duration, which would result in a total duration of 20 seconds. This ambiguity would have been prevented if the repeat semantics had been specified in a more formal context.

First, all formal specification techniques have their limitations. These limitations may lead to obscure specifications, or — even worse — limit the model described. As an example, the limited way the Z type system supports the specification of recursive types, has led to the counter-intuitive notion of a BASE_COMPONENT in the original Dexter model. This notion is not only used in the formal specification, but even in the informal description of the model [114]. Another example is Dexter's claim of the component hierarchy being a directed, acyclic graph (DAG). In languages with a value semantics (such as Z), it is hard to specify characteristics of a system which is based on referencing, and one can argue that the hierarchy as defined in the formal specification is in fact a strict tree, and not the DAG it claims to be.

Another general source of ambiguity is a lack of conformance between the formal and informal specification. Because a document with only formal text is close to unreadable, it is common practice to include both a formal and informal description of the model, and when these two descriptions do not match, the validity of the formal specification becomes at least doubtful. In the case of the Dexter model there are several places where the informal description deviates from the formal specification. For instance, the informal text describes an example document, where a composite component directly includes media content, while in the formal specification only atomic components are allowed to include media content. This has resulted in some confusion in the hypertext literature [114].

A third class of errors is caused by a mismatch between the well defined meaning of certain concepts in a formal context, and their often loose interpretation in other contexts. For instance, in Dexter, the notion of the resolver function is used to map component specifications (e.g. a database query) to the UID of the component matching that identifier. However, in the formal context, the meaning of the word "function" is defined as a mathematical function: i.e. it maps each element of its domain to exactly one element in its range. As a result, a query as modeled by Dexter can only return a single UID — a limitation not found in the systems it claims to model.

Finally, one of the most important aspects of a multimedia system or document is its real-time behavior. Formally describing this real-time behavior is, however, still notoriously difficult, and which specification technique is best-suited for this task is, from many aspects, still an open issue.

Nevertheless, the processes that resulted in the formal specifications given in this part of the thesis helped in developing a better understanding of the systems involved, and to detect errors in a early stage. It also facilitated a more precise explanation of many of the issues discussed in the first part of the thesis.

Now we have a clear notion of the important issues in hypermedia modeling, we can discuss the software architectures needed to implement these models. In the last part of this thesis, we look at architectural issues for hypermedia systems, based on two examples of hypermedia architectures which are based on the models and techniques discussed in the first two parts of the thesis.

Part III

Architectural Issues in Structured Hypermedia Processing

Chapter 7

Hypermedia Software Architectures

This last part of the thesis provides an overview of the research issues related to the design of hypermedia systems. Once more, we revisit our research agenda, but now from an architectural perspective, focusing on the software components supporting style sheet processing and document transformation. Chapter 7 gives an introduction to hypermedia software architectures. Chapters 8 and 9 give a detailed description of two hypermedia architectures, the architecture of the DejaVu framework developed at the Vrije Universiteit, and the Berlage environment developed at CWI. Finally, Chapter 10 summarizes the main results of the thesis.

This chapter is structured as follows. First, we explain the notion of a software architecture and its importance in the hypermedia domain, then we give an introduction to the basic concepts of hypermedia architectures. Additionally, we describe a number of architectures that are representative of those developed within the different research communities described in Part I, and conclude with a summary of the fundamental differences between them.

7.1 Introduction

We use two definitions of the term software architecture. The first one, a loose definition, defines architecture as:

> The set of specifications that describe the interfaces, protocols and file formats that the components of a particular system use to communicate with the external world.

This definition is often (implicitly) assumed when the architecture of the Web is defined in terms of the HTTP, URL and HTML specifications. It emphasizes the openness of the architecture, by focusing on the mechanisms that allow external applications to communicate with the system described. According to this definition, an architecture describes a system as a black box, and only defines the interfaces with the external world.

However, in some cases we would like to have a clear understanding of the internal decomposition of a system, for instance because we need to extend the system or replace a particular component. Therefore, we also use the stricter definition of Bass et al. [19]:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Using this definition, the description of the architecture of the Web in terms of HTML, HTTP and URLs needs to be extended with a description of a decomposition into client and server components and, if necessary, with a further decomposition of the client and the server, the interfaces of these components and their inter-relationships.

The two definitions are closely related. A set of specifications is usually developed after system development in a particular domain has reached a certain level of maturity: a number of (prototype) implementations have been developed independently within different organizations. A need for interoperability among these systems may result in the specification of an architecture as a set of common interfaces and protocols (loose definition). In some cases, such specifications can be developed independently of the internal decomposition of the system involved. Different systems can implement the same interfaces, while their internal decompositions diverge. New systems may design their own internal decomposition in order to meet the given specifications.

In other cases, however, the interfaces or protocol specifications only make sense in the context of a given decomposition, which need to meet a number of minimal requirements. The description of the decomposition and the associated requirements is then an integral part of the specifications. Such specifications often define a *reference architecture*. Where a reference *model* provides a standard division of the functionality of a system, a reference *architecture* is a mapping of that functionality onto a system decomposition [19]. Note that a reference architecture does not necessarily define a complete software architecture, but merely describes the architectural assets that are reused across similar software architectures.

In the following discussion of hypermedia architecture, we will use the term software architecture to explicitly refer to the second and more strict definition.

7.2 Hypermedia systems and their Architectures

Early hypermedia systems were often designed as closed, monolithic systems, and their description in the research literature focused on the system's user interface and the underlying hypermedia document model. The limitations of these monolithic designs have made modular designs more popular (see Figure 7.1 on the next page). The software architecture of hypermedia systems has become an important issue in the research literature (see [50] and [176] for an overview). The main reason for the decomposition of these systems can be summarized as the need for reuse, flexibility, distribution and interoperability.

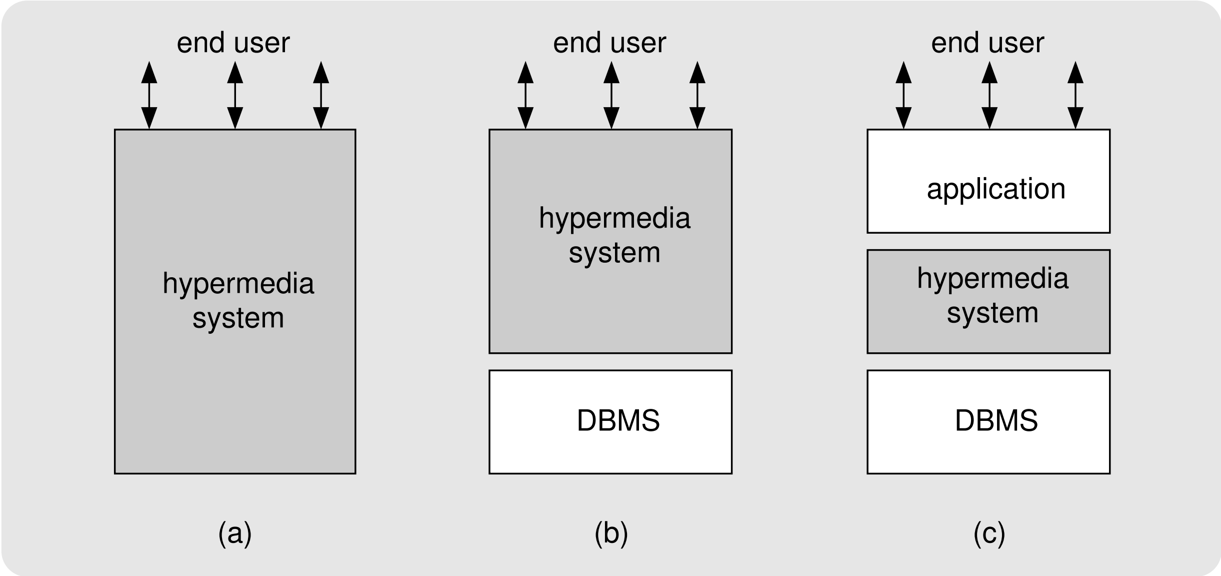

Figure 7.1: Monolithic hypermedia architectures (a) evolved to architectures delegating data storage to a DBMS (b) and user interface to domain-specific applications (c).

- **Reuse** — As a consequence of the rapid development of the hypermedia field in the eighties and early nineties, a large number of new systems needed to be built. New prototypes were built to experiment with new hypermedia functionality, and existing systems were extended to add domain specific behavior. Modular hypermedia architectures were needed in order to reuse components that were not subject to change, or to delegate parts of the implementation to other (sub)systems. For example (see (b) in Figure 7.1), many hypermedia systems delegated data storage functionality to a database management system (DBMS).

- **Flexibility** — Another reason for decomposition was flexibility. Users wanted to be able to tailor and modify domain-specific aspects, without changing the more generic hypermedia functionality of the system. Hypermedia architectures were further decomposed, and usually provided generic hypermedia functionality in a middle layer. Domain-specific applications could then be built on top of this common hypermedia layer (see (c) in Figure 7.1).

- **Distribution** — A third argument for decomposition was the need for distributed hypermedia systems. Second generation hypermedia systems (see Chapter 2, page 36) were typically stand-alone applications, and only those documents that were stored on the local file system were available to the user. To overcome this limitation, hypermedia architectures based on a client/server model have been developed, see (a) in Figure 7.2 on the next page. Typically, domain-specific components are located at the client side, and a generic server is responsible for storing the documents. Whether the components that implement the core hypermedia functionality should be located at the client side or the server side is an impor-

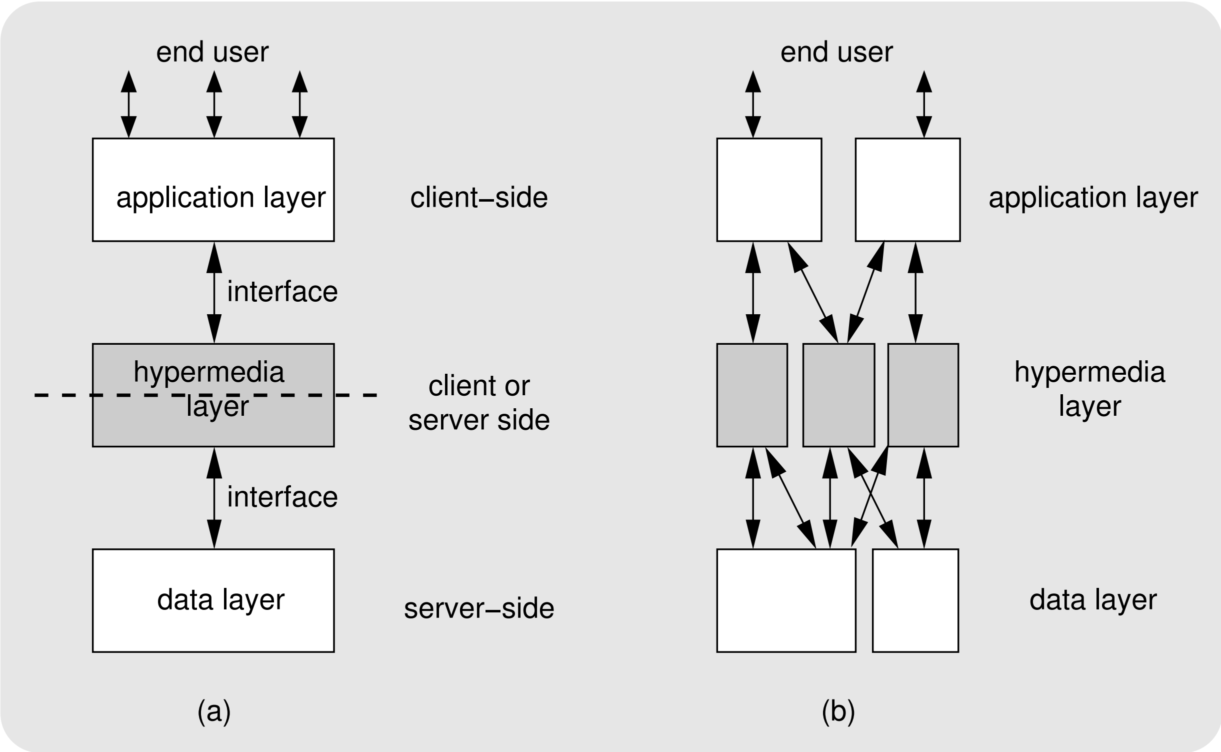

Figure 7.2: The core hypermedia functionality of a distributed hypermedia system can be located at the client or server side (a). Further decomposition may be needed to improve interoperability (b).

tant design consideration, since it has a large impact on the functionality and the performance of the resulting system (to be discussed in the next section).

- **Interoperability** — A fourth reason for decomposition is interoperability. In order to exchange information or to make use of services provided by other systems, users need hypermedia systems that are able to interoperate with other hypermedia systems. Additionally, users want hypermedia systems that are capable of interoperating with other applications and tools they frequently use (such as editors, word processors, spreadsheets, mail readers, etc). Both classes of interoperability require stable, standardized and well-defined public interfaces to the components concerned (see (b) in Figure 7.2). Typically, a per-component specification of the interface is not sufficient to allow other applications to make use of the system's hypermedia services. In these cases a reference (software) architecture is defined to describe the environment in which the various components are expected to operate.

Despite these common requirements in terms of reusability, flexibility, distribution and interoperability, the different agendas of the research communities described in Part I have resulted in many different architectures. In the sections below, we describe and compare the architectures of a number of specific hypermedia systems. These systems can be regarded as representative of other systems developed within their particular domain.

7.2.1 Open hypermedia systems

Within the hypertext community, and especially in the Open Hypermedia Systems Working Group [177], interoperability with legacy applications is an important issue [70]. Typical examples of open hypermedia systems (OHS) include Microcosm [125], Hyper-G [146, 165], Chimera [12], and HyperDisco [239]. All these systems offer link functionality to "hypermedia-unaware" applications. To support linking in these applications, it is necessary to store links without corrupting the application's native file format. Therefore, open hypermedia systems are designed to support external links (see Chapter 2, page 42).

The focus on external links has a major impact on the architecture of open hypermedia systems, that are typically variants of the three tier architecture depicted in (a) of Figure 7.3 on the following page. To relieve new client applications from the burden of processing external links, and to allow legacy applications to make use of the hypermedia system's link services, all basic hypermedia functionality is implemented at the server side. To use the hypermedia services at the client-side, and to communicate to hypermedia-unaware applications, a small part of the OHS is assumed to be present at the client. This part is often called a *shim*, and is responsible for wrapping unaware applications and to perform protocol conversions when communicating with third party applications. Additionally, a minimal amount of client-side hypermedia functionality is needed to launch new viewer applications when currently running applications cannot display the destination of a traversed link.

Traditionally, the OHS hypermedia layer provides application-independent, navigation-based link services, strongly influenced by the Dexter Hypertext Reference Model (see Chapter 4). For some applications, however, Dexter-based navigation did not suffice, because these applications required support for application-specific link semantics. As a result, the hypermedia layer has been further decomposed, and components for generic, navigation-based linking have been separated from components implementing application-specific link management.

Currently, the protocol defining the interface between client and navigational components of the OHS link server is under the process of standardization [176, 240]. This protocol is also based on the Dexter model, but has been extended and adapted to fit the specific needs of an OHS architecture. The protocol increases interoperability, because it allows clients to use the link services provided by other hypermedia systems. Even more interoperability can be achieved if hypermedia systems can access each other's databases. This requires standardization of the interface between the hypermedia and data layer (see (a) of Figure 7.3 on the next page). Note that standardization of this protocol requires agreement about the basic hypermedia data model stored in the databases. While agreement exists for the application-level, navigation-based interface discussed above, the various systems involved still use different hypermedia models at

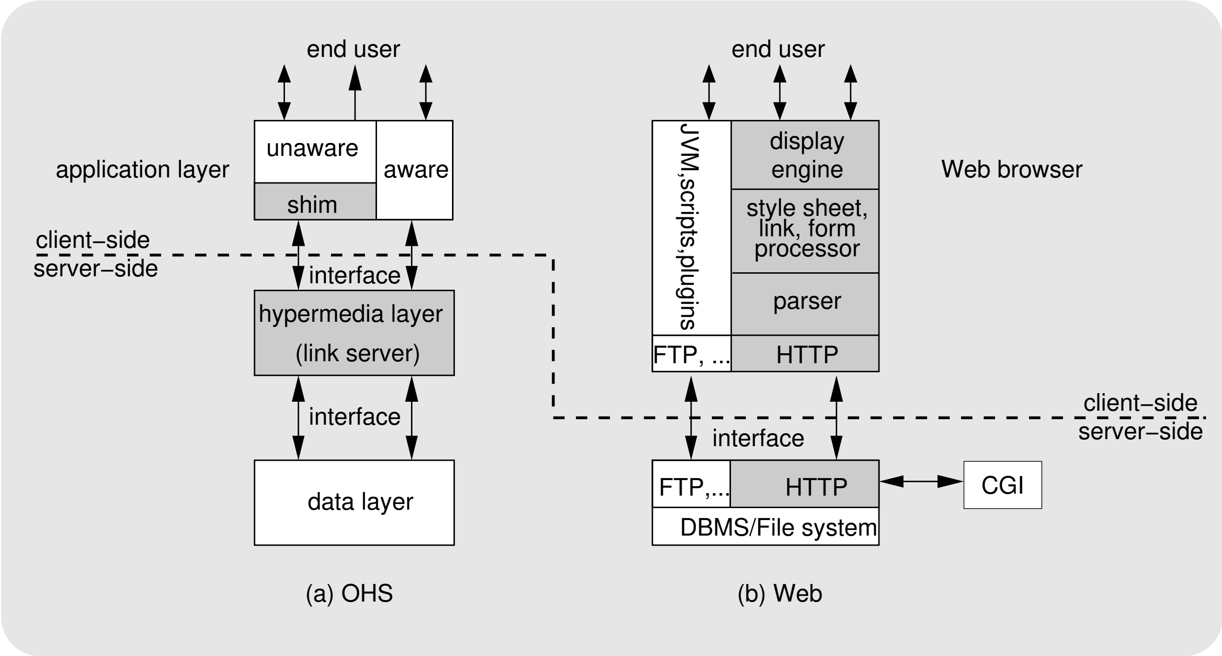

Figure 7.3: Open hypermedia architecture based on an open link server (a) versus the architecture of the Web, were hypermedia processing is done at the client side (b). The layers with core OHS and Web components are gray, external layers are white.

the data storage level [109].

From a reusability point of view, a major advantage of the OHS approach is that new client applications can easily be added. There is no need to add generic hypermedia components to these clients, because all generic hypermedia components reside on the server-side. In addition, OHS architectures can also be highly distributed because of the separation between the user interface, document storage and link processing. The OHSWG [177] advocates a further increase of the level of distribution and modularization through use of component-based technology.

In terms of interoperability, however, a major drawback of the OHS architecture is the central role of the link server: all authoring tools need to communicate with a link server in order to be able to manipulate links. In other words, users cannot add or modify hyperlinks unless they have access to a running link server, and other users cannot traverse these links unless they have (directly or indirectly) access to the link server that manages these links. In terms of flexibility, a drawback is that the addition of application-specific link behavior often requires modification of the server and the associated communication protocols.

The Web architecture described below does not have these drawbacks. In contrast to the OHS architecture, which can be characterized as a "thin client, thick server" architecture, a Web server can be relatively small. Most hypermedia processing is done by the relatively large client.

7.2.2 World Wide Web

Web applications typically use links that are embedded within HTML documents. The focus on server-side processing in the design of the open hypermedia architectures discussed above is especially suited for systems dealing with external links. Server-side processing of links has, however, major limitations when the majority of the links are embedded within the documents. First of all, it would require Web servers (or link servers) to parse the documents to extract the embedded link information. In the current Web architecture, new document formats can be added without the need to modify the server. Server-side processing of links, however, would make servers dependent on the format of their documents and would also introduce a significant performance problem.

In contrast, client-side processing of the link information (see (b) of Figure 7.3 on the facing page) involves hardly any extra computation-intensive processing. To be able to display the document, a Web application has to parse and interpret the document structure anyway, and can extract all relevant link information in the same run. Other advantages of client-side processing of the link information is that authoring tools can now manipulate documents and links independently from a server, and the server does not need to parse the documents at all. In the case of XML, this architecture is also sufficiently flexible to allow Web applications to perform the necessary application-dependent processing of links and other document structures.

A potential drawback, however, of this architecture is that a complete Web client needs to perform many tasks, and are thus typically fat applications. Future support for more advanced hyperlinking (e.g. external XLinks as discussed on page 94) may result in even more complexity in the client applications. Since the majority of the new Web specifications focus on client-side implementations, this is not likely to change in the near future. Fortunately, the architecture of Web-clients is becoming more and more modular and many of their components can already be reused in other applications.

Another drawback is that making existing applications interoperable with the Web typically requires a large amount of redesign in comparison to the OHS approach. This might involve, for example, changes to the native file format and data structures to support for HTML-like linking or adding HTML import/export functionality. The Web's initial focus on HTML and the absence of external links is, according to many researchers in the hypertext community, the reason why the Web does not qualify as an "open" hypermedia system: it is closed for any application whose internal document format does not or cannot support HTML's linking model.

From another perspective, however, the Web can be regarded as a system that is open and interoperable on all levels of its architecture. While the main transport protocol between client and server is HTTP, new protocols can be (and frequently have been) added without changing the overall architecture. Server-side functionality can be extended by employing the common gateway interface (CGI) and other techniques to make the functionality of existing applications (e.g. databases) available on the Web. At the client side, a variety of extension mechanisms are available by means of the browser's built-in Java Virtual Machine (JVM), plug-in interfaces and scripting environments. The drawback of having to adapt legacy applications still applies, but has become less important since conversion tools have become commonly available and many applications now offer HTML export functionality.

At the time of writing, the focus is moving from HTML to XML-based Web architectures. To a certain extent, such architectures give up the simplicity associated with having HTML as a single document format. From this perspective, it is interesting to compare the architecture of current Web applications with the architecture of SGML systems, because SGML systems were designed from the start to deal with multiple document formats.

7.2.3 SGML systems

Unlike the architecture of the Web, SGML architectures have never assumed a single document format. Another difference is that the client/server architecture that shaped many Web applications, never had a significant influence on SGML applications. An important objective of structured markup has always been to hide documents from the platform-specific details of the methods used to store them. SGML's concept of entities (see Appendix B) allows architectures to localize all interfaces to storage and access mechanisms in a single component which is known as the *entity manager*. For instance, the entity manager that comes with James Clark's SP [56] supports access to the local file system and HTTP servers, which gives all SGML documents (and their applications) transparent access to documents on a remote server. Because all access mechanisms are hidden by the parser, the distinction between client and server-side components has never had a major impact on the architecture of SGML systems.

Most SGML architectures, however, feature a dichotomy similar to the distinction between client and server components: the separation between generic components (i.e. the SGML parser) and application specific components. Until recently, only the parsing process of SGML documents was standardized. All processing necessary after parsing of the document was not standardized and as such considered to be application specific. This has led to expensive and fat client applications[1] (see (a) in Figure 7.4 on the next page). It also severely limited the potential for easy interchange of documents between different SGML systems, which was another important objective of SGML. The SGML-based Web browsers of the Hyper-G system, for example, had — due to this lack of standardization – to employ a style sheet format that was specific to the Hyper-G system [165].

This situation changed with the publication of the HyTime [134] and DSSSL [136] standards, which contained new concepts which allowed some common types of processing to be moved from the applications to the domain of generic tools. Both standards needed to operate on the data structure that resulted from parsing. However, this data structure was never defined explicitly, and caused major problems during

[1] The following chapter will introduce the DejaVu framework, which offers an application framework to facilitate more flexible development of the application-specific part of SGML systems in the realm of hypermedia applications.

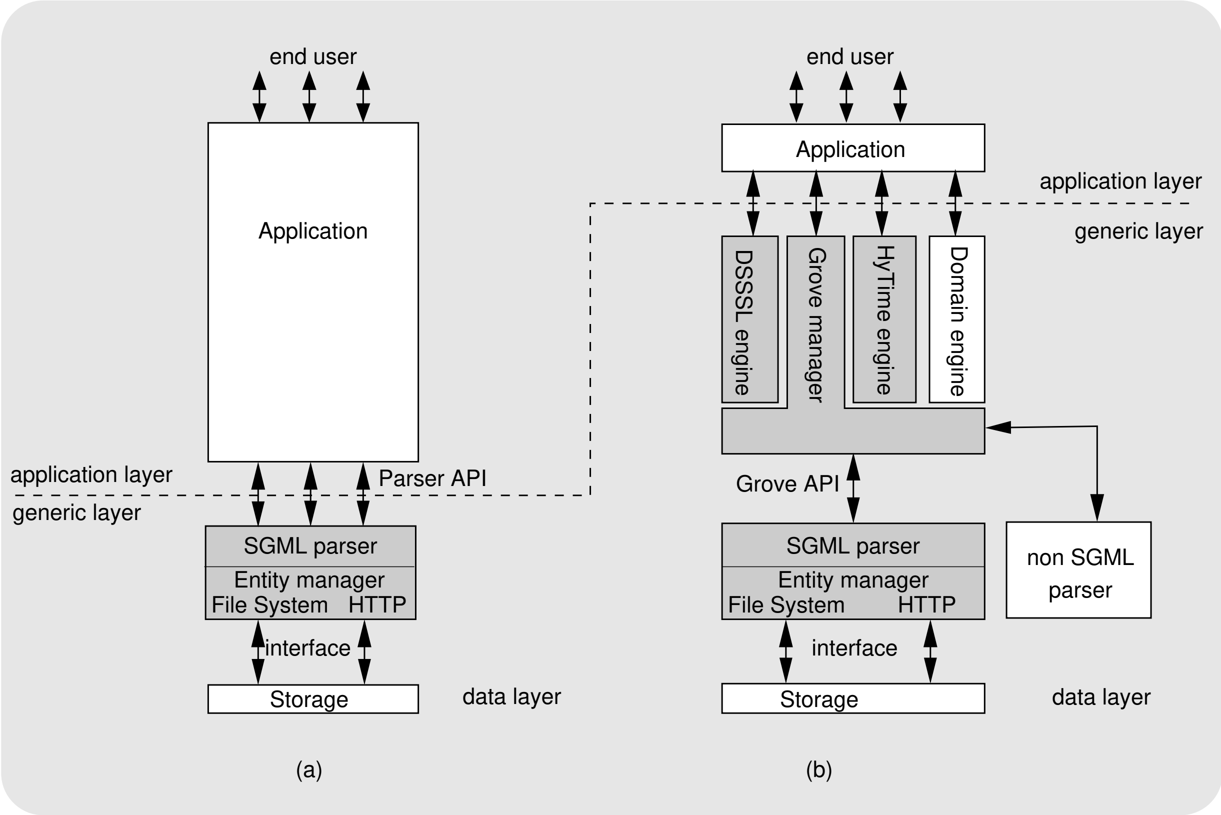

Figure 7.4: Application-specific and generic processing in a traditional SGML architecture (a) versus a (tentative) grove-based architecture (b). The generic components that are based on SGML related standards are grayed.

the definition of both standards. To solve these problems, the DSSSL specification formally defines the result of the parsing process as a data structure called a grove (see Appendix A). Currently, groves are also advocated as the basis for a common parser API, allowing SGML tools to use the same interface in combination with parsers from different vendors.

DSSSL relieves applications from two common but difficult tasks: transformation of SGML documents into other SGML documents, and formatting of SGML documents for printing and online display. Because these processes have been standardized, they can now be performed by off-the-shelf, generic SGML and DSSSL tools [57, 65][2].

HyTime standardizes many commonly usable hypermedia concepts, moving standard hypermedia processing code from the realm of applications to that of generic SGML and HyTime tools. While several systems implement (a small) part of HyTime's functionality within the application, more modular architectures have also been proposed [173]. These architectures (see (b) in Figure 7.4) become increasingly important

[2]Chapter 9 discusses the application of such generic DSSSL tools to the transformation of time-based hypermedia documents from one format into another.

for SGML systems that use HyTime's concept of architectural forms (see Appendix A) for domains other than hypermedia. These environments need to interface an SGML parser and multiple domain-specific modules with their application. While such architectures potentially relieve the client from many processing tasks, developing such reusable components has proven to be a rather complex process.

Many of the XML systems that are currently developed follow a comparable trend, slowly moving from architectures similar to Figure 7.4 (a) to the one depicted in (b). In the first systems, the only standardized component is the XML parser, while all other components are application specific. Over time, more and more components are being standardized and become easily reusable. Examples include DOM implementations (cf. groves), XSL transformation and formatting engines (cf. DSSSL) and components for processing XML Namespaces (cf. SGML architectures), XML Schemas (cf. SGML DTDs), XPath (cf. HyTime location addressing), etc. Note that Web-servers have become more complex because more and more Web-pages are currently generated on the fly. Many of the XML techniques described above are, in addition to their use in client applications, also used in the server-side document generation process.

7.2.4 Distributed multimedia systems

All arguments in favor of decomposition discussed above also apply to distributed multimedia systems. Unfortunately, the intrinsic characteristics of multimedia systems make the design of good multimedia architectures difficult. The prime characteristic of multimedia architectures is that they have to meet the time constraints that are needed to ensure adequate playback quality of the multimedia data. This means that the inter- and intra-media synchronization constraints of the multimedia documents (see page 50) need to be translated into the appropriate system level quality of service (QoS) parameters. This characteristic, combined with general performance requirements and the diversity of the large number of media types that are currently in use significantly complicate the design of multimedia architectures:

- Dividing a system up into relatively independent components and the physical distribution of these components typically has a negative impact on the performance and real-time behavior of the resulting system.

- Decomposition allows different media types to be processed by various, independently developed components. On the other hand, a coherent presentation of multiple media data formats typically requires a close cooperation between components (e.g. to realize tight synchronization requirements or to share screen real estate), a requirement that conflicts with the independent development of these components.

- Independently developed components typically use divergent interfaces (especially in multi-vendor environments), and a multimedia architecture needs to hide these differences from the application developer.

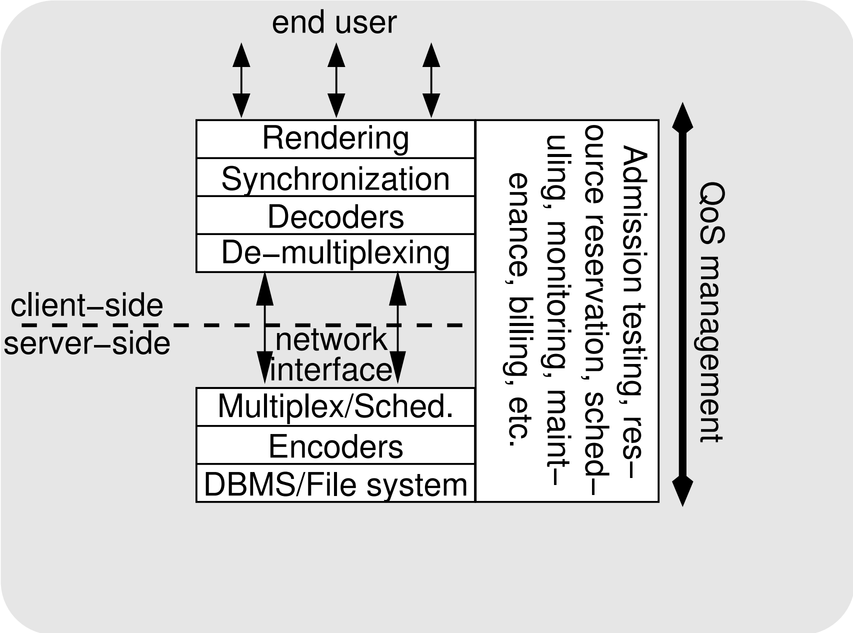

Figure 7.5: QoS management is needed in all layers of a multimedia architecture

- The need for hiding platform-specific details is also a requirement that often conflicts with the sheer storage capacity and computing power needed to process the multimedia data, demanding the utmost of the underlying hardware, network infrastructure and operating system.

Providing the appropriate application-level QoS is one of the most important objectives in distributed multimedia architectures. To achieve a high quality of service at the application level, all layers in the system's architecture need to provide real-time behavior and QoS management (see Figure 7.5). Architectures that are designed to provide such end-to-end QoS functionality are called QoS architectures (see [16] for an overview). Components of a QoS architecture need to provide interfaces to pass QoS parameters from one component to the other. During this process, components need to map lower-level, system-oriented QoS parameters to higher-level, application-oriented parameters and vice versa. Open QoS architectures allow several systems to communicate with and make use of each other's components. They thus require a high degree of consensus about the components' interfaces, including the QoS-functionality offered, the abstraction of the QoS specification, and the mapping of this abstraction to QoS specifications required by components in other layers of the architecture.

Note that on the Web, many multimedia applications are still based on an (HTTP-based) architecture similar to the one depicted in Figure 7.3 (b). Many SMIL applications, for instance, use this architecture, where only the HTTP layer is replaced by an alternative protocol such as RTP. As a result, the QoS level these systems offer is not comparable with the end-to-end architectures described above. Some SMIL implementations, such as GR*i*NS [51], only support "soft" synchronization, while others, such as

G2 [188] offer relative tight synchronization at the expense of large delays that result from the use of extensive buffering techniques.

7.3 Conclusion

This chapter provides a high-level overview of the typical software architectures used for open hypermedia systems, Web applications, SGML systems and distributed multimedia systems. In all four fields, architectures have evolved from closed and monolithic systems to open, modular and component-based architectures. In addition, the advantages of distributed hypermedia systems have made client/server architectures very common in all four domains, and recent developments indicate that the architectures of future hypermedia systems will employ more fine-grained decomposition mechanisms, for example by using distributed object technology. These developments will further increase the importance of standardized interfaces and protocols.

There are, however, also a number of important differences when we compare the Web's architecture with the other systems. Open hypermedia system design is moving in the opposite direction from the Web when it comes to embedded markup. For open hypermedia systems, the trend is clearly to move away from embedded document structures, and to store and process these structures independently from the document itself. This is the main underlying principle for many features of OHS architectures, and indeed many OHS applications depend on precisely these features. At the same time, on the Web, HTML is based on embedded markup, and the use of embedded markup will only become more common with the adoption of XML. Embedded XML markup is used more and more for embedding different types of structural information in the document's contents. Examples of such information types go beyond embedded linking and include the use of domain and application-specific markup languages, embedded meta data (e.g. RDF [155]), embedded timing (e.g. SMIL [231]), etc. Despite the advantages of OHS systems, in the short term it is unlikely to expect that Web applications will provide extensive support for both embedded and externally stored and managed structures [224].

When we look at the evolution of the XML family of standards and their implementations, we see many similarities with the evolution of their SGML predecessors. At first sight, it may even seem that XML is merely repeating SGML's history. In both cases, the generic markup language itself was established first, along with the required parser implementations. As more parsers became available, a need for standardized APIs arose. In addition, common functionality for document transformations, layout and formatting, link processing and temporal alignment became standardized. In addition, some of the problems of SGML-related standards still hold for XML-related specifications. A closer look at the XML-related specifications, however, clearly shows the differences between the two families. The lessons learned during the development of the SGML family of standards have considerably improved their XML-counterparts, if only in the reduced level of complexity. In addition, a typical SGML-based environment differs radically from its XML-based counterpart. While SGML is mainly used in complex applications, by large organizations who can (and need to) control most aspects of the document processing chain, XML architectures should also suit the needs of lightweight Web applications that are used by individual users or small organizations who can only control a small part of the entire chain.

In the discussion on distributed multimedia architectures, we showed that real-time and other QoS related behavior of hypermedia systems cannot be isolated in a single part of an architecture. As a result, incorporating such aspects into the Web may take more time than other aspects, because they need to be added to virtually every layer of the existing Web-architecture. Replacing transport protocols such as HTTP by real-time alternatives such as RTP will not be sufficient, as QoS management will also be needed at the higher levels (e.g. the other application components) and lower levels (e.g. the basic Internet protocols, operating systems and underlying hardware components).

The general discussion of hypermedia architectures in this chapter provides the background for the remaining chapters. In Chapter 8 and 9 we discuss the architectural issues addressed by two specific approaches to structured hypermedia processing on the Web. Chapter 8 describes the DejaVu framework, that applies SGML and style sheet technology to the processing of media-centric documents on the Web. Chapter 9 describes the Berlage environment, that applies document transformation technology to explore the different abstraction levels in time-based hypermedia document models. Finally, Chapter 10 summarizes the main results of the thesis and discusses the still outstanding research issues for processing time-based hypermedia.

Chapter 8

DejaVu: Object Technology for SGML-based Hypermedia

This chapter describes the experiences gained during the development of the DejaVu framework, an object-oriented application framework for hypermedia applications. The DejaVu framework addresses the major drawback of SGML systems that was discussed in the previous chapter. Where in most systems only the parser-related components are standardized, and the largest part of the architecture is considered to be application specific (see architecture (a) as depicted in Figure 7.4 on page 173), the DejaVu framework provides a reusable architecture for the application-specific part of SGML-based hypermedia systems.

From a hypermedia research perspective, the framework was developed to explore the interaction between structured hypermedia documents and the multimedia software components that could be used to present these documents. From a software engineering perspective, the framework explores the use of object-oriented design to combine and integrate the hypermedia functionality of a large set of heterogeneous software packages. The framework provides an interface to this functionality that suits both novice application programmers and more expert hypermedia researchers.

The chapter is structured as follows. First we discuss the concept of a framework and some other object-oriented software design terminology. Second, we provide an overview of the DejaVu framework and its key components and a description of the framework's multimedia extensions. We conclude with an overview of the lessons learned, both from a hypermedia research perspective and from an object-oriented design perspective.

8.1 Framework Terminology

In the object oriented literature, a *framework* is considered to be a generic application that allows the creation of various applications from an application (sub)domain [201]. A framework facilitates reuse on two levels: it provides both reusable *code* and reusable *designs*. The framework's code is reused through software components that are common to the applications of the target domain. From this perspective, a framework is similar to an object-oriented toolkit in that it allows developers to reuse pre-fabricated software components in their own applications.

A framework, however, is more than just a toolkit of reusable software components. A framework also defines one or more reusable software architectures by providing explicit *design guidelines* about how components can be combined and/or extended to build a particular application. There is, therefore, a strong correlation between object-oriented application frameworks and object-oriented *design patterns* as described by Gamma et al. [104].

A design pattern describes a problem to be solved, it names and describes a technique that might solve the problem, and discusses the context in which the technique works along with the associated costs and benefits. Design patterns provide developers with a shared vocabulary and a catalog of design techniques that have been successfully used in a number of applications. It is important to realize that patterns and components are on a different level of abstraction. Patterns can typically *not* be realized in terms of a set of reusable components.

Instead, patterns can be used to guide the *design* of the components in a framework and to *describe* and to *communicate* this design to the users of the framework. The reverse, however, also holds: many design patterns have been distilled after examining the design and documentation of a number of frameworks [142].

Another good example of the use of patterns can often be found in the way a framework caters for the *differences* between applications. Frameworks typically define *variation points*, or *hot spots* [201], to encapsulate the points in the framework where the applications differ from one another. A hot spot is implemented by a *hot spot subsystem*, which allows developers to plug-in components that are specific to their applications. To be able to do this effectively, a developer needs to be aware of the design patterns underlying the hot spot subsystem to make sure the plug-in components provided by the application can interact with the rest of the framework.

The remainder of this chapter describes the DejaVu framework. After providing a short background, the framework's hot spot subsystems and plug-in components are discussed in the context of the underlying design patterns.

8.2 Overview of the DejaVu Framework

The DejaVu framework is an object-oriented application framework developed in the early nineties at the Vrije Universiteit in Amsterdam [90]. The framework provides a flexible research environment for developing experimental hypermedia applications.

Although the framework serves primarily as a vehicle for research in hypermedia systems and object-oriented programming, the software components developed within the DejaVu framework are also used for educational purposes. The framework is used during practical courses in both software engineering and object-oriented programming, where it provides undergraduate computer science students the opportunity to gain hands-on experience with the application of hypermedia technology.

The design of the framework needs to reflect these two different types of users: application and system developers. Application developers use the framework by integrating the hypermedia components into their own applications. These developers are typically computer science students, with relatively little experience in hypermedia application development. In contrast, system developers that use the framework are typically experienced programmers, who are extending and improving the framework to experiment with new hypermedia and Web technology[1]. Supporting these two types of users resulted in the following list of requirements:

- **Support both stand-alone and Web-based hypermedia applications** — During the development of the framework, the Web gained importance. This changed the key subject of the framework — hypermedia user interfaces — from a rather exotic feature into a more commonly accepted basic programming technology. It also involved a shift from stand-alone hypermedia applications to networked, client/server Web applications. This shift requires the framework to support data access techniques other than the system's file system, most notably HTTP. It also stresses the need for separation between (server-side) hypermedia documents and (client-side) hypermedia applications.

- **Support experimentation with different document models** — While supporting the development of Web-applications requires built-in support for HTML, the framework should also provide an experimentation platform supporting research into different hypermedia document and presentation models. It should not be limited to a specific built-in model such as HTML.

- **Support a wide variety of multimedia software** — In addition to the common presentation capabilities (text and bitmap graphics) of the framework's GUI-toolkit, the framework continuously needed to incorporate new software packages to support other media types: two and three dimensional vector graphics, audio, video, music, HTML, etc. Integrating these software packages into the framework was hard because they all used different architectures, different programming styles, different application programming interfaces, etc. In addition, many media packages are developed as stand-alone applications with their own main event loop. Conflicts between the event loops of different software packages is a frequently recurring problem during the development of many hypermedia applications.

- **Support a wide variety of development tasks** — The development and implementation process of hypermedia applications includes many tasks, including developing media encoders/decoders, network transport protocols, GUI design, composing new components by combining other components, etc. These tasks differ significantly and each task can often be done best using a specifically designed

[1]Note that the boundary between these two types of users is not always clear cut: over the years a number of students evolved into experienced developers that contributed significant improvements and extensions to the framework.

(programming) language. In addition, different stages in the development process (e.g. prototyping early in the process versus speed optimization during the final stage) may also require different languages, as do programmers with different skills and personal preferences. To address these needs, the framework should not choose a single programming language, but offer an environment where many different types of programming languages can be mixed and used in combination within a single application.

- **Documenting and communicating design solutions** — After a problem has been solved, the solution needs to be documented. During the development of the framework it often proved to be hard — even when communicating to peer researchers — to explain a problem, its solution, when and how this solution should be used, the details of a specific implementation of the solution, etc. In addition, the requirement to make all functionality available to students makes providing good documentation even more important.

In order to meet these requirements, the DejaVu framework evolved over the years and underwent several redesigns. The framework is currently characterized by its three hypermedia-related variation points (hot spots), its multi-paradigm programming approach and a set of design patterns that guide the design of constructing and extending software components. These features are discussed in the following sections.

8.2.1 Hypermedia-related variation points

The requirements sketched above directly imply that the framework should give its users a high amount of support and flexibility when it comes to composing a graphical user interface (GUI) for their hypermedia application. The facilities for embedding window toolkits and media libraries, along with the framework's built-in set of common GUI and media components constitute by far the largest hot spot subsystem in the DejaVu framework, and its design has a major impact on the overall design of the framework, as is discussed in the next section.

A second hot spot deals with the required flexibility related to document processing, which will be different across applications and different document models. While it uses a generic SGML parser to be able to parse different document types, it hides much of the complexity of the SGML parsing process from the application programmer.

A third hot spot can also be easily identified from the requirements, and relates to functionality needed to access documents. While the entity manager of the SGML parser deals with access to SGML related resources, access to non-SGML resources on the Web, file system, etc is realized via the extensible set of access strategies of this hot spot. Again, it also serves to hide the associated complexity and differences between access protocols from novice programmers.

An overview of the framework and its hot spots is given in Figure 8.1 on the next page. Note that the hot spots related to presentation and access are also identified by Demeyer et al. [74], who describe an application framework based on the architecture

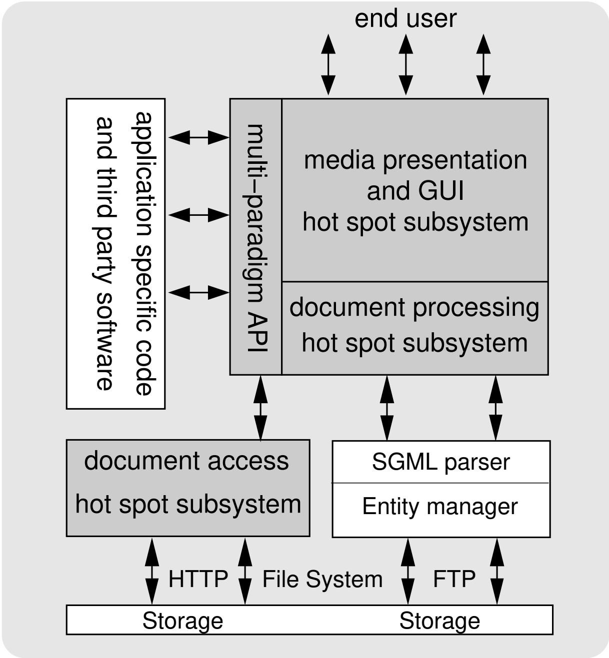

Figure 8.1: Overview of the DejaVu framework.

of open hypermedia systems as described in Chapter 7. The third hot spot defined by Demeyer relates to navigation. Within the DejaVu framework, the focus is on embedded links, which are considered to be an integral part of the document model. Variations in the link model are handled by the document processing subsystem, and have not been identified as a separate hot spot within the framework. Support for external linking, however, could be introduced by adding another hot spot to the DejaVu framework.

8.2.2 Multi-paradigm programming

To support different types of programming tasks, the framework supports different programming styles and languages, including object-oriented languages such as C++ [216] and Java [15], procedural languages such as C [147], scripting languages such as Tcl [179] and Python [161], and logic-based languages such as Prolog [36] and DLP [86]. The functionality of all built-in GUI components, for example, is accessible via interfaces for the languages C, C++ and Tcl. During the prototyping phase, DejaVu programmers typically benefit from the interactivity and high-level language constructs offered by scripting languages. Implementation of new, and extension of existing components is facilitated by features such as inheritance and information hiding of object-oriented languages. The C language is typically used for system-level programming and interfacing to third party software packages.

8.2.3 Design patterns for composition and extensions

Apart from its hot spots, the DejaVu framework has some additional facilities that simplify the composition of different predefined and new software components into a single, coherent application. First, the multi-paradigm approach also applies to extensions: functionality implemented by a new component can be made accessible from multiple languages (e.g. extension components typically have both a C++ class interface and a Tcl scripting interface).

Additionally, the framework deploys a number of design patterns that are targeted to hide the complexity of the many heterogeneous and composite components for the application programmer. For example, it simplifies integrating multiple event loops by providing a single, extensible event loop that is capable of dispatching a wide range of events, including application-specific event types (cf. the *Reactor* design pattern described by Schmidt [202]).

A pattern that deserves special attention is the *dynamic role-switching* pattern. It is used to implement two of the hot spots of the DejaVu framework in a way that provides the flexibility required by experienced users, and the simplicity that is required by inexperienced application developers. Dynamic role switching is a variation of the *Bridge*, *State* and *Abstract Factory* design patterns described by Gamma et al. [104]. The Bridge pattern provides flexibility by decoupling an object's interface from its implementation, and is one of the more frequently used patterns in object-oriented design. On the implementation level, the Bridge pattern is also known as the handle/body idiom (see also [66, 89, 183, 216]). The State pattern allows an object to change its behavior when its internal state changes, and such a state change frequently *appears* to change the class of the object. The Abstract Factory pattern provides flexibility by decoupling the information needed to instantiate an object of a specific class from the applications using the object's services.

Note that a Bridge interface object typically chooses one implementation object from a set of possible alternatives during its instantiation. In contrast, an interface object that uses dynamic-role switching provides an interface to the functionality implemented by a number of other objects (see the diagram in Figure 8.2 on the facing page). The interface object keeps track of all the roles by assigning a *role* name to each implementation object. The implementation objects need not be visible to the application programmer, typically only the interface object and the names of the roles are visible. The number of roles (and associated implementation objects) is not fixed, and new roles and implementation objects may be added dynamically to the interface object's role dictionary. When the interface object is assigned a specific role, it delegates all its services to the object which is associated with that role. The interface object can switch roles after an explicit request, or on its own behalf. The document presentation and access hot spots of the DejaVu framework are described in terms of the underlying dynamic-role switching pattern below.

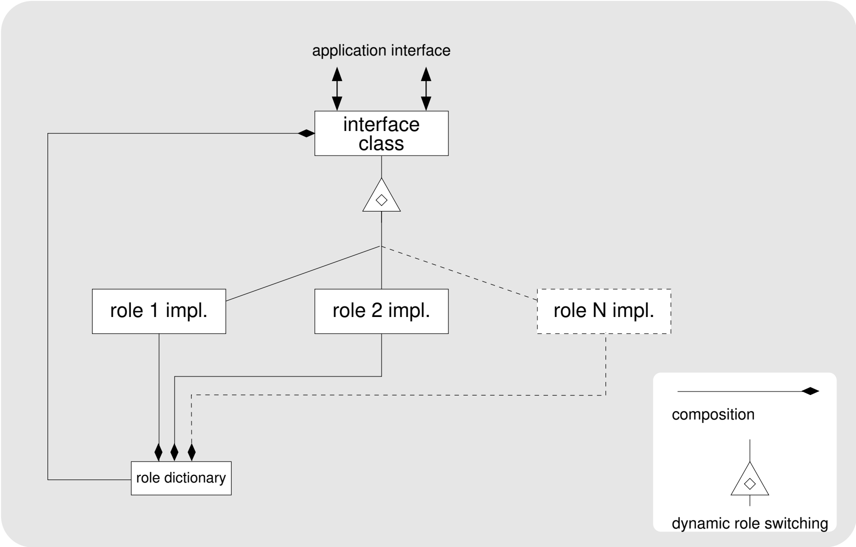

Figure 8.2: The dynamic-role switching pattern provides a single object interfaces to a set of implementation objects stored in a dictionary.

In the following sections, we will first discuss the components that implement the three hypermedia related hot spot subsystems, and illustrate their usage by discussing several plug-in components. The design and interaction of these components give a good insight into the type of applications that can be developed within the framework, and also provide good examples of the other features described above.

8.3 The DejaVu Hot Spots

The GUI and media presentation hot spot subsystem is the largest hot spot subsystem in the framework, since it provides a large and extensible set of readily usable GUI components. It is implemented by the hush toolkit, discussed below. Where the design of the hush toolkit reflects the research into object-oriented software design, the hypermedia-related research is mainly embodied in the design of the framework's Web browser named *eye*. The browser is in fact a reusable set of components built on top of the hush toolkit. From a hypermedia research perspective, the main motivation for developing the browser was to explore the ways in which the rich functionality of the hush toolkit and its many multimedia extensions could be made available to documents in a Web environment. The browser implements the two other hot spot subsystems of the DejaVu framework that deal with document processing and accessing.

Together, the hush toolkit and the eye browser provide the key components of the DejaVu framework. The remainder of this section gives an overview of the toolkit and the browser.

8.3.1 The hush toolkit

The hush (hyper utility shell) toolkit [87, 88] was originally developed by Anton Eliëns in the early nineties to bridge the gap between the two major freely available GUI toolkits for Unix: InterViews [157] and Tcl/Tk [179, 180]. InterViews featured a pure object-oriented class interface, but using the interface was complex to use, especially by inexperienced programmers. On the other hand, Tcl/Tk had an easy to use scripting interface, but no object-oriented API. The hush toolkit combines the flexible scripting interface of Tcl/Tk with an object-oriented API that was inspired in part by the InterViews toolkit.

Multi-paradigm programming

Hush provides an application with different interfaces to the underlying toolkit. This multi-paradigm approach makes hush particularly useful for developing hypermedia applications, where it is often convenient to mix different programming paradigms [90]. To access the functionality of the toolkit, programs may use, for example, hush's object-oriented C++ class API, the Tcl scripting language or the native C API of Tk.

The hush C++ class API provides a simplified and object-oriented interface to the basic functionality of the toolkit. It hides many details of the Tk toolkit to allow inexperienced programmers to use the toolkit in their own applications. During the prototyping phase, one often needs an alternative to the tedious edit, compile, link, run and debug cycle associated with application development in a compiled language such as C++. The Tcl scripting language provides an interface which is more suitable for prototyping. Tcl provides full and interactive access to the functionality of the Tk toolkit. Additionally, hush provides easy mechanisms to add new commands to the Tcl language in order to provide the same scripting interface to functionality implemented by C++ classes. Note that the Tcl scripting commands are also available from the hush's class interface, C++ components can also execute script commands. This feature is particularly important for the SGML-browser described in the following section, since the C++ browser components rely on Tcl as the basis for the style sheet language.

Finally, experienced hush programmers may bypass the limitations of the hush class interface or the Tcl script interface by using Tk's native C API. If necessary, these three approaches can be mixed within a single application.

Composition and extension facilities

Another feature which makes hush a suitable toolkit for hypermedia application development is that it has several facilities that simplify interfacing to existing, third party, software packages. Reuse of existing software is important for a hypermedia application framework, even if it were only to reuse the many software packages needed to process the media types that are in use today. The multi-paradigm approach of hush also proved to be effective as a gluing mechanism to provide access to software packages other than the Tk window toolkit. Several hush extensions have been developed by wrapping existing third-party software in C++ classes. By deriving them from hush base classes, these classes provide both an easy to use, object-oriented API and a flexible script interface to the functionality of the underlying software. Three other composition mechanisms that characterize the hush toolkit include:

1. **Integrated, extensible event loop** — A common problem when integrating different software packages is that many packages have their own main loop. Window toolkits, for example, typically have an event loop that dispatch the events generated by the GUI, while HTTP and CORBA servers loop to dispatch incoming requests from the network. Hush facilitates the integration of such applications by a common, extensible event handling mechanism that components can use to dispatch both system and application-defined events. The approach is comparable with the Reactor design pattern described by Schmidt [202]. Hush models a broad range of interactions between components by means of event objects. Not only user interactions are modeled by events, but data arriving on a network, the execution of script commands and timer expirations are also modeled by events. Additionally, the SGML browser described below also has an event-based API. All events are derived from a single *event* class, and developers can define new events to model application-specific interactions. Events are processed by event *handler* objects. All event handlers are derived from the same handler class. Most hush components are capable of handling at least one type of event, which explains why almost every class in the toolkit is directly or indirectly derived from the handler class. A handler object can be explicitly assigned to a specific type of event, this assignment is called the *binding* of the handler.

2. **Handle/body idiom** — The handle/body idiom separates the class defining a component's abstract interface (the handle class) from its hidden implementation (the body class). All intelligence is located in the body, and (most) requests to the handle object are delegated to its implementation. The handle/body idiom forms the basis of other, more complex, object-oriented idioms and patterns, of which many examples can be found in [66, 104]. The basic idiom is well suited for dealing with complex memory management [66, 216] and to allow an object's interface vary independently of its implementation (this is also known as the *Bridge* design pattern in [104]). Hush makes frequent use of the handle/body programming idiom. In fact, almost every class that is part of the hush API is in reality an empty *handle* class that redirects incoming requests to another *body* class. In addition, the idiom is used as the basis for the dynamic-role switching pattern discussed earlier and the *virtual self reference* idiom that simplifies the interface of composite objects.

3. **Virtual self reference** — When a component in a framework needs to be extended by adding functionality from other components it is frequently the case that within the new aggregate component, the original component remains responsible for carrying out the core functionality of the aggregate component. A typical example is a plain text-box object in a GUI toolkit that needs to be extended with scrollbars. A new composite component can be created to frame the text with the

scrollbars. The composite will need to redirect the majority of the incoming function calls, events etc. to the original text-object. The *virtual self reference idiom* is used by many components in the hush toolkit to automate this redirection in a way that can be controlled dynamically by the application programmer. It allows composite components to use one of its children in the part-whole hierarchy as an alternative body object for some of the functionality. A more extensive treatment of the handle/body and virtual self reference idioms is given in [90].

The hush toolkit provides a number of frequently used composite components which use the virtual self reference idiom. These components can be readily used by application developers. Most importantly, hush's object-oriented API also allows developers to use the same composition mechanisms to build application-dependent components that extend the functionality of the hush library. Extensions that have been developed this way include the SGML-based Web browser described below, a software sound synthesis package, a graphical music editor, a (networked) MIDI library, a discrete event simulation package and various extensions supporting digital video, 3D-graphics and virtual reality. The Web and music related components will be discussed later in this chapter.

8.3.2 DejaVu's SGML-based Web browser

The hush toolkit described above provides the basic functionality that all applications with a graphical user interface need: buttons, menus, list boxes, etc. Additionally, the multimedia extensions developed for hush provide readily usable components for displaying many media types. These components, together with the composition facilities and built-in extensible script language, provide sufficient support for developing prototypes of simple hypermedia applications. We have used hush to script several small hypermedia applications, which required relatively little programming effort.

Most of these applications were, however, governed by ad-hoc solutions for common hypermedia problems. Additionally, they were characterized by a fuzzy boundary between low-level, procedural application code and high-level, declarative document content. Frequently, script code was directly embedded in the hypermedia document, and fragments of the document were embedded in the scripts. Sometimes, the very distinction between document and application was nonexistent.

Obviously, more complex hypermedia applications require a more structured approach: the structure of a hypermedia document should abstract away from the low-level implementation techniques used for its presentation. At the same time, the abstractions used should be sufficiently flexible to support prototyping and experimentation with hypermedia document models that use the media extensions developed within the hush framework.

In addition, during the development of the DejaVu framework, the Web began to increase rapidly in popularity. It became clear that the framework should also support Web specifications such as URLs, HTML, HTTP and other specifications. In particular, there was a need for off-the-shelf components that offered easy access to the Web, so

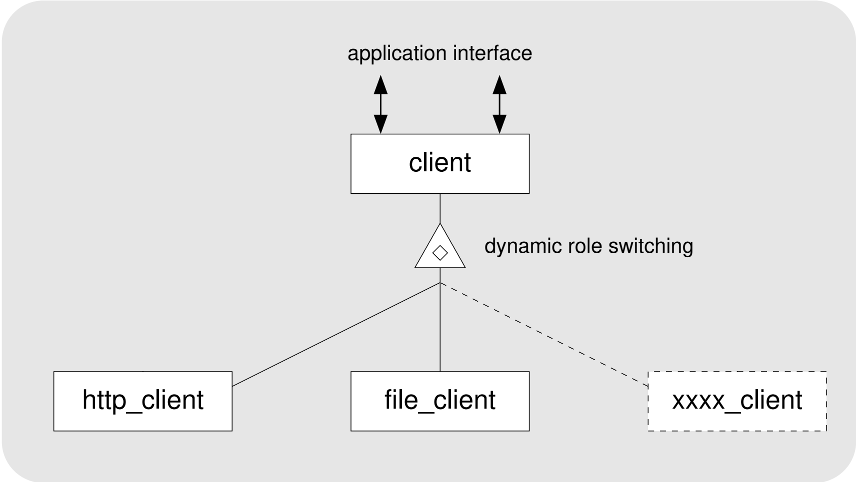

Figure 8.3: Dynamic role switching is used to provide a single client interface to alternative document access strategies.

that application developers could incorporate Web access and HTML rendering in their own applications. The components should also offer researchers the environment to experiment with document types other than HTML. These requirements bring us back to the two other hot spots of the DejaVu framework, that are implemented as part of the browser components.

The hot spot subsystem that implements the extensible set of document access protocols of the DejaVu framework (see Figure 8.3) is a typical example of the dynamic role switching design pattern that was discussed above (see page 184). The client class provides access to the classes that implement a specific file access protocol. Some of these classes are built-in (such as the implementations that access resources by means of the HTTP protocol or local file system), and can be readily used by the application programmer. New protocols can be dynamically added by passing the role name and an instantiated protocol implementation object to the interface object. A predefined role name is assigned to all built-in classes (here, the predefined roles are `http` and `file`). Instantiation of the built-in roles is lazy, i.e. the implementation object associated with a particular role is not instantiated before its interface object is required to play that role. The client can change the role it is currently assigned to after an explicit request from the application or, which is more common, it can autonomously switch to the appropriate role on the basis of a given URL. The dynamic role-switching pattern hides the details of several implementations — including the information needed to instantiate the objects associated with its built-in roles — from the application programmer. At the same time, it also provides the flexibility experienced system developers need to dynamically plug in their new components.

The document processing hot spot subsystem allows the framework to deal with new document models. In particular, we used the framework to experiment with extensions of the various document formats used on the Web. In general, extensions to

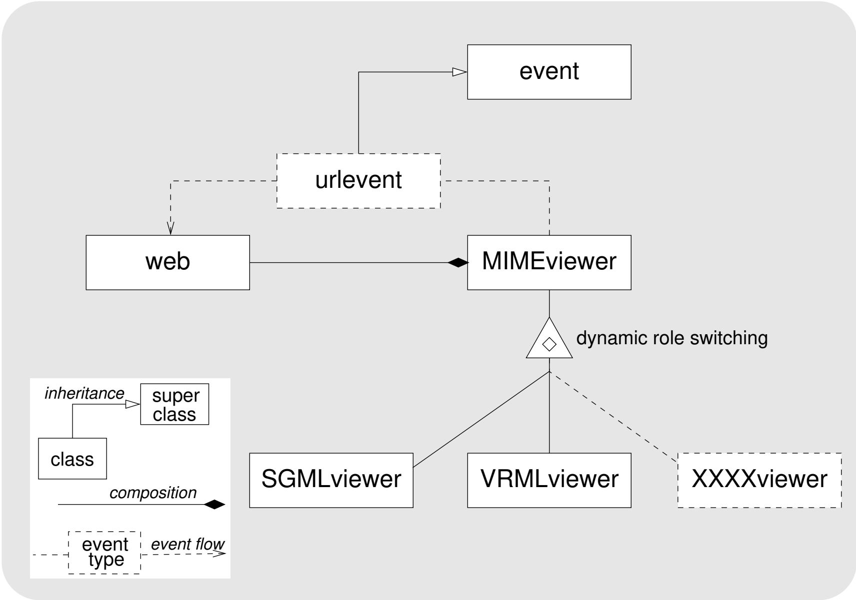

Figure 8.4: The MIME viewer component provides a single interface to viewers for several document types.

document formats can be useful for developers who want to add application-specific functionality to their Web documents. In our case, we explored extensions that allowed us to employ the rich set of multimedia components of the hush toolkit in a Web-based environment.

In order to provide both flexibility and a more structured alternative to embedded scripts, SGML[2] was chosen as the underlying syntax for most of the document processing components of the framework. This choice enabled the use of plain HTML documents, and also provided support for experimenting with extensions to HTML or even entirely new document types. Additionally, SGML provided the necessary abstraction mechanisms for hiding presentation details from the document structure.

Given the choices for hush as the basis for the presentation environment, and SGML for the document format, the document processing hot spot subsystem of the framework focused on a set of reusable components that together form an extensible SGML-based Web browser. The components hide the complex API of the underlying SGML parser and the APIs of the various hush extensions from the application developer. At the same time, it allows extensive experimentation. The browser uses an event-based SGML parser API and binds the events generated by the parser to a handler object that is responsible for processing and formatting the document. The handler object delegates this responsibility to fragments of Tcl code, defined in the style sheet associated with the document.

To allow document formats other than SGML, the actual parsing and display of a

[2]The choice for SGML was made in 1994, four years before the XML Recommendation was published. However, the research issues discussed here would still apply if XML had been used instead of SGML.

document is delegated to a special class, the *viewer* (see Figure 8.4 on the preceding page). The viewer uses the same dynamic role-switching pattern as described for the network client. Where the client used the name of the network protocol associated with the URL to determine its role, the viewer uses the MIME type of the target resource as the basis for its role names. As was the case for the network client, a number of roles are built-in (`text/plain`, `text/html` and `text/sgml`), and again a lazy instantiation algorithm is used for the creation of the associated implementation objects. New viewers can be dynamically added by passing the implementation object and the associated MIME type to the viewer interface.

Discussion

The development of the browser-related components of the DejaVu framework started with a pure HTML-based browser in 1993 which evolved into the SGML-based toolkit described above. Development ended in 1997. Over the years, the components have been used both by application programmers and system developers. The first group mainly needed to incorporate an HTML renderer into their own applications. They used the browser primarily for "conventional" tasks such as implementing help functionality and online documentation.

In general, the SGML parser is sufficiently robust to parse invalid HTML, so that most HTML documents can be processed by the SGML parser. To avoid requiring application programmers to provide a style sheet for HTML documents, the SGML viewer had a default style sheet for HTML, which could be explicitly overridden by the application developer or HTML document. Nevertheless, for the relatively simple task of HTML rendering, the heavy-weight SGML approach was often regarded as overkill.

The browser was also used by system developers to experiment with SGML documents other than HTML. For such documents, the style sheets needed to be explicitly developed by the developer. For these purposes, the combination of a tailorable and declarative SGML syntax with an associated operation semantics defined by a scripting language did indeed provide the flexible environment described by the initial requirements. Additionally, the framework provided three innovative features that are now familiar in today's browsers:

1. **Applets and HTML-embedded scripting** — The characterizing feature of the first version of the browser was the ability to embed and refer to executable (Tcl) code in an HTML page. The code was executed by the browser's interpreter when the document was rendered [220, 221]. This technique turned out to be very effective: it allowed every component of the hush framework to be embedded in an HTML page. It also emphasized the importance of the multi-paradigm approach: to be addressable from the embedded code, multimedia components developed in C++ also required a scripting interface.

   Obviously, the introduction of Java(Script) diminished the interest in Tcl as an applet/scripting language for the Web. However, many of the advantages and disadvantages of the use of Tcl are still true for the use of Java and JavaScript today.

A major disadvantage is that, for security reasons, the embedded code is typically processed by a "safe" interpreter. Such interpreters use a reduced instruction set or take other measures to prevent the execution of potentially harmful instructions. These security precautions, however, also prevent the execution of many potentially beneficial applications. In many cases, client-side executed code is now only used as an escape mechanism for achieving those dynamic and presentation-oriented effects that are beyond the features of the declarative models of HTML and CSS.

Another disadvantage of scripting is that the distinction between declarative markup and procedural code is blurred, which makes complex applications harder to develop and maintain. It also results in documents that are tightly bound to a specific platform. These drawbacks forced us to look for a more structured alternative to embedded scripting, where the division between declarative document markup and procedural code was easier to maintain. This required a shift from an HTML-only to a more generic Web browser.

2. **Extensible document syntax** — To allow structural multimedia extensions to HTML and the use of experimental hypermedia document types, we redesigned the browser from an HTML-only browser to a generic SGML browser [92, 226] (based on James Clark's SP parser [56]). Note that in general, there has been very little interest in generic markup languages from the Web community for a relatively long time (the few notable exceptions published in the international WWW conference series include [213] and the Hyper-G system [13]). This only changed after the publication of XML in 1998 [38].

   During the development of our SGML-based browser we encountered problems that are remarkably similar to those encountered several years later by the developers of the first generation XML applications. Examples of these problems include: the limitations of DTDs when it comes to developing extensions; namespace clashes when combining multiple DTDs; and the lack of appropriate standards for hyperlink functionality, interactive forms and multimedia style sheets and output formats.

3. **Style sheets** — While SGML provided us with sufficient flexibility on the syntax level, all these syntactic constructs needed to be associated with some operational semantics. To define the presentation semantics of our HTML extensions and new document types in a flexible manner, we chose not to implement these semantics directly into the browser but to employ Tcl as a style sheet language. Note that the choice for Tcl was made before the publication of DSSSL [136] and XSL [73]. However, even with these style sheet languages in mind, the use of a general purpose scripting language such as Tcl has some major advantages. In contrast to the text-flow oriented character of most style sheet languages (including DSSSL, XSL and the style sheet language of the Hyper-G system), Tcl allowed us to define the operational semantics of SGML document structures directly in terms of the functionality of the multimedia components in the hush toolkit.

The hush toolkit and the SGML browser components provide a platform to experiment with innovative multimedia applications on the Web. For these experiments, several extension components have been developed, including components that support integration of discrete event simulation [29, 91], digital video [221] and three dimensional graphics [93] into interactive Web-pages. In addition, many extensions have been developed to explore the possibilities of integrating music into the Web. As an example of the practical use of the three hot spots, multi-paradigm programming and design patterns, we describe the music-related DejaVu components in the next section.

8.4 DejaVu Extensions and Plug-ins

Several of the extension components that have been developed during the lifespan of the DejaVu project relate directly to music. While this domain certainly matched the personal interests of the developers, music has some additional properties which makes it a suitable domain for experimentation within the DejaVu framework.

First, the inherent complexity of the domain requires an incremental approach to the design and implementation process. With new functionality that was added, new third-party software needed to be integrated, new user and programming interfaces needed to be designed, existing components needed to be reconfigured, etc. This process provided, from a software engineering perspective, valuable insights in the flexibility of overall design and implementation of the framework.

Second, because timing and synchronization play such an important role in music, it is a good example to explore time-based media in general. In addition, timing also plays a role in the different software components involved, and integrating the different models proved to be a challenging — and partially unsolved — problem.

Third, music can be rendered acoustically (i.e. by synthesizing and playing the audio representation) and visually (i.e. by displaying a sheet music representation), so it is also a good example for experimenting with multiple delivery publishing for multimedia.

In addition, from a Web-application perspective, music is an interesting added feature. Music can significantly enhance the perception of hypermedia documents, especially in a commercial or educational environment. However, due to the relative high costs of good quality audio, music has for long been a rare phenomenon in networked hypermedia environments such as the World wide Web. Higher bandwidth networks, better compression and streamed delivery techniques have decreased the costs associated with audio. Still, dissemination of music based on techniques other than (compressed) audio streams has many advantages.

High-level encodings of musical scores, and even raw MIDI files, are usually a few orders of magnitude smaller, and the audio signal can be synthesized at the client side at any appropriate sample rate. Additionally, a high-level encoding of music provides the client with far more information when compared to the raw samples. Clients supporting intermedia synchronization might employ such information to provide high-level synchronization (e.g. "synchronize the start of the second scene of the video with the third measure of the intro tune"). Search engines and server back-ends may use the information to answer specific queries (e.g. select all tunes in 3/4 time and key C-minor).

This section describes the extensions to the hush media and GUI components that have been developed to experiment with the dissemination of music in a networked hypermedia environment by transmitting high-level descriptions of the musical scores and MIDI data, instead of the raw audio samples.

8.4.1 The Csound wrapper

The hush Csound wrapper was developed to experiment with client-side software sound synthesis in a networked hypermedia environment by transmitting high-level descriptions of the musical scores. Part of the material described here has been adapted from [222, 223].

Traditionally, sound synthesis is performed by dedicated hardware such as digital signal processors. Many modern personal computers can use such hardware to play MIDI encoded music, either through an external synthesizer or a sound-card with a MIDI interface. The DejaVu framework, however, was developed on Unix platforms, where MIDI support is less common. Fortunately, these workstations are sufficiently fast to make real-time software sound synthesis (SWSS) possible. SWSS does not need special hardware except for a digital to analog converter (DAC), found on every modern workstation. Because all synthesis operations are defined by software routines, SWSS is inherently flexible and well-suited for musical experimentation. In addition, the document formats that SWSS packages use to describe musical events are typically text based, which facilitates integration into a hypermedia application.

Csound is a SWSS package developed at MIT's Media Lab in the tradition of the Music V system. Scot [229] is a high-level, text-based notation to describe musical scores. Before the synthesis process, scores encoded in Scot have to be translated to the internal notation used by Csound. The tool used for this translation comes with the standard Csound distribution.

The components that integrate Csound and Scot into the DejaVu framework provide a good illustration of the typical design issues that need to be addressed for integrating third party software into the DejaVu framework.

First of all, the many mandatory parameters and configuration settings make Csound and Scot unsuitable for direct usage within an educational environment. Second, the underlying processing models do not match: the APIs of Csound and Scot are geared towards batch processing and do not satisfy the needs of an interactive hypermedia system. Third, the DejaVu framework's main event loop proved to be too coarse-grained for the specific requirements of this application, that is, generating audio samples at real-time speed.

These issues are addressed by a software wrapper, with an object-oriented, event-based interface[3]. The wrapper defines a convenient C++ interface that provides suitable

[3]With hindsight, this can be classified as an example of the *Adapter* pattern as described in [104].

defaults for the configuration parameters needed by Csound, which makes the functionality also accessible to inexperienced programmers. The wrapper classes implement all direct communication with the Csound and Scot applications. Higher level classes inherit this functionality and provide additional primitives to (re)play fragments starting at an arbitrary moment in time and to perform other useful operations upon these fragments. All details of the sound synthesis process are hidden behind the class interface of these high level classes. Inexperienced application programmers only need to use these components, and never need to interface directly with the lower-level wrapper classes. The wrapper can also be used without the hush environment, to extend other browsers or arbitrary C++ applications with sound synthesis functionality. In addition, they allow playback of small fragments where the standard batch-oriented interface focuses on the playback of complete files. Instead of using the framework's main event loop, the wrapper object runs Csound in a separate process with its own main loop. It allows applications to communicate via standard C++ I/O streams.

To be able to use the high-level Scot notation in addition to the low level note lists that are used by Csound, the batch-oriented interface of the Scot score translator is wrapped in the same way as the Csound program. Score fragments can be translated on-the-fly by the Scot wrapper and played by Csound. The Scot language is sufficiently powerful to denote most common note combinations (including chords, slurs, ties, triplets, etc), but the plain Csound note lists may be used as well. The set of instruments that is used to play the notes is described separately. Csound provides many operators which can be combined to define new instruments. While the wrapper provides a set of default instruments, applications may define and switch dynamically to other instrument definitions.

To allow the use of Csound and Scot from within the framework's hot spots, the wrappers have some additional features. Output from Csound and Scot is wrapped in event objects, allowing applications to use the same event handling technique to process this type of information. It provides run-time information about the way playing proceeds: how many notes have been played, which notes are being played at the very moment, how long it will take to play the rest of the notes, etc. More importantly, the wrapper also provides access to Csound's sound synthesis functionality via a scrpting interface. This makes it accessible from within embedded scripts, so that it can also be easily integrated with the Web browser described before. This integration allows interactive, real-time synthesized music to be embedded in an HTML (or SGML) Web page (as shown in Figure 8.5). In addition, it allows the use of music as part of an external style sheet. In this, our approach differs significantly from M.I.T.'s Netsound [54], which is based on a Csound player that is implemented as a stand-alone helper application, that is not integrated into the Web browser.

The approach sketched above is essentially based on encoding music in embedded scripts and external style sheets. We focused on the use of music to enhance Web documents that are encoded in general purpose document formats such as HTML. An alternative approach, however, is required when music itself is the primary type of information that needs to be described in the document. For such music-oriented applications,

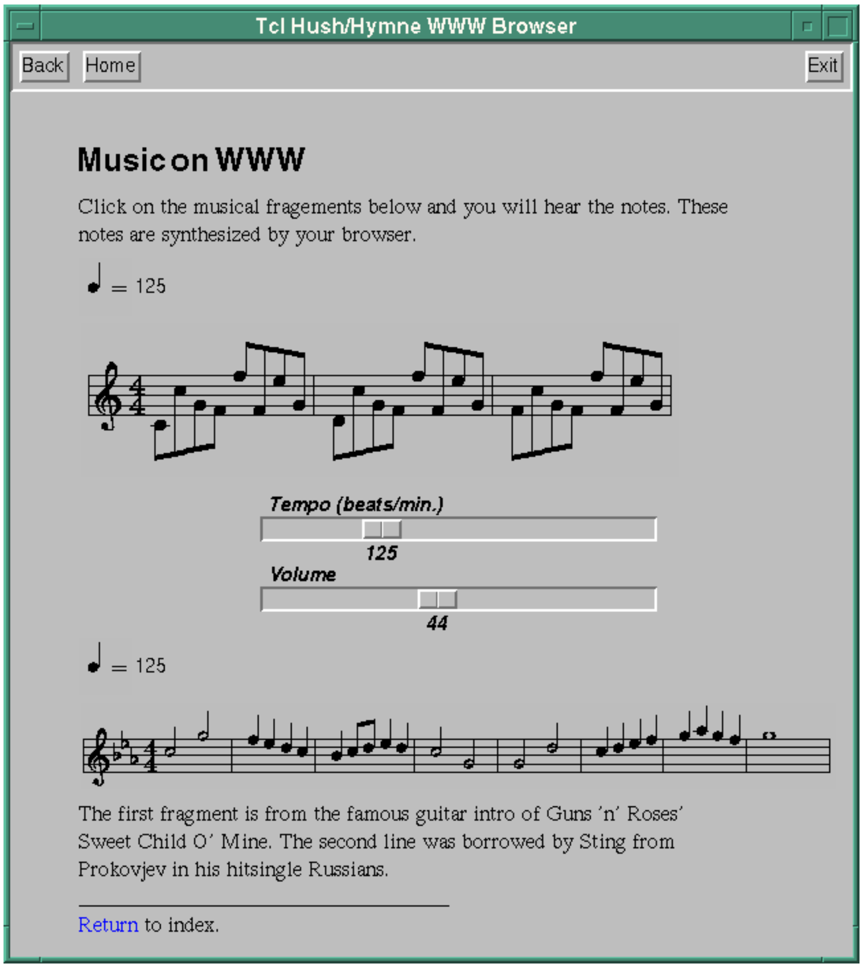

Figure 8.5: Interactive music embedded in an HTML web-page.

one could define a special purpose document format with a document model that is designed for representing music. Integrating such a format in a Web-based hypermedia environment is radically simplified when an SGML-based syntax is used. In comparision with raw audio files, this makes it relatively straightforward to use anchoring, hyperlinking and other standard document processing technology. A Web application that is based on this approach is the score editor described in the next section.

8.4.2 The score editor

While the Csound wrapper provides functionality for synthesizing sound from a high level, text-based description of a score, the editor components described here were developed to support the display and editing of such scores in a Web environment[4]. In contrast to the examples of HTML-embedded scores discussed in the previous section,

[4]The score editor was developed by Martijn van Welie and Bastiaan Schönhage. Part of the material described here has been adapted from [94].

```
<SCORE>
  <TITLE>Corrente</TITLE>
  <COMPOSER>Antonio Vivaldi</COMPOSER>
  <STAFF>
    <MEASURE Sig="3,4" Key=F Clef=Gclef>
      <NOTE Pos="1,3" Stem=down>d6 4 0
      <REST Pos="3,6">C6 8 0
      <NOTE Pos="4,6" Stem=up>a5 8 0
      <NOTETUPLE Stem=down>
        <NOTE Pos="5,6">f5 8 0</NOTE>
        <NOTE Pos="6,6">a5 8 0</NOTE>
      </NOTETUPLE>
    </MEASURE>
    ...
  </STAFF>
</SCORE>
```

```
SCORE {
  margin-left: 30;
  margin-right: 30;
  margin-top: 80;
  margin-bottom: 20;
  page-height: 1000;
  page-width: 920;
}
TITLE {
  title-align: Center;
  title-font: -*-Times-Bold-R-*;
}
COMPOSER {
  composer-align: Center;
  composer-font: -*-Times-*-R-*;
}
```

Figure 8.6: An example of an Amuse score encoded in SGML, with an associated style sheet in CSS.

the editor uses a dedicated SGML-based document format to encode the scores.

As shown by the example document in Figure 8.6, a straightforward SGML representation of a score is used, primarily geared to high-level editing, printing and displaying. The editor is, however, quite different from most generic SGML applications. Since it is quite hard to edit music in a generic SGML structure editor, users need a special-purpose graphical authoring interface. Music is one of the many domains that, although they can be very well encoded using SGML, they still benefit from a special-purpose graphical authoring interface. In addition, the specific typographic requirements of sheet music make it impossible to base the rendering of SGML-encoded music completely on standard style sheet methods.

Instead, the major part of the intelligence associated with the graphical rendering of the notes is hardwired into the application. The editor uses style sheets mainly as an additional technique to control some of the visual properties that are similar to text. A declarative style sheet language with a CSS-based syntax is employed to manipulate these properties, which affect both the screen display and the printed version of the score. Changes in the style sheet are dynamically reflected in the display of the score. An enlargement of the `page-width` property, for example, will allow for more measures on a single staff, and will result in a reflow of the complete score, similar to the way text would reflow. The editor's graphical user interface does not require users to have knowledge about the underlying SGML or CSS syntax.

The editor is also a good example of the use of the multiple delivery publishing model applied to a domain other than text. Despite its originally display-oriented design, the format is sufficiently rich to be able to automate the generation of playable MIDI and Csound representations from the score. Explicit interpretations of tempi, articulations and accents are, however, not supported in the current version.

Figure 8.7 on the next page shows the browser interface displaying a score. The figure also shows the dialog window associated with one of the supported playback modes. This generates a MIDI version of the score which can be saved on file or streamed

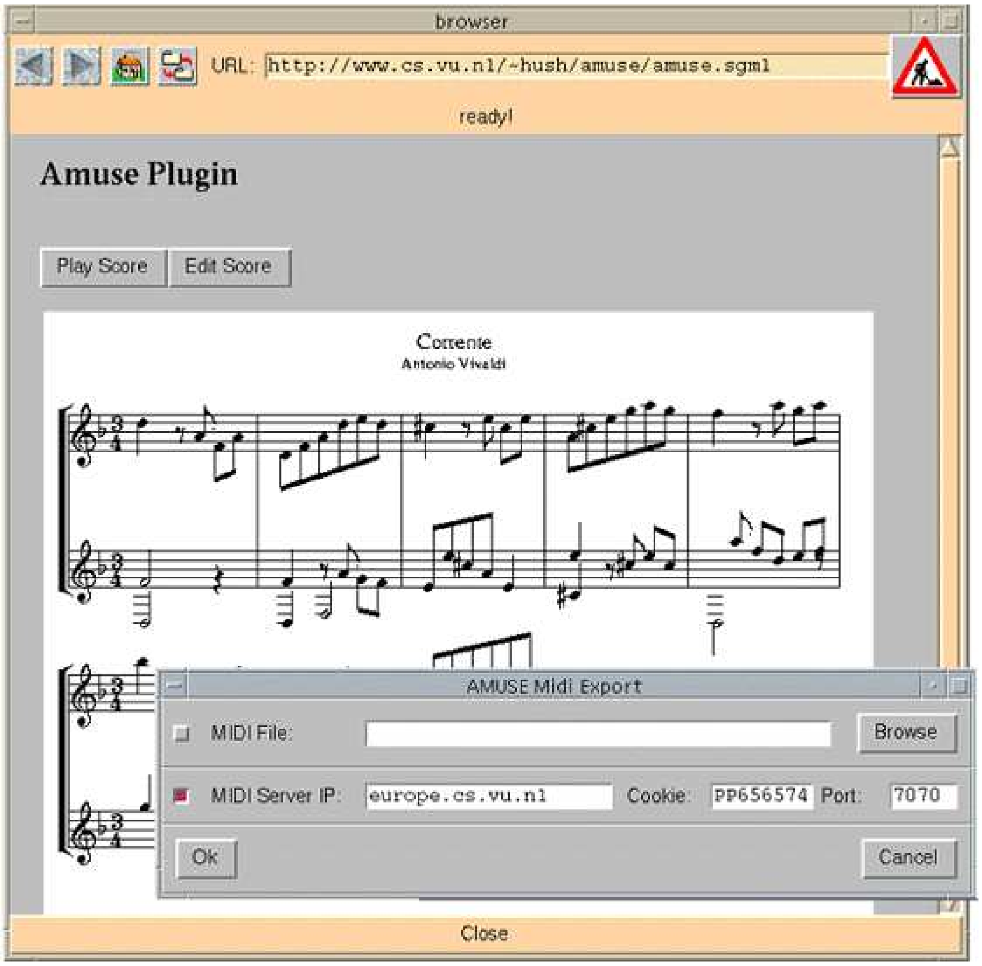

Figure 8.7: A musical score editor embedded in a Web page.

to a MIDI server over the network, as described in the next section.

From the perspective of the application developer, the score editor behaves like the other GUI components and can thus be directly plugged into the framework's GUI and media presentation hot spot. In addition, because it has its own document format, it can be used as a plug-in for the framework's Web browser. To be able to act as one of the alternative viewer roles in the framework's document processing hot spot, it implements the abstract viewer interface described on page 190.

8.4.3 The hush MIDI toolkit

To offer MIDI services to multiple, simultaneously running networked applications, the MIDI playback facilities of the DejaVu framework[5] are centered around a dedicated server as depicted in Figure 8.8 on the facing page.

[5]The MIDI components described here have been developed by Sebastiaan Megens. Part of the material has been adapted from [94, 166].

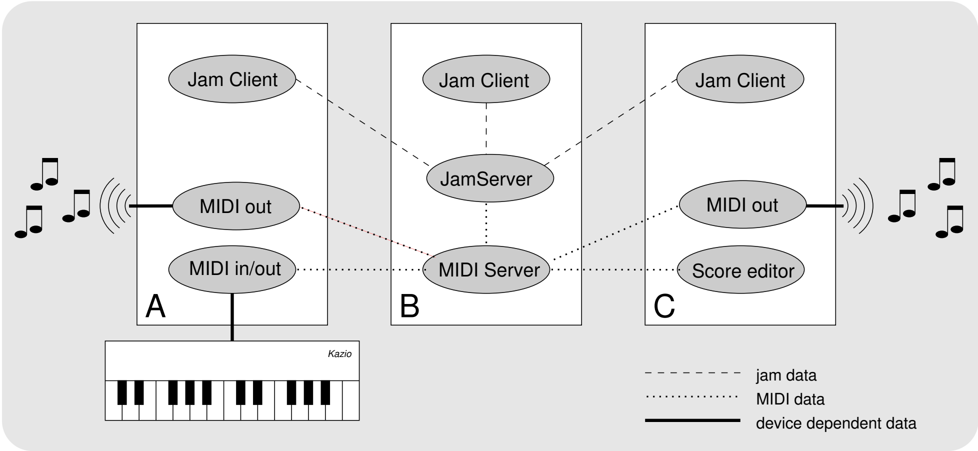

Figure 8.8: Overview of the special-purpose network connections between the MIDI components.

For this purpose, the network facilities provided by the framework's document access hot spot proved to be insufficient. The framework's document-oriented model did not match the session and streaming-oriented requirements of the MIDI components, needed to provide the necessary (real-time) cooperation between the various clients.

To use the provided services, client applications such as the score editor need to register as a MIDI-client, after which they may stream their MIDI data to the MIDI server over a UDP connection. Instead of using the framework's document access hot spot, the MIDI components build upon a third-party socket-level client/server library. The MIDI components provide a class library that implements the basic functionality for MIDI devices, MIDI clients and the MIDI server. Note that on many platforms, the audio device is an exclusive resource, and by connecting to a single MIDI server, several client applications can have simultaneous access to a single output device. The functionality of the MIDI server comprises:

- registering and unregistering MIDI devices,
- routing MIDI data between clients and MIDI devices, and
- administration and security checks.

When a MIDI device is registered, a cookie is given out that may be used by a client to request the server to set up a virtual connection with that device. The cookie also prohibits unauthorized clients from accessing a MIDI output device.

While the lower level components could not be easily integrated into the rest of the framework, this limitation did not hold for the higher level components that implement the main programming interface. For example, we used keyboard applets embedded in Web pages as alternative input devices to be able to send "live" MIDI data to the server. Since multiple applications can have access to the MIDI-server, a user can have a score edit session running, and simultaneously be playing a keyboard applet. To engage in such a "jam session", the keyboard applets connect to a *JamServer* instead of the MIDI server.

The JamServer, built on the same class library, acts as the central point of a jam session, keeping track of all clients engaged in the session. To start a session, all jam clients connect to a single JamServer. The JamServer is connected to one or more MIDI servers, as depicted in Figure 8.8 on the page before. By having the JamServer separate from the MIDI server itself, the latter is relieved from the burden of jam session management. Every connected MIDI device will receive all the MIDI data submitted by the jam clients. This data is relayed to these devices by the MIDI server(s), through the virtual MIDI data stream that is created when registering as a jam client. The figure depicts three jam clients connected to a single JamServer (on machine B). The MIDI server is running on the same machine as the JamServer. Both the clients on machine A and C have registered a MIDI-out device (a software sound synthesis MIDI program developed for Solaris) with the MIDI server on B. The user on A has additionally registered a MIDI-in device (the keyboard). Using the keyboard, the user on A can contribute to the jamming. The score editor on C is directly connected to the MIDI server and is not engaged in the jam session. The MIDI server will redirect MIDI request from the score editor only to the MIDI device on C.

The development of the MIDI components showed the framework's ability to deal with highly interactive documents. On the other hand, the integration of the lower level MIDI components revealed the limitations of the document-oriented network access of one of the framework's hot spots, which proved unsuitable for communicating the streaming MIDI data itself, and the session-oriented meta data needed to support multiple users.

8.5 Conclusion

Two main objectives have determined the design of the DejaVu application framework. First, the framework had to be flexible and extensible, because it was used in a research environment as a hypermedia experimentation platform. Additionally, it had to be easy to use, because it was also used for educational purposes by undergraduate computer science students, to give them hands-on experience with embedding hypermedia functionality in their own applications.

From an object-oriented design perspective, the framework applies a number of design patterns and programming idioms to integrate a large number of heterogeneous software components into a single framework. A large part of the design effort went into the (re)design of component interfaces that combined information hiding with sufficient flexibility. The framework provided student application developers with an overall architecture into which they could plug their application-specific code. This relieved the students from the need to design their own architecture. However, for many students giving up control over the overall architecture of their application was not easy and proved to be a significant hurdle in the learning process. In addition, some applications fitted well in the framework, but contained application-specific functionality that required its own architecture. The specific network requirements of the MIDI components, for example, made it necessary to implement application-specific network components and completely bypass the network functionality provided by the framework.

From a programming perspective, the framework's emphasis on interface design significantly reduced the programming effort during application development. In some cases, however, functionality needed to be sacrificed to keep the interfaces simple. For example, the Csound wrapper provided a simple interface to the underlying synthesis software, but did not allow other components to take advantage of Csound's sophisticated event scheduler. It also proved to be difficult to keep the interfaces up to date with the frequently changing underlying software. This was especially true for the API to hush's Tk widget set, which often needed to be extended to keep up with a new release of Tcl/Tk.

From a hypermedia research perspective, the Web components of the framework featured three innovative techniques that are now in common use on the Web. First, it used client-side computing by embedding executable code into Web documents, a technique that soon became common place with the advent of Java and JavaScript. Second, it used an extensible syntax for document markup. While the idea of using SGML as a meta-markup language to extend HTML was not new, most generic SGML systems were (and still are) text-based and not suitable for experimentation in a multimedia setting. Even at the time of writing, multimedia document types that use XML, such as SMIL, need special purpose rendering software that goes beyond the functionality of generic XML tools. Third, style sheets were added to the DejaVu framework as a means of mapping declarative document structures to the rich set of multimedia primitives of the hush toolkit and its extensions. While style sheets have always been applied within the SGML community, their use has mainly focused on text-based applications. The same applies for the Web, where style sheets became popular after the introduction of CSS. Both CSS, and more recently XSL, are based on a formatting model that is based on text-flow. By choosing a general purpose scripting language as our style sheet language, we avoided the limitations of a style language with a text-based formatting model.

The techniques described above allowed the creation of many hypermedia Web-extensions without committing to a specific document model. The main drawback of the approach was the relatively low-level programming work that was necessary in the form of developing SGML documents with embedded code and associated Tcl style sheets. For documents that required more complex processing, an approach that was based only on style sheets proved to be insufficient. For example, in the case of the score editor, we needed a fixed document model for the scores that were processed by special-purpose rendering components in C++. On the other hand, for applications that needed only simple HTML rendering, the SGML-approach was often regarded as overkill and better replaced by a lightweight, special-purpose HTML renderer.

In general, however, it was the lack of a common underlying time and hyperlink model that made it hard to develop more complex hypermedia applications. The various music components, for instance, all used their own technique for encoding music, and all techniques were based on a different timing model. To allow communications between these components, we relied on ad-hoc conversion techniques, which required a significant programming effort. The same applied to hyperlink functionality, which was underdeveloped due to the lack of a common link model.

In the next chapter, we look at the Berlage environment, which has been developed around a common hypermedia document model that includes support for both timing and hyperlinking.

Chapter 9

Berlage: Experiments in Format Conversion

In this chapter we describe the Berlage environment developed at CWI in Amsterdam. The DejaVu framework described in the previous chapter can be characterized as a hypermedia framework without any underlying hypermedia document or presentation model. Where the DejaVu framework is centered around a flexible output environment, exploring the use of different hypermedia document and presentation models, the Berlage environment is centered around a single, presentation-oriented document model: the Amsterdam hypermedia model (AHM) discussed in Chapter 5 and its implementation in the CMIFed authoring environment [228].

Additionally, the Berlage environment differs from the DejaVu framework and the architectures discussed in Chapter 7 in that it does not describe an architecture for a specific type of application or tool. Instead, the Berlage environment is an integrated collection of hypermedia document transformation tools, each implementing a specific translation. The tools converting from formats defined in SGML or XML are based on functional transformations specified in DSSSL. These tools use an off-the-shelf DSSSL engine with a typical SGML architecture (i.e. similar to architecture (b) as depicted in Figure 7.4 on page 173). The tools converting from CMIFed's native format, however, are built directly on top of CMIFed's parsing components. These parsing components have a proprietary API, and the mapping from the CMIF data structures provided by the parser to the structures of the target format are coded in a procedural programming language.

The main contribution of the Berlage environment, however, is not so much the resulting tool set and architecture, as the hands-on experience it provided us in dealing with multiple delivery publishing of hypermedia documents. This experience was the basis for part of the more theoretical work reported in Parts I and II. In particular, the Berlage environment provided valuable insights into the different levels of abstraction used by hypermedia document models.

The objective of this chapter is to focus on the more practical aspects of the Berlage environment, to discuss the possibilities and limitations of hypermedia document transformations in practice, and the trade-offs one has to make when support for different hypermedia formats and applications is required.

9.1 Introduction

Berlage is built around the Amsterdam hypermedia model (AHM). Most of the hypermedia features modeled by the AHM are implemented by the CMIFed authoring environment developed at CWI. Hypermedia documents that are authored using CMIFed are stored in a portable file format named CMIF. CMIF uses a structured, platform-independent encoding, so a hypermedia presentation can be authored once, and played back on multiple platforms. Another advantage of CMIFed is that both the file format and the authoring paradigm are based on the same hierarchical document structures that form the basis of the Amsterdam Hypermedia Model. This approach abstracts from the low-level timing and other presentation details, which simplifies authoring and maintaining complex hypermedia documents [124].

However, the approach also has a few disadvantages. First, while the CMIF file format is platform independent, playback of a CMIF document requires installation of the CMIF player. Second, neither the AHM nor CMIF separate presentation-oriented structures from other, more semantic document structures. This makes it hard to integrate the authoring environment into the multiple delivery publishing model.

To overcome these disadvantages, we explored several document transformations. To be able to play-back CMIF documents on applications other than CWI's CMIF player, we investigated the conversion to other, more standardized formats. We also explored the feasibility of automatic "down conversions" from CMIF to lower-level document output formats such as MHEG, and the "up conversion" from CMIF to HyTime. In addition to the use of standardized hypermedia document formats, we also evaluated the use of standardized document transformation techniques in the realm of time-based hypermedia. The conversions are implemented in a set of related tools which together constitute the Berlage environment (as depicted in Figure 9.1 on the next page).

Horizontally, the figure can be split in two halves. The left side depicts the software components and document formats that are specific to Berlage and CMIF. The right side depicts more generic third-party software and standardized file formats.

Vertically, the figure can be decomposed into four layers. The top layer represents the "HyTime layer". HyTime documents are used in two transformations: first, as the result of an up-conversion from CMIF, and second, as the starting point of a down-conversion to SMIL or MHEG. Note that we had no appropriate style sheet model, the dashed component in the top-layer indicates that this component has not been realized within the Berlage environment. For processing HyTime documents, we did not have access to a HyTime engine. We used the architectural form processing features of *SP*, an off-the-shelf SGML parser [56], to validate our documents against their DTDs and the HyTime and Berlage meta-DTDs.

The second layer is the "conversion layer", largely based on DSSSL, except for the

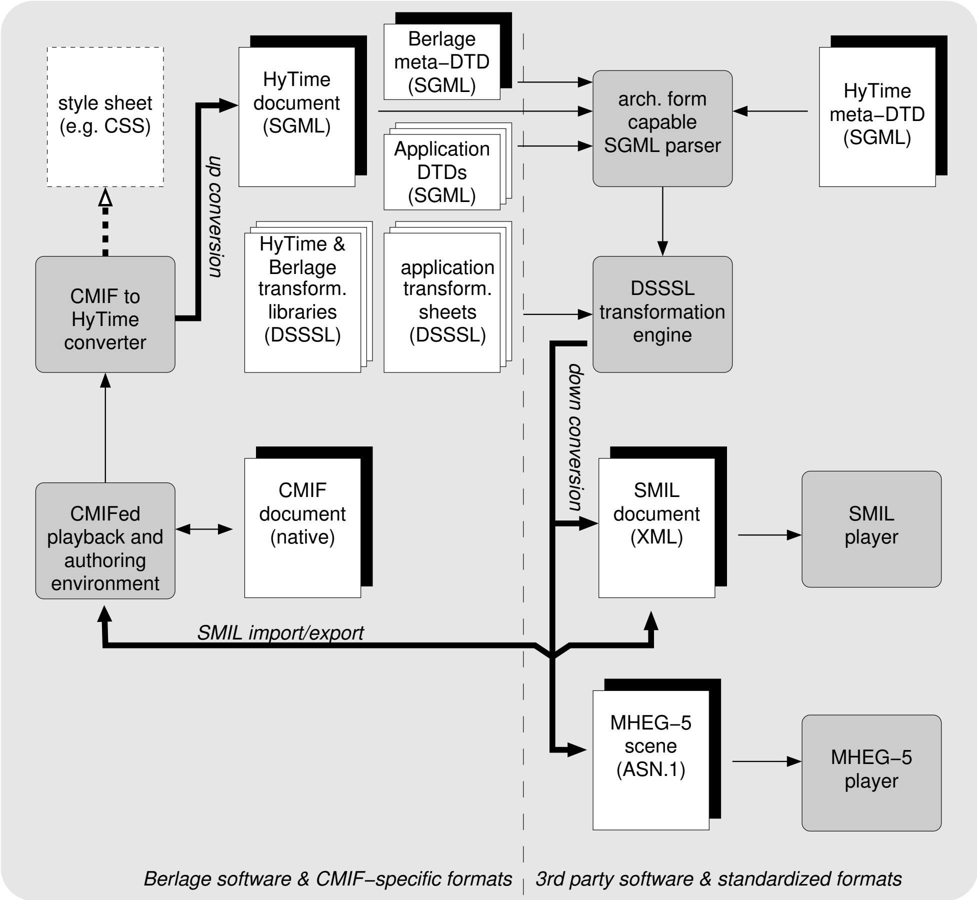

Figure 9.1: Document flows in Berlage. Conversions are depicted by the thick arrows.

CMIF to HyTime converter, which is based on CMIFed-specific software. For the DSSSL-based transformations, we used *Jade*, an off-the-shelf DSSSL engine [57]. We needed only a small amount of the functionality of a HyTime engine, this has been replicated in the DSSSL libraries used by the style sheets[1].

The third layer is the CMIF and SMIL layer. Note that CMIF and SMIL are sufficiently similar to allow fully automatic conversion between the two formats. These conversions have been implemented as an "import/export" feature of the GR*i*NS authoring environment.

Finally, the bottom layer represents the "MHEG-5 layer", which is only used as the result of the down-conversion from HyTime. We used a third party prototype MHEG-5 player to test our MHEG presentations.

Within the transformations sketched above, the AHM is the pivotal document model, which has two main advantages. In the case of conversions from other document mod-

[1]Note that Jade does not implement the DSSSL transformation language. To circumvent these limitation, we employed a commonly used technique, that allows the the DSSSL formatting proces to be used for transformation purposes. The technique is based on extensions to DSSSL which are part of the standard distribution of Jade.

els into the AHM model (i.e. both CMIF and SMIL), CMIFed provided us with a complete playback-environment that could be used to test the resulting documents. On the other hand, CMIFed also functioned as a convenient authoring environment for creating the source CMIF documents that were needed to test the conversion from CMIF to other document formats.

The sections below explore the following document conversions. First, we discuss the translation from CMIF to HyTime. This translation involves a mapping from AHM and CMIF structures to their more generic HyTime equivalents.

Second, we explain the mapping from concepts found in HyTime to those found in AHM, using SMIL as our output format. We deployed HyTime's facilities for rich semantic hypermedia markup in an example application, and wrote style sheets to convert a single HyTime document to different SMIL presentations.

Finally, we deal with transformations to MHEG. To be able to reuse the methods and tools developed for the HyTime to SMIL transformation we used the more generic HyTime documents as a starting point, rather than develop a conversion directly from CMIF to MHEG.

9.2 Extracting Structure: Generating HyTime

The conversion from CMIF to HyTime was originally driven by a practical requirement: the wish to have a more standardized interchange format for CMIF documents. From a research perspective, we hoped to gain insights about the abstractions underlying CMIF and the AHM model by mapping CMIF's document model to that of HyTime. In addition, despite the significant research in this area carried out by Buford et al. in the context of the HyOctane HyTime engine and its applications [46, 47, 48, 49, 191], when we started to look into the conversion in 1996, HyTime was still a new standard, and practical experience with implementing the standard was limited. And despite the name Hy*time*, it was precisely the time-based part of the standard that had hardly ever been used in practice. So the use of HyTime to express CMIF also served as one of the first practical tests for the expressiveness of HyTime in this domain.

In this section, we describe the development of the CMIF to HyTime transformation tool. It provides a number of insights into complex hypermedia document transformations, and clearly shows how multimedia document abstractions may differ in the different phases of the document processing chain. Before we can describe the transformation from CMIF to a HyTime document format, we first have to explain what a "HyTime document format" really is.

9.2.1 Developing a HyTime document format

HyTime documents use SGML syntax, and HyTime is defined by an SGML DTD. Nevertheless, HyTime is *not* a concrete hypermedia document format comparable to other SGML-based formats, such as, for example, HTML. In fact, HyTime is more comparable to SGML itself, in that it provides meta-level syntactical structures that can be used to *specify* a concrete document format.

Consequently, the DTD that defines HyTime plays a different role than most other DTDs, it plays a role that is often referred as a "meta-DTD". Applications do not use this DTD directly, but need to define their own, application-specific DTD, which uses constructs from both SGML and the HyTime meta-DTD.

This inherent meta-level aspect is one of the fundamental complexities of the HyTime standard, and we only gained a full understanding of its implications during the development of the Berlage environment. One of the more direct implications is that since HyTime only provides a meta-DTD, before a mapping to a HyTime-based document format can be developed, this document format needs to be specified. In other words, we needed a HyTime-based DTD for CMIF documents.

9.2.2 Developing a HyTime-based DTD for CMIF

The HyTime-based DTD needs to be able to capture the document model of (converted) CMIF documents. We developed this DTD with two goals in mind. First, to make as much of CMIF's hypermedia functionality accessible to other HyTime applications, the DTD should express this functionality in terms of HyTime architectural forms. Second, the DTD should capture the CMIF semantics as closely as possible, so no important information is lost as a result of the mapping. Given the document model defined by the DTD, a mapping can be defined from the original CMIF concepts to those defined by the new, HyTime-based DTD. The Berlage transformation tool that automatically converts native CMIF documents to their HyTime-based equivalents is based on this mapping.

In theory, development of a transformation tool is best done after the mapping between the source and destination format has been defined. However, the development process of a complex DTD and the mapping to such a DTD is significantly simplified if one can, in every phase of the development, work from concrete examples of original documents and their HyTime equivalents. For this reason, it is extremely useful to start with the prototype of the transformation tool *before* the DTD and the mapping have been fully defined. We thus used an incremental approach and simultaneously developed the DTD, the mapping and the associated conversion tool. We gradually extended the DTD to capture more of CMIF's hypermedia semantics, and subsequently prototyped the associated mappings in the transformation tool.

The development of the DTD and the tool raised a number of concerns that are also relevant for other applications that consider the use of HyTime constructs. In particular, we found that HyTime was suited to represent almost none of the information modeled by the AHM:

- **Style** — Because HyTime is not designed to represent style information, style information cannot be expressed in terms of HyTime architectural forms. To prevent the loss of style information when documents are converted to HyTime, style information can be collected from the source document and encoded in a style sheet separate from the target HyTime document. Note that the HyTime specification does not specify how HyTime applications should process associated style sheets, nor does it standardize a mechanism to refer to a style sheet from within a HyTime document.

- **Presentation layout, timing and linking** — HyTime's rich facilities for spatial and temporal alignment and complex hyperlinking made us decide to use HyTime architectural forms to represent the basic scheduling and synchronization aspects, the basic spatial layout and all hyperlink structures of the AHM. For all of these concepts, however, one can argue that they are presentation specific and should be encoded in the style sheet, not in a HyTime document. In fact, most of the more interesting run-time aspects of hypermedia can not be represented in HyTime. We will return to this issue below.

- **Semantic annotations** — The AHM supports semantic annotations on almost all of its components, and uses simple attribute/value pairs to encode this type of information. Because neither the attributes nor their values have been further standardized, it is not possible to automatically convert these annotations into meaningful HyTime structures.

- **Other hypermedia structures** — Of the other AHM constructs we can not represent in HyTime, some are specific to our CMIF implementation and hence of little interest to other applications. We also found, however, concepts that were sufficiently broad to be useful for document models other than the AHM.

  For example, the strength of many of the timing facilities found in CMIF relies on the ability to adapt to information that is only available at runtime. In this way, the presentation of the document is automatically adapted to deal with network delays, resized window's, etc. Since HyTime abstracts from these runtime aspects, this adaptive behavior could not be encoded using HyTime.

  To define commonly usable SGML equivalents for this type of concept, we have taken the same meta-DTD approach as HyTime, and have defined these concepts as architectural forms in the Berlage meta-DTD. Berlage can be seen as an extension of HyTime that adds common hypermedia abstractions that are, from the perspective of HyTime, too much oriented towards run-time presentation. Examples include structures to extend HyTime links with link context, support for the behavior of CMIF's atemporal composition features (e.g. CMIF's choice node) and the description of media alternatives (e.g. SMIL's switch element), and facilities to use the intrinsic size and duration of a media object in a HyTime schedule[197].

9.2.3 Discussion

The original objective of the design of a HyTime DTD for CMIF was to provide a more standardized alternative to CMIFed's native file format. During the development, we learned that while both the AHM and HyTime model can be used to model space, time and link structures, the two models address different levels of abstraction. In fact, the transformation from CMIF to HyTime is an up-conversion, in that it tries to extract higher level abstractions from a document that is defined in terms of lower level concepts. In general, up-conversions can only be automated to a limited extent. In addition, up-conversions are also limited by the fact that the target model is often not designed to capture the more lower level aspects that are present in the source document. Both limitations apply to the CMIF to HyTime transformation described above.

In terms of timing and spatial alignment, HyTime cannot — by itself — express many features that are needed for multimedia synchronization and layout. While this may seem surprising, it is a direct result of HyTime's high level of abstraction. HyTime does not deal with many types of lower level run-time, presentation, application and user-specific information. This is especially confusing for timing and spatial alignment, since for multimedia, these dimensions are traditionally directly associated with a document's presentational behavior, and not with the abstract structures underlying the document's content[2]. Even in HyTime, however, the line between abstract structure and presentation is not always clear: HyTime's links, for instance, model presentation-oriented information such as the traversal directions that are allowed during the presentation of the document.

In contrast to the CMIF structures we could not encode in HyTime, HyTime offers many hypermedia structures we could not generate automatically because of the presentation-oriented nature of CMIF. So the transformation suffered from two fundamental problems:

1. The most interesting features of CMIF cannot be represented adequately in HyTime. Because SMIL's features are much closer to those of CMIF, this limitation can be addressed by transforming CMIF to SMIL instead of HyTime.

2. The most interesting features of HyTime cannot be generated automatically from CMIF. This limitation can be addressed by authoring the source HyTime documents by hand, and use HyTime's more abstract hypermedia structures as the source, instead of the target of the transformation.

Both approaches are discussed in the next section.

9.3 Using Structure: Generating SMIL and MHEG

During the development of the Berlage environment, our group at CWI became involved in W3C's synchronized multimedia working group, the group that developed SMIL. When the work on the first version of SMIL ended in 1998, and third-party SMIL applications became available, we started to use the new SMIL format for two purposes:

[2]There are only a few domains in which we are comfortable with highly abstract notion of timing, and where we are used to the fact that these abstract structures need to be mapped to "real-time". The typical example is music, the domain that originally started the HyTime effort.

as a standardized interchange format for CMIF documents and as an output format for HyTime documents.

9.3.1 SMIL as an interchange format for CMIF documents

Many of the concepts underlying CMIF found their way into the SMIL specification, along with several other improvements to provide better integration into the Web's infrastructure. The new features defined by SMIL were added to the authoring environment, which was renamed to GR*i*NS (GRaphical *i*Nterface for SMIL) to reflect these changes. Rapidly, SMIL, and not HyTime, became the proposed standardized interchange format for CWI's authoring environment.

For this purpose, SMIL proved to have many advantages over HyTime. First of all, HyTime and CMIF are very different when it comes to presentation-oriented and run-time related issues. Here, SMIL and CMIF are on exactly the same level. Secondly, to test the validity of an interchange format, one has to do interoperability tests. This was, due to the lack of HyTime implementations, impossible. As third-party SMIL players became available, we could finally evaluate to which extent other applications could playback documents authored in CMIFed, and test the validity of SMIL as an application independent multimedia interchange format.

The main issues that came up during playback of files that where converted from CMIF to SMIL 1.0 can be summarized as:

- **Limitations of the SMIL 1.0 functionality** — While a large part of the CMIF functionality found its way into the SMIL 1.0 specification, some CMIF functionality cannot be expressed in SMIL. More complex CMIF documents often depend on this functionality and can thus not be played back by standard SMIL players. This applied in particular to documents that made frequent use of CMIF's choice composition technique and implicit link context.

- **Limitations of the SMIL 1.0 specification** — Some unexpected behavior of SMIL documents on other players was caused by under-specification or error in the W3C SMIL 1.0 Recommendation [127]. Examples included different default values for background colors and ambiguities in the timing model. Most of these limitations had only a minor impact or were easy to work around.

- **Limitations of the SMIL 1.0 implementations** — Other problems were caused by incomplete or erroneous SMIL implementations. These problems applied in particular to the first generation of SMIL players and became less serious as newer versions appeared over time.

- **Media support limitations** — Most serious limitations were caused, however, by the different sets of supported media types across different applications. SMIL 1.0 follows the HTML standard in that it does not specify a list of media types for which support is required. For HTML, this is only a limited problem since most HTML pages only include objects of a few media types (e.g. JPEG or GIF images).

> Despite the fact that support for these media types is not required by HTML, almost all HTML browsers support them in practice. Unfortunately, the situation for SMIL is quite different. Using many media types is much more common for multimedia, so a typical SMIL document uses a much wider variety of different media types. Additionally, the players of the first generation all supported a different set of media types — to the extent that it was virtually impossible to design a single SMIL document that would play on more than one platform. While this situation has improved since the publication of SMIL 1.0, the issue is, at the time of writing, still one of the major problems with SMIL interoperability.

So in general, the CMIF to SMIL translation made it possible to play back CMIF documents on players other than CMIFed. The conversion, however, did not deal with CMIF features that are not part of SMIL 1.0. In addition, it needed to take into consideration the capabilities of the specific SMIL player the transformation targeted. These considerations included media type conversions, adding explicit default values and other workarounds.

9.3.2 SMIL as an output format for HyTime-based documents

In addition to the use of SMIL as a standardized format for our authoring environment, we also explored the generation of hypermedia presentations from presentation-independent documents encoded in HyTime, where we used SMIL as the final presentation format [71, 193, 194, 199].

The conversion from HyTime to SMIL is different from the conversion to from CMIF to HyTime discussed above. First, by authoring the HyTime source documents by hand (in an ordinary text-editor), we could fully exploit HyTime's facilities for presentation-independent markup. Second, the SMIL presentations that resulted from the conversion could be readily tested on various SMIL play-out environments. The conversion also has an important advantage over the conversion from CMIF to SMIL. By mapping HyTime's presentation-independent markup to SMIL in a style sheet, we could generate multiple presentations from the same underlying document, and thus experiment with the multiple delivery publishing model in a hypermedia context.

For the conversion from HyTime to SMIL, we used a relatively small hypermedia application as an example. Our source HyTime documents contained information about historical buildings in Amsterdam (a more detailed description of the example and the implementation of the DSSSL-based transformation is given in [198]). Again, we used HyTime's scheduling and alignment facilities to encode the spatio-temporal information associated with the buildings. Rather than encoding their position on the screen, or the time they appear in the presentation, we used HyTime to encode intrinsic spatio-temporal attributes of the document's content, such as the street address and year of construction of the various buildings. Additionally, we used HyTime links to encode several semantic relationships between the media items associated with the buildings.

Given these conceptual time, space and link structures, we used various style sheets to map these structures to SMIL's presentation-oriented time, space and link structures.

While spatial structures that are described on a conceptual level in the document can be mapped onto spatial structures of the presentation, many other mappings can be defined. Examples of all nine possible mappings between time, space and link structures on the two different levels are given in [199][3].

These conversions helped to gain an insight into how the fundamental differences between HyTime's conceptual time, space and link structures, and their presentation-oriented counterparts in SMIL and the Amsterdam Hypermedia Model could be practically used to generate different multimedia presentations from a single source document. They made clear which types of presentation information can or need to be added when generating a concrete presentation from an abstract HyTime document, and how the application of style sheets to multimedia differs from their application to text.

To explore the extent to which our conversion tools were applicable to hypermedia formats other than SMIL, we also explored conversions from HyTime to MHEG-5.

9.3.3 Final-form Multimedia: Generating MHEG

The major differences between the two conversions relate to the more final-form nature of MHEG-5. In addition, MHEG-5 is targeted at players with a relatively small footprint. As a result, there is a certain amount of processing that, in contrast to SMIL where this is left to the player application, needs to be carried out by the authoring tools that generate the MHEG presentation.

As MHEG presentations are composed of different scenes, the translation involves conversion from a single HyTime document into one or more MHEG-5 scenes [196, 217]. Each scene is composed of a set of media items. In contrast to SMIL 1.0 timing, dynamic behavior in MHEG-5 scenes is event driven. Play-out of media items can be triggered by using timer events. Users can move from one scene to another by using hyperlinks, which can be triggered by (predefined) user interface events. Hyperlinks can also be used for scheduling purposes by firing links via timer events.

We used the same DSSSL-based transformation tools as for the conversion to SMIL, and separated DSSSL code that was developed for the transformation to SMIL but was reusable into DSSSL libraries that were subsequently used in both transformations. The reusable code was further divided into a generic part dealing with SGML and HyTime structures and a part that was specific for the concrete HyTime DTD used.

While the transformation sheets could, at least in principle, derive all the information needed for the transformation to MHEG from the information in the source document, some types of information were not accessible by using only standard DSSSL primitives. For instance, the intrinsic duration and size of media items can only be automatically determined by parsing the media item itself, but access to media parsers requires non-standard extensions to DSSSL. Another limitation was that to specify more fine-grained synchronization relations than possible with event-based timing, MHEG

[3] Note that while the link and alignment structures provided by HyTime are useful, there are many other abstract structures that can be used to model hypermedia documents, e.g. structures based on the document's underlying rhetoric or narrative.

presentations need to resort to other techniques (e.g. multiplexing of media streams) that cannot be controlled from a standard transformation language.

While MHEG's timer events are sufficient for coarse-grained synchronization and can be generated during the transformation, it is hard to specify the event wiring necessary for an even moderatly complex presentation in a transformation sheet. One approach to address this problem involves style sheets that transform to higher level, abstract presentation concepts. The style sheet would than be relieved from the transformation of these concepts to lower level concepts such as MHEG's event model. This transformation could than be carried out by a reusable back-end application.

9.3.4 Discussion

The implemented conversions to SMIL and MHEG mainly explored different uses of spatial layout, scheduling, linking and style issues. Generation of more lower level presentation information, however, has not been explored. Future research is needed to determine to which extent such information can be generated automatically. This could include more automatic support for alternative media formats, generating presentation hints regarding buffering strategies, and automatic adaption to screen size, network bandwidth, etc.

Another limitation was that our transformations could only be applied to a small set of very similar documents. For example, the presentation of a document with a small number of media items (i.e. all items fit on the screen at once) will require a different spatio-temporal layout than a document with a larger number of media items (e.g. multiple interconnected scenes). This lack of flexibility is due to a "template-oriented" description of the presentations in the DSSSL style sheet, and the direct mapping of the source document structures to these templates. More recent research indicates that finding an adequate presentation for a given document often requires a trial and error approach, where the transformation needs to be able to backtrack over alternative presentation possibilities [192]. At the time of writing, this type of specification is not supported by the functional transformation languages.

Another drawback of our approach, which applies both to the conversion to SMIL and to the conversion to MHEG-5, is that even after separating out the HyTime and Berlage related functionality, the transformation sheets still combine two design tasks into a single style sheet. Ideally, a hypermedia transformation separates the first task, determining the way the document should be presented, from the second task, determining how to realize such a presentation in SMIL, MHEG-5 or another document format. This separation, however, would require a common abstract hypermedia output model similar to the output model formatting objects define for text-based applications (the pros and cons of this approach are discussed in Section 2.2.2 on formatting models, see especially the discussion on page 25).

9.4 Conclusion

During the development of the Berlage environment, we explored the boundaries between generic hypermedia structures and high and low level hypermedia presentation information. In particular, we explored the differences between the presentation-oriented notions of space, time and links in CMIF and SMIL versus the more abstract interpretation of the same concepts by HyTime. In addition, we explored transformations to the lower level and final-form structures of MHEG-5. These transformations, and their current limitations, provided insights into the requirements for a time-based hypermedia version of the multiple delivery publishing model. These requirements include:

- **Appropriate source document format** — At the time of writing, most (time-based) hypermedia document formats (including SMIL) and document models (including the AHM) are mainly presentation-oriented. The main drawback is that they have very limited support for modeling (application specific) semantics from which presentations can be generated more automatically. In this respect, HyTime was clearly ahead of its time — it is still the only standard that is developed to support more abstract and presentation-independent hypermedia documents. Given the lack of experience with this type of abstraction, however, it is not clear whether the most commonly needed structures in presentation-independent hypermedia documents are indeed the structures that are defined by HyTime, or whether other type of abstractions would be more useful.

- **Appropriate target presentation format** — Given an appropriate source document format, one also needs an appropriate presentation format that can be used to present the document. Despite the research focus on presentation formats, the availability of standardized presentation formats for time-based hypermedia is still limited. MHEG-5, for example, has never become a widely supported standard, and at the time of writing, support for SMIL and MPEG-4 is still limited, even on the Web.

- **Appropriate transformation tools** — As said before, the DSSSL formatting model does not support time-based media, and implementations of DSSSL's transformation process are not commonly available. This makes DSSSL highly unsuitable for hypermedia processing. The situation has improved considerably after the introduction of XSL. Ironically, for XSL the situation is reversed: the transformation part is widely supported, in contrast to the XSL formatting part. This makes it easy — at least in theory — to realize XSL transformations from XML-encoded documents to SMIL presentations. In practice, however, hypermedia transformations are often hard to describe in a purely functional language.

- **Appropriate hypermedia presentation abstractions** — Hypermedia document transformations combine the task of defining the presentation with the task of realizing that presentation in a particular format. This could be solved by the specification of an abstract, format-independent presentation model. While theoretically attractive, it is not clear to what extent such a model could be successfully used in practice. Similar models for text, defined in terms of formatting objects in both DSSSL and XSL, have at the time of writing not been widely adopted. A major drawback of an abstract hypermedia presentation model is that is should model an extremely wide variety of presentation features, and that these features, especially in the case of on-line presentations, are constantly evolving.

Many organizations already generate the HTML pages of their Web site automatically from content stored in a database. With the increasing support for both proprietary and open formats for the dissemination of time-based hypermedia, more and more organizations want to automate the media intensive part of their web site in a similar way. At the same time, alternative means of access to the Web (e.g. hand-held devices) also gain in popularity. In the context of these developments, the presentation-independent storage of hypermedia content and adequate models and tools for hypermedia transformations will only gain in importance.

Chapter 10

Summary and Discussion

This last chapter first gives a summary of the conclusions reached in the thesis. It then gives a high level overview of the interrelationships of the key document models that have been discussed. Finally, the last section discusses open issues and future work.

10.1 Summary

The thesis consists of three parts. This section provides for each of the three parts a short characterization of the individual chapters, followed by a summary of the main results.

10.1.1 Summary of Part I

The first part of the thesis describes the multiple delivery publishing model for electronic documents, which supports publishing multiple presentations based on the same source document. From the perspective of this model, an overview of the many different aspects of electronic documents is given, along with the underlying models, associated file formats and protocols, and the main research topics in each research area.

Chapter 1 briefly discusses the research fields that have had a large influence on the document models in the thesis. These fields include electronic publishing, hypertext, multimedia and the Web.

The first section of Chapter 2 describes basic document related terminology, along with the multiple delivery publishing model itself. The remaining sections take a closer look at the document models used for text, hypertext, multimedia and hypermedia documents, and discuss how the characteristics of these models relate to the multiple delivery publishing model.

Chapter 3 deals with the document formats and protocols found on the Web, again from the perspective of the multiple delivery publishing model. It starts by providing an overview of the first generation Web protocols, most notably the HTML, HTTP and URL specifications. It then discusses the second generation protocols, focusing on XML and related specifications for style sheets (such as CSS and XSL), hyperlinks (XLink and XPointer) and multimedia (SMIL).

One of the major observations of the first part is that from the perspective of the electronic publishing community, many of the limitations of the initial Web protocols have been resolved. Separation of structure and presentation can now be effectively realized by using XML or a stricter version of HTML, in combination with standardized style sheet languages. Most of the document models and tools that are required for a Web-based implementation of the multiple delivery publishing model are now available or in the last stages of their development.

From the perspective of the hypertext community, the Web realized the ubiquity of hypertext technology. It has, however, still a long way to go before it provides the type of functionality the early hypertext pioneers envisioned. The second generation Web-protocols mainly address general electronic publishing issues, while the Web's hypertext functionality hardly showed any improvement. While XLink addresses one of the long outstanding issues of the hypertext community (the ability to specify non-embedded, multi-ended hyperlinks), it remains to be seen to what extent this functionality becomes commonly available. The same applies to another objective of the hypertext community: blurring the distinction between author and reader. This requires support for collaborative forms of writing and annotating. The functionality realized by HTTP's PUT command and new specifications such as WebDAV provide only a minimal starting point, and even this functionality is hardly used in practice. To a large extent, the Web is a still read-only medium.

From a time-based hypermedia perspective, the good news is that with bandwidth increasing and the adoption of new protocols and document formats such as RTSP and SMIL, the possibilities for disseminating multimedia information over the Web are rapidly improving. On the other hand, providing good hypermedia content remains difficult and expensive, especially when compared to the more text-based content currently available on the Web. In addition, the available models and tools used to create and disseminate time-based hypermedia are still hard to integrate with those commonly used for text-based content on the Web. For a hypermedia version of the multiple delivery model, several issues still need to be solved. The distinction between document characteristics that are of a purely presentational nature versus those that are of a purely structural or semantic nature, for example, proves to be difficult to make in many time-based hypermedia documents. Instead, it appears to be more fruitful to distinguish between those characteristics that may change across different presentations, and those that remain constant because they are considered to be an inherent aspect of the document. In addition, multimedia models and tools have traditionally focused on the presentation-oriented aspects of time-based hypermedia documents. We lack commonly accepted models and tools that support a more generic specification of hypermedia documents and the transformation of these documents to a final-form, synchronized and interactive hypermedia presentation.

Main results

The main contribution of the first part is the systematic overview of the relationships between the research on structured text, hypertext, multimedia, hypermedia and Web-based document processing. It analyzes the fundamental aspects of the different approaches and the resulting incompatibilities. Out of this analysis, a number of open issues that need to be addressed for a seamless integration of the various approaches can be distilled:

- Reconciliation of the various approaches based on the text-flow/page paradigm with the approaches based on time-flow/scene paradigm. Current structured document models and tools assume the document consists of (one or more) text flows that need to be presented on a sequence of pages or scrollable view port. For most time-based hypermedia documents, this assumption does not hold. In addition, most structured document models and tools do not support multimedia requirements regarding layout, synchronization and timing, alternative content and quality of service management. Related to this issue is the fact that many models assume a hierarchical document structure that reflects either the text-flow or temporal progression, which makes integration of the two models hard in practice.

- Reconciliation of the approaches based on external markup with those based on embedded markup. Most SGML and XML tools, especially those in use on the Web, are based on embedded markup. Many of the advantages of open hypermedia systems (OHS), however, are based on the fact that document structures are encoded, managed and processed independently from the document content. OHS models for linking (which support bidirectional and multi-ended links, semi-private annotations etc.), OHS document formats (which encode links externally) and OHS system architectures (which use dedicated link servers) are all based on this principle.

- Move away from the current "presentation versus structure" dichotomy towards a "stable versus variable" document characteristics dichotomy. The strict distinction between presentation and structure is hard to maintain in time-based hypermedia. Spatio-temporal relations can impact the document's semantics, while these semantics may be hard to make explicit in the document structure. But even in this case, it is important to discriminate between such spatio-temporal relations and other, more variable, presentation-oriented aspects.

- Support for this separation between stable and variable document characteristics, not only in text-centric applications, but also in document models and tools for time-based hypermedia. Currently, many hypertext, multimedia and hypermedia models do not discriminate between the two, nor do they discriminate between the document and its presentation. For example, few hypermedia models discriminate between document-oriented, structured links versus presentation-oriented navigational links. As a result, many different hypermedia presentation models exists, but there are very few usable, higher-level hypermedia document models.

- Integration of the more advanced hypertext features found in many of the more traditional hypertext systems into the common Web infrastructure. Traditional hypertext features that are hardly supported in the systems developed by the electronic publishing, multimedia and Web communities include composition, versioning, CSCW (Halasz's issues), bidirectional and multi-ended links, persistence, security and authenticity (Engelbart's requirements). Note that several of these features closely relate to the support for external markup.

10.1.2 Summary of Part II

The second part provides a more formal treatment of the most important document models discussed in Part I, the Dexter Hypertext Reference Model and the Amsterdam Hypermedia Model (AHM).

Recognizing the object-oriented nature of some of the main components of the Dexter model, Chapter 4 builds on the object-oriented modeling features of the Object-Z specification language to give a formal specification of the Dexter Model. The Object-Z specification of the Dexter model also provides the basis for an incremental specification of the AHM given in Chapter 5. The AHM is formalized by refining and extending the concepts defined by the Dexter specification.

Two main topics of Part I have not been addressed by both the Dexter and the AHM formalizations: document transformation processes and the temporal behavior of hypermedia documents at runtime. The first section of Chapter 6 uses a formal model of a stateless transformation to illustrate the most important aspects of current stylesheet-driven document transformations. The second section investigates methods that can be used to formalize the specification of the real-time behavior of hypermedia systems.

Main results

The Object-Z formalizations in Part II are used to illustrate important aspects of the models involved, especially when these aspects do not fit in the multiple delivery processing model. The use of Object-Z for the specification of the Dexter model resulted in a more intuitive and concise version of the original formalization in Z.

The use of Object-Z in the formalization of the Amsterdam Hypermedia Model allowed an more incremental specification. This facilitates comparison of the two models and makes the differences between the AHM and Dexter more explicit. In general, the formalization process helped us make explicit those characteristics of the AHM that could not be derived from existing (informal) descriptions or would otherwise have remained implicit and hidden in prototype implementation code.

The specifications also make clear that neither Dexter nor the AHM discriminate between the structure of the source document and the structure of the presentation. This theoretical limitation can be the source of practical problems. Hyperlinks, for example, are typically defined in terms of the document structure. Linking and browsing semantics of the associated presentation become unclear when document and presentation structures differ. The consequences of this problem are not limited to these two models — linking on the Web, for instance, often suffers from the same ambiguity.

Finally, suitable formal specification methods for specifying the real-time behavior of interactive and time-based multimedia are still under development. The more mature, often used and off-the-shelf specification methods do not support the quantitative and interactive timing that is required for the specification of time-based hypermedia presentations.

10.1.3 Summary of Part III

The third and final part of the thesis discusses software architectures that have been designed to implement the document models discussed in Parts I and II.

Chapter 7 discusses the general notion of a software architecture and gives a high-level description and evaluation of the typical architectures used to process hypermedia documents. It describes open hypermedia architectures that are are characterized by server-side processing of links and other external document structures, the architecture of the Web that is characterized by relatively large client and thin server implementations, and SGML systems, that are traditionally characterized by a relatively large amount of application-specific processing and only a few standardized and reusable components.

Chapter 8 discusses the DejaVu framework, that provides both an architecture and a set of reusable components for exactly that part of an SGML system that is traditionally regarded as application-specific. Unlike most other SGML applications, DejaVu focuses on multimedia and hypermedia documents instead of text-oriented documents. Rather than a single application or toolkit, DejaVu provides an object-oriented framework for developing applications that process hypermedia documents on the Web. The framework is not tied to a built-in document or presentation model, and this makes it very flexible. Without such a model, however, it proved to be difficult to provide high level support for common hypermedia functionality such as hyperlink navigation and synchronized presentation.

Finally, the Berlage environment discussed in Chapter 9 addresses this problem by focusing on a hypermedia document model with built-in support for both synchronization and hyperlinking: the Amsterdam Hypermedia Model. The architecture implements document transformations that have been used to explore the differences in presentation independence among time-based hypermedia document models. It explores the up-conversion from (presentation-specific) CMIF to (presentation-independent) HyTime, and the down-conversions from HyTime to SMIL and MHEG-5.

Main results

To a large extent, the explorative research carried out during the development of the DejaVu framework and the Berlage environment presented in Part III also forms the basis of the material presented in Part I and Part II. Note that the first parts of the thesis showed the differences between document models, the need for an integration of these models in order to meet the requirements of the different communities, and the difficulties that arise when these models are combined and integrated. In the third part of the thesis, the overview of different hypermedia architectures showed that, while integrating the document models may be hard, integrating hypermedia applications and the different underlying architectures is likely to be even harder.

The DejaVu framework combines structured document technology with a highly extensible multimedia presentation environment. While the framework focuses on documents encoded in SGML or HTML, the main strength of the framework is that it is not tied to the limitations of a built-in document model or presentation model. Instead, it employs a general purpose scripting language to encode style sheets and embedded scripts in SGML documents. Combined, these techniques are sufficiently flexible to exploit the functionality of a range of multimedia components provided by the DejaVu runtime environment. The main drawback of the design is that without an underlying presentation model, scripting alone proved to be a too low-level technique to realize common hypermedia presentation functionality such as synchronization and hyperlinking.

The development of the Berlage environment provided insights on how presentation-oriented space, time and link structures of hypermedia presentation models can be used to convey the presentation-independent space, time and link relations in structured hypermedia documents, and the requirements on the underlying architecture to support both types of structures and the necessary conversions. It showed several limitations of today's text-oriented style and transformation languages when they are applied to multimedia documents. Berlage also gave us hands-on experience in dealing with the differences between higher-level models, such as CMIF and SMIL, and lower-level models, such as MHEG-5. We explored the conversion of the typically structured, implicit and coarse-grained presentation-information in the former models to the often flattened, explicit and fine-grained presentation-information in the latter. When compared with the DejaVu framework, an advantage of the Berlage environment is that it uses the built-in hypermedia features provided by existing document models and their implementations. This avoids much of the low-level programming effort needed in the DejaVu framework. The drawback of the approach is that it is limited by the functionality provided by standardized presentation formats such as SMIL and MHEG, which makes adding new hypermedia functionality much harder than in the case of the DejaVu framework.

The work on document transformations within the Berlage environment also formed the basis for more recent work on automatic generation of hypermedia presentations, a topic that is beyond the scope of this thesis.

One of the more general results of the research reported here is the insight into the differences between electronic document models, and their potential role in the multiple delivery publishing model. The following section provides a high-level overview of the interrelationships between the key document models that have been discussed.

10.2 Document Models from 50,000 feet

While there are many ways to differentiate between the document models discussed, the thesis focused on the type of hypermedia functionality provided by each model, and the extent to which presentation-specific information is modeled. Figure 10.1 on the following page[1] uses these two dimensions to give a high-level perspective of most models that have been discussed in the thesis. Note that languages for markup and meta-markup, structured document models and presentation models are all mixed in one figure, and, for brevity, are all regarded as document models in the following discussion. The purpose of the figure is not to give an accurate classification, but to illustrate the discussion below. See Appendix A for a short characterization of the models listed in the figure.

The vertical axis in the figure represents the level of presentation independence. The models allowing descriptions of fully formatted presentations are positioned near the bottom, and the more abstract models, describing the logical structure of the contents without presentation-specific details are at the top. While documents based on the models at the bottom lack the flexibility to adapt to different presentation environments, documents that are based on the models at the top lack the presentation information that is needed to convey the content effectively to the human user. The multiple delivery publishing process addresses this issue. It uses the models positioned at the top for authoring and long-term-storage of documents. Such documents are then subsequently converted to one or more concrete presentations by using the models positioned at the bottom. The wish to apply the same process to complex hypermedia documents is an important motivation for the research upon which this thesis is based.

The horizontal axis indicates hypermedia functionality; it runs from linear text-based models on the left to fully hyperlinked multimedia on the right. While the models to the far left lack sophisticated hypermedia support, they are indispensable when it comes to high quality structuring, formatting and typesetting of complex text documents. The models to the far right typically lack this type of sophistication when it comes to text formatting, but instead, these models support interactive and dynamic hypermedia presentations. Other important differences between text, hypertext, multimedia and hypermedia document models have been discussed in the remainder of Chapter 2. Note that applications might need features from models from both the left and right hand side of the figure, and many of the open issues discussed in the following section relate directly to this requirement.

The Web-based models discussed in Chapter 3 are also represented in the figure.

[1] An earlier version of this figure appeared in [195].

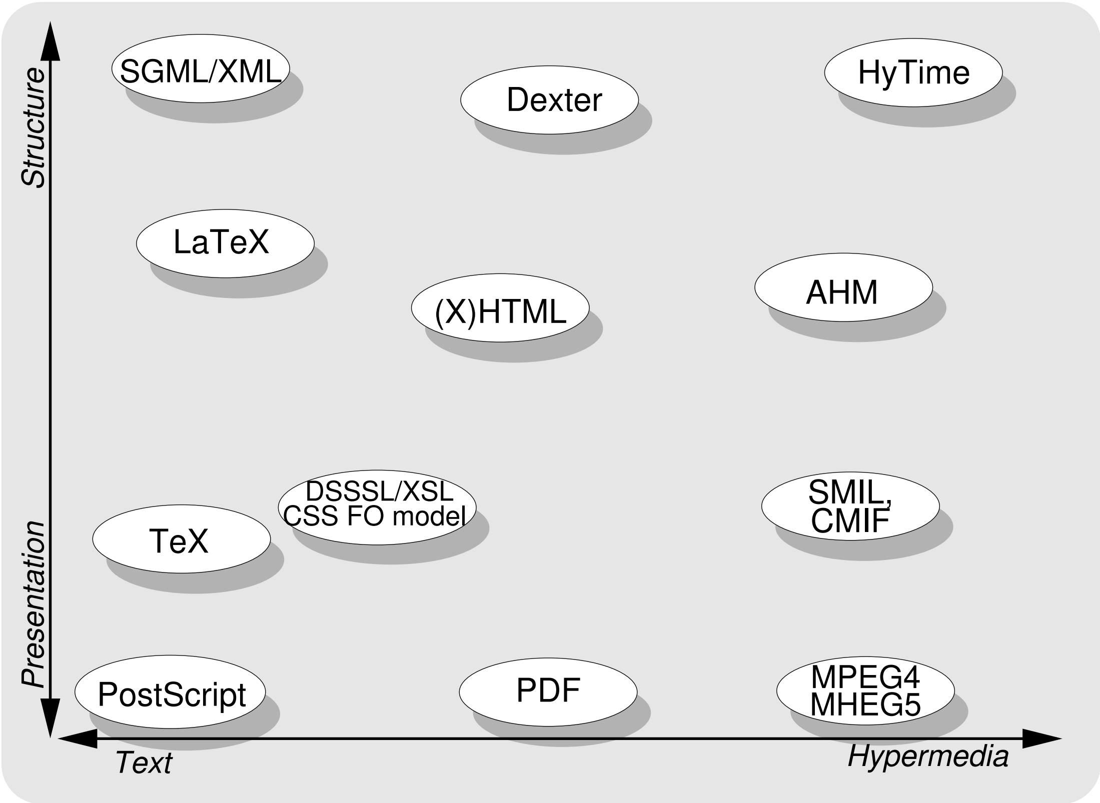

Figure 10.1: Document and presentation models.

Examples include the abstract, hierarchical document model of XML, the "hybrid" document model of HTML, the common W3C formatting model of CSS and XSL, and the multimedia presentation model of SMIL.

Note that the DSSSL formatting model and the W3C formatting model are developed as reference output models for many types of text-oriented documents, including HTML. They do not support, however, synchronization and other presentation-issues that characterize synchronized hypermedia. In this context, it is interesting to observe that on the presentation-independence axis, SMIL is positioned at about the same height as the W3C formatting model. SMIL is usually classified as the Web's baseline document format for synchronized hypermedia documents, playing a role similar to the role HTML plays for hypertext documents. However, its position in the figure suggests that SMIL could also serve as a standardized output model for synchronized hypermedia presentations, thus playing a role similar to the role the W3C formatting model plays for text-oriented XML. This potentially dual role for SMIL is typical for multimedia, where the line between presentation-oriented and structured document modeling is a lot fuzzier than for plain text.

The Dexter hypertext reference model and the Amsterdam Hypermedia Model (AHM) do not — unlike the SGML and XML related models — explicitly discriminate between the document structure and the presentation structure. Nevertheless, we positioned the Dexter model very high in the figure because it only models abstract placeholders for presentation-specific information. The AHM provides more time-based functionality and multimedia-specific layout features, and is consequently more presentation-specific. Therefore, the AHM is positioned to the lower right of the Dexter model.

Note that while the AHM, SMIL and CMIF all contain presentation-specific information, this information is often implicit and relatively high-level. One of the advantages is that such information can still be manually authored, but a disadvantage is that it requires more processing to render the final presentation, and that the author has no control over low-level timing and other presentation information. Standards such as MPEG-4 and MHEG-5 deal with these issues, by specifying a more lower-level presentation format that is readily processable by players with a relatively small footprint or give the content provider more control over issues related to streaming, multiplexing, encoding and decoding, etc.

10.3 Discussion

Based on the high-level perspective given in the previous section, one might conclude that most of the issues related to processing hypermedia documents on the Web have been solved. HTML has been considerably improved, and — maybe even more important — is no longer the only document format on the Web. In fact, there are now standardized document models for most strategic points on the two dimensions depicted by the figure. Ranging from high quality printed text to advanced hypermedia, and from purely abstract structures to final-form delivery formats, there seems to be at least one appropriate document model for any document-oriented application.

In addition, most of the requirements from the electronic publishing community have been met. The ingredients for a multiple delivery publishing model — that used to be only available for SGML — are now also available on the Web: XML to encode structured documents, XSLT to specify the transformation and formatting process, and several standardized output formats (including the upcoming XSL formatting object vocabulary) that can be used as the final-presentation format. The multiple delivery publishing model and the new document formats on the Web can be used to overcome most of the disadvantages associated with the initial versions of HTML.

There are, however, still many open issues. While the multiple delivery publishing model has already been successfully applied in SGML-environments, the typical SGML-based environment differs radically from its XML-based counterpart. A side-effect of the complexity of SGML is that SGML is mainly used within large industrial consortia and large organizations. As a result, most users only need to deal with the limited number of different document formats that is in use by their organization. In contrast, with XML being an almost ubiquitous technology, the number of different XML-based document formats is rapidly growing and already overwhelming. While some of these formats are for local use only, many of them are used for interchange over the Internet. One of the risks related to XML is that it will not develop into the common syntactical basis upon which all Web-based documents are built, but instead gives rise to a new Babel [150] of many, many different documents that may be well-formed XML, but are otherwise all completely incompatible.

Additionally, in SGML-based environments, structured documents are typically used for industrial scale authoring and long-term storage. As such, they are carefully designed by large organizations or (industrial) consortia. The organization that designs a new document format is typically also responsible for designing the associated style sheets for that particular format, for specifying the necessary transformations to other, more standardized interchange formats, and for developing other tools that were used for internal processing. On the Web, XML data is exchanged between various organizations on a much wider scale, not only in a document-oriented context, but also in a more data-oriented context. XML documents, potentially received from multiple organizations using different document models, need to be partly disassembled and recombined.

At the time of writing, SGML and XML technology is not sufficiently flexible to be applicable in the more dynamic XML environments sketched above. While XML Namespaces (and, to a certain extent, SGML's architectural forms) can be regarded as a first step into supporting this new type of processing, they are only addressing the most basic, syntactical aspects of the problem. Many steps of the multiple delivery publishing model (such as document validation and style sheet transformations) are still based on techniques that worked well in a relatively stable SGML environment. In addition, it remains to be seen to what extent the multiple delivery publishing model itself is a suitable model to cope with the new requirements.

From this perspective, the Web still has two important needs:

- The need to integrate presentation models
- The need to integrate document transformation techniques

These requirements are discussed below.

10.3.1 The need to integrate presentation models

One of the consequences of the introduction of XML is the explosion in the number of output formats on the Web. While the Web used to be based on HTML as the single presentation format, many other XML-based presentation formats are now emerging. In many cases, the multiple delivery publishing model can help to generate several Web-based presentations from a single source document. For example, one could choose SMIL for a multimedia version, SVG for documents with 2D vector graphics, X3D for 3D graphics, MathML for mathematical documents, WML (Wireless Markup Language) for small wireless devices, etc.

This process assumes, however, that there is at least one adequate presentation format for each output device. Despite all the different output formats, this is not realistic. Often, one needs to combine features from different presentation formats into a single presentation. For example, to animate vector graphics, one needs features from both SVG and SMIL Animation. In addition, the use of animated vector graphics should not be limited to SVG or SMIL documents and their applications, it should also be possible to support these features in browsers for HTML or (future versions of) WML. As another example, to turn an HTML document into a synchronized slide show, one might want to use the features of SMIL timing to synchronize the presentation of HTML elements. The features of HTML's interactive forms could also be used to deal with form-filling in SMIL documents, etc. These examples require an unprecedented level of integration in terms of document models, software architectures and implementations. While the multiple delivery publishing model can be successfully used to transform documents to different presentation formats, it does not address the problems that arise when different presentation formats need to be combined into a single format.

The need to handle a wide variety of different types of presentation features is the core of the problem. Each particular format can only support a limited subset of the total set of available presentation features, if only because this set is changing over time as presentation technology evolves. As a result, applications require features from multiple existing languages to be combined into a new one. In this perspective, it seems unlikely that format-neutral reference output models (such as the text-oriented formatting models specified by both DSSSL and XSL, or their future time-based hypermedia equivalents) can provide the required variety in presentation features. Instead of defining even more document markup languages and presentation formats, it will be more fruitful to define models and techniques that make the process of defining and implementing new languages (possibly based on the features of existing languages) more efficient and effective.

In a Web environment where many formats are needed, and new formats are continuously created, basic hypermedia features need to be applicable across different formats, rather than being limited to a few special-purpose document formats. Examples of such basic hypermedia feature include the key features we have explored in this thesis: hyperlink navigation, spatial layout and temporal synchronization. To realize the required ubiquity, however, we need a way of describing and implementing these features that goes beyond the individual document format or Web application. For hyperlinking, the long history of XLink proves that it can be surprisingly difficult to develop a commonly accepted link model and syntax. On the other hand, such agreement does exist for an important aspect of linking: the ability to specify arbitrary portions of a document. Specifications such as XSL and XPointer now share a common basis for this functionality in the form of XPath. A similar example can be found for spatial layout: CSS, XSL and SMIL all use the same underlying box model as a basis for their spatial layout. Finally, when it comes to a common timing model on the Web, the recent modularization of SMIL [17] provides a promising starting point, since it allows the use of SMIL's interactive synchronization features in other presentation formats, including XHTML.

10.3.2 The need to integrate document transformation techniques

Above we addressed the need for an extensible set of document presentation features that are applicable across multiple formats, using hyperlinking, layout and timing as the typical examples. Similar arguments apply to support for interactive forms and other types of interactive behavior, such as script-based interactions. The multiple delivery publishing model assumes that all presentation-specific information is specified in (one or more) style or transformation sheets. However, in practice the situation is more complex.

First, many source documents are "hybrid" documents, to the extent that they mix both structured markup and (inline) presentation-specific markup. For multimedia, we advocated that such a hybrid approach is often required, since part of the presentation information can be an invariable part of the document with strong associated semantics. As a minimal requirement, style sheets need to be able to resolve conflicts between the inline and external presentation information. But in general, applications might require more application-specific trade-offs between the information in the document and the information in the style sheet.

Second, style and transformation sheets are not the only techniques in the processing chain that manipulate documents and the way they are presented. Using the DOM, for example, scripts and other application-specific code can manipulate any part of a document before it is presented, and such manipulations can occur both at the client and server side. Additionally, properties of time-based or animated documents also change dynamically. In XHTML+SMIL, the value of the visibility and other style properties of HTML elements are dynamically controlled by SMIL timing attributes embedded in the HTML markup. In SVG, attributes values can be manipulated during the presentation via embedded SMIL animation elements.

For all these different document manipulation techniques, it needs to be clear how they affect one another, what the precedence and conflict resolution rules are, etc. For example, many manipulation techniques depend on the structure of the document and its content. These techniques break down when the order of application of the document manipulations is ill defined. For instance, a selector of a CSS style rule often matches on the value of a particular attribute. It needs to be clear whether the selector should match on the value of the attribute before or after a DOM call, or whether it should be re-evaluated every time the value changes. Or, as another example, a hyperlink using an XPointer often points to a particular element based on the tree structure of the document. It is not clear how to resolve such a pointer when the target document's structure has been changed by an XSLT transformation: the target element might be relocated in the document tree or even be completely removed.

Conclusion

The multiple delivery publishing model as described in this thesis is above all a model that is designed for transforming a structured document into a wide variety of presentations. The diversity in user, application and platform requirements on the Web will increase the need for such a model even more. In most practical cases, the model can already be effectively employed, since many of its individual components have been standardized and incorporated in commonly available Web software. For time-based hypermedia documents, however, important parts of the processing chain are still missing. These missing parts include generally usable hypermedia abstractions on the document and the presentation level, and a transformation technique that is suited for the more complex transformations between those two abstraction levels.

But even if the future would fill in those missing parts, the multiple delivery publishing model does not solve all technical problems related to hypermedia publishing on the Web. As described above, the model does not deal with the issues that arise when features of different presentation models need to be combined. In addition, the model does not take into account manipulation techniques other than those described in style and transformation sheets.

To address these issues, a more modular approach to presentation models on the Web is needed. Building blocks such as XML Namespaces and modularizations of specifications such as XHTML, SMIL and CSS are a first step in this direction. But such a modular approach also requires a clear description of how the different presentation models and document transformation techniques should work together. To be able to answer this question, the Web needs to move away from its weak notion of Web architecture, that is, the Web as a set of specifications and protocols. Instead, it should move towards a stronger notion of architecture by providing a more explicit description of the Web's document processing chain, for example by means of the definition of a common reference software architecture describing the most common document processing steps, the related components and specifications, and their interfaces.

Appendix A

Glossary

Hypermedia Document Formats

AHM: The Amsterdam Hypermedia Model [121] is developed at CWI, Amsterdam as an extension to the Dexter model. It includes the primitives for spatio-temporal composition, synchronization and link context needed to model the portable hypermedia documents generated by CWI's CMIFed authoring environment.

CMIF: The native file format of CWI's CMIFed authoring environment [228]. See AHM for more information.

DSSSL: The Document Style Semantics and Specification Language (ISO/IEC 10179 [136]) is a language for encoding the presentation of text and graphics in a two-dimensional environment. The language is based on Scheme (a dialect of Lisp, see [60]). Scheme code is embedded in an SGML document which conforms to the SGML architecture defined in the DSSSL standard. DSSSL consists of two parts, a tree transformation language that can be used to re-order structured documents prior to presentation, and a formatting process that associates formatting instructions with specific nodes in the target representation, the *flow object tree*. DSSSL specifications are device independent pieces of information that can be interchanged between different platforms. The back-end formatters needed to generate the final form of document (e.g. a PDF or RTF file, or a presentation in a computer display) are not standardized by DSSSL.

Dexter: The Dexter Hypertext Reference Model [114] is the result of various meetings — the first of which was held in the Dexter Inn — of the designers of the majority of the well-known hypertext systems in 1988. The model defines three layers, one to abstract from the actual media type, one to describe the hypertext data layer and one for describing the user interface related processes. The Dexter model provided a common terminology, especially on the differences between links, anchors and link markers. In dealing with access by means of n-ary and bidirectional links, queries and virtual structures, Dexter solves part of the problems addressed in Halasz' famous "Seven Issues" article. However, being a closed and single-user model, it does not solve the problems related to versioning and security in a multi-user environment, which makes it unsuitable for modeling open systems such as the World Wide Web. Additionally, it does not address temporal and spatial relations between components, which are important issues in a multimedia environment.

HTML: The HyperText Markup Language (W3C Recommendation 24-Apr-1998 covers version 4.0 [186]) is a simple markup language used to create hypertext documents that are portable from one platform to another. It aims to capture recommended practice and standardize the many extensions to earlier versions. HTML 4.0 in particular provides better support for integrating script and style sheet languages and accessibility and internationalization issues. HTML is an application of SGML, future HTML work of W3C is expected to focus on XHTML [232], an XML compliant version of HTML.

HyTime: The Hypermedia/Time-based Structuring Language (ISO/IEC 10744 [134]) is an application of SGML that provides facilities for describing the relationships between different types of data. It provides standardized methods for describing hypertext links, scheduling and alignment, event synchronization and projection in multimedia and hypermedia documents. HyTime extends SGML by defining a specific SGML *architecture*, and it defines how other SGML extensions (such as DSSSL) can be described by defining new architectures. HyTime does not provide a standardized way of coding hypermedia presentations, but it provides a language that can be used to describe how any set of hypermedia objects has been interconnected and how it could be accessed. HyTime's roots are in an effort to develop a standardized music description language (SMDL is now an application of HyTime), which accounts for HyTime's strong scheduling and projection facilities (including the use of virtual time).

LaTeX: LaTeX [153] is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents, but it can be used for almost any form of publishing. LaTeX is based on Donald E. Knuth's TeX typesetting language. LaTeX was first developed in 1985 by Leslie Lamport.

MHEG: The Coding of Multimedia and Hypermedia Information (ISO/IEC 13522, parts 1–5 [137]) is a standard defined by the Multimedia and Hypermedia information coding Expert Group. It provides a standardized set of objects that can be used to control the presentation of multimedia and hypermedia information. The information can be encoded using a binary encoding of ISO's Abstract Syntax Notation One (ASN.1) or a textual representation.

ODA: The Office Document Architecture and Interchange Format (ISO/IEC 8613, parts 1–14 [131]) defines an architecture that describes business documents (e.g. letters, reports, memos) in terms of their content and two hierarchical structures: a logical structure and a layout structure. Documents can be interchanged by using one or both structures, in an ASN.1 or SGML encoding. Note: the acronym "ODA" is also expanded as the "Open Document Architecture".

PDF: The Portable Document Format [8] is a proprietary standard developed by Adobe Systems Inc. that allows pre-formatted pages to be interchanged over a network. It is based on Level 2 PostScript and supports hyperlinking and annotations, thumbnail icons of pages, and faster processing by supporting a tree structure instead of PostScript's linear structure.

PostScript: PostScript [9] is a stack-based interpreted language — developed by Adobe — with powerful built-in graphics primitives for controlling raster output devices and page description. The features provided by PostScript Level 2 are standardized by ISO's Standard Page Description Language (SPDL, ISO/IEC 10180:1995 [135]).

SGML: The Standard Generalized Markup Language (ISO 8879:1986 [130]) defines a set of semantics for describing document structures, and an abstract syntax for formally coding document type definitions (DTDs). SGML does not suggest any particular way in which documents should be structured but allows users to define the structure they require in a DTD. These structures usually model the logical structure of the document, but several applications of SGML (including HTML) also describe presentation-oriented information.

SMIL: The Synchronized Multimedia Integration Language (a W3C Recommendation since June 1998 [231]) is a document format with an XML-based syntax that allows integrated presentation of hyperlinked multimedia objects over the Web.

SPDL: See PostScript.

TeX: TeX [149] is a computer language designed by Donald E. Knuth for use in typesetting. It is used in particular for typesetting math and other technical material. See also LaTeX.

XHTML: See HTML.

XML: The eXtensible Markup Language (a W3C Recommendation since February 1998 [38]) is a subset of SGML designed in such a way that Web browsers do not need to access the DTD to validate the document before display, while still being interoperable with both SGML and HTML.

Hypermedia Terminology

Architectural form: A syntax construct defined by HyTime to describe properties that several SGML elements have in common. To a certain extent, this is comparable with an abstract base class in object-oriented programming languages. HyTime defines architectural forms for a large number of abstract hypermedia concepts. In addition, the mechanism is used by DSSSL and could also be applied to other domains. A set of architectural forms can syntactically be defined by an SGML DTD, which is often called a meta-DTD, to stress the difference in the abstraction level when compared with an application's DTD that defines a concrete document type. At first sight, architectural forms may seem comparable to XML Namespaces because they allow applications to mix syntax defined in multiple DTDs. There are, however, many differences. In the case of namespaces, the DTDs are on the same level of abstraction, where in the case of architectural forms, the meta-DTDs constrain the syntax defined by the application's DTD. Additionally, architectural forms allow documents to be validated, both against their application DTD and their meta-DTDs. Automatic validation against the DTDs referred to by a document that uses XML namespaces is not possible.

External link: External, or out-of-line, links are stored separately from the documents that contain the endpoints of the link. One of the advantages of external links is that they allow linking without modifying the documents being linked. Cf. internal links.

Grove: The data structure that is the result of parsing an SGML or XML document. Technically, a grove is a set of trees (hence the name), where each tree represents a different relationship between its nodes. The grove is the basic data structure upon which DSSSL and HyTime are defined, and are also used to define the API of SGML parsers. Its role is comparable with the role of the Document Object Model (DOM) in XML.

Internal link: Internal, or inline, links, are encoded at the source anchor of the link within the main text stream. While often simpler to process and maintain, the drawback of internal links when compared to external links is that modification of the link requires modification of the document.

Appendix B

Introduction to SGML and XML

The following introduction to the basic concepts of SGML and XML is adapted from [226]. It is mainly provided to help in understanding the examples in the thesis that use SGML or XML syntax.

SGML (Standard Generalized Markup Language) [130] and XML (eXtensible Markup Language) [38] are the dominant standards for the encoding of structured documents. Both are tag-based markup languages and the set of tags needed to describe a specific document structure typically depends on the type of documents involved. For example, the tags needed to markup a mathematical article are in general not suitable for describing the structure of a telephone book. As a consequence, the standards do not describe a fixed set of tags, but a way to define an appropriate set of tags and the order these tags should appear in a document instance. Such a definition is called a *document type definition* or DTD. In SGML, every document has an associated DTD, in XML the DTD is optional.

An SGML or XML document essentially consists of two parts: a prolog, containing the document type declaration, followed by the document instance, containing the data interspersed with markup.

Document instance

A document instance is a hierarchical structure of (possibly empty) *elements*, where each non-empty element contains other elements or character data. Each element has a name (the *general identifier*) and the start and end of an element are indicated by tags (typically <name> content </name>). In SGML, begin or end tags may be mandatory or optional, in XML both tags are always mandatory. Moreover, elements contain zero or more *attributes*. Consider the example SGML document below. Note that many optional end tags have been omitted, as this is common in many SGML documents.

```
<memo security=confidential>
 <to>Anne
 <cc>Anton<cc>Bastiaan
 <from>Jacco
 <subject>EP submission
 <body>
   Dear Anne,
   I'm sorry we couldn't make it before the deadline,
   but we will send you a PostScript copy of our
   EP submission before next Wednesday.
 </body>
</memo>
```

The document's root element `memo` has one attribute, indicating the security level of the document. The memo element contains six other elements: a `to` and two `cc` elements stating the addressees, a `from` element specifying the sender, a `subject` and a `body` field. In this example, all elements of memo contain character data and no other elements, but in general elements can be nested to arbitrary depth. Note that the tags emphasize the logical structure of the document rather than stating how it should be formatted. Also note that, instead of encoding it by an attribute, we could have encoded the security level as part of the content by introducing a new element as in:

```
<security>confidential<security>
```

While the difference is generally a matter of taste, the general rule of thumb is to enter all information that is likely to be part of the final presentation as content, e.g. as character data inside an element. Attribute values are generally used for "metadata", that is, information that will not appear in the presentation.

Document type declaration

The document instance above must be preceded by a `DOCTYPE` declaration. The main part of the document type declaration is the *document type definition* or DTD. It defines the elements of a document and the required order of their sub-elements. The elements and their contents are defined by the use of `ELEMENT` declarations.

```
<!DOCTYPE memo [
<!ELEMENT memo O O (to+,cc+,from,subject?,body) >
<!ELEMENT (to|cc|from|subject|body) - O (#PCDATA) >
<!ATTLIST memo security (low|confidential|topsecret) low >
]>
```

The second line declares the memo element, and defines its content as a sequence of one or more `to` elements, one or more `cc` elements, a `from`, an optional `subject` and a `body` element. The two 'O' characters stand for "omit", indicating that the begin and end tag of `memo` may be omitted. The third line defines the elements containing character data only. Their start tags are mandatory, indicated by the '-'. Note that XML does not allow omission of any tags, if this would have been an XML DTD, there would have been no 'O' or '-' indications at all.

The list of attributes of each element is declared by an `ATTLIST` declaration. Attributes can be of different types, and be mandatory or optional. The DTD can specify a default value, as is shown in the case of the security attribute. The DTD declaration may be contained within the `DOCTYPE` declaration, but is typically defined in a separate file. In that case, the `DOCTYPE` declaration contains a reference to that file.

Processing instructions

Processing instructions are used to pass system dependent information to the application to tell how the document is to be processed. Processing instructions are typically contained within '<?' and '>' characters and can appear on arbitrary places within the document. Since their effect is system dependent, documents should make minimal use of processing instructions to benefit from the advantages of platform independence.

Entities

Fragments of markup and character data can be given a name using an `ENTITY` declaration. The declaration of an entity is part of the document type declaration, but the entity may be used within the document instance. The contents of an entity may be defined by a string, or may be contained in an external file, in which case the entity declaration contains a reference to the file. External entities may be referenced by a *system identifier*. Support for these identifiers is system dependent and may include filenames, URLs and database queries. Consider a variant of the previous example:

```
<!DOCTYPE memo
          SYSTEM "http://www.example.org/memo.dtd" [
<!ENTITY ps "PostScript"> ]>
<?stylesheet
  lang=Tcl
  src="http://www.example.org/memo.style">
<memo>
   ...
   but we will send you a &ps; copy
   of our submission
   ...
  </body>
</memo>
```

The first line references an external DTD by means of a system identifier, and the second line defines an entity for later usage. Note that the entity definition is enclosed within the square brackets of the `DOCTYPE` declaration. The third line contains a processing instruction to define the location of the style sheet. In contrast to the URL of the DTD, which is resolved by the parser, the URL of the style sheet is passed to the application without further processing by the parser.

To avoid system dependent identifiers such as filenames, an extra indirection is provided by the concept of *public identifiers*. These identifiers are assumed to be publicly known, and the parser of the target application is expected to be able to resolve them. Typically, a local catalog file is used to map public identifiers onto system dependent ones. *Formal* public identifiers have a standardized and meaningful inner structure, to facilitate automatic resolving without the use of catalogs. In XML, one is required to provide at least a system identifier for each external entity.

Appendix C

Introduction to Z and Object-Z

The Z notation can be used to formally specify the properties of an information system (see [214] for a full description and a more extensive tutorial). Z specifications mix formal mathematical notation with informal prose to explain the formal parts. The formal parts of a Z specification are based on:

- *common mathematical data types*, including but not limited to sets, subsets, power sets ($\varnothing, \subseteq, \mathbb{P}$, etc); booleans ($\mathbb{B}, \mathrm{true}, \mathrm{false}$); numbers ($\mathbb{N}, \mathbb{Z}, \mathbb{R}$, etc); several function types such as partial ($\nrightarrow$) and total functions ($\rightarrow$), injections ($\rightarrowtail$), surjections ($\twoheadrightarrow$), bijections ($\rightarrowtail\!\!\!\!\twoheadrightarrow$), etc; function domain and range ($\mathrm{dom}, \mathrm{ran}$); sequences and bags.

- *predicate logic*, including variable declarations ($x, y : \mathbb{Z}$); predicates that can be true or false ($x \leq y$); the usual connectives of the propositional calculus ($\vee, \wedge, \Rightarrow, \neg, \Leftrightarrow$); universal and existential quantifiers ($\forall, \exists, \exists_1, \nexists$), etc.

- a decomposition construct called a *schema*.

Schemas typically come in three flavors, *state schemas*, *initialization schemas* and *operation schemas*. All schemas can be divided into two parts. State schemas are used to describe the static aspects of a system. The first part of the schema defines the states the system can occupy. The second part describes the invariants that apply to every state. The following example is based on [82], and defines a stack of at most 100 integers. The stack's state schema could be defined as:

$$
\begin{array}{|l}
\hline
\textit{Stack} \\
\hline
\textit{items} : \mathrm{seq}\,\mathbb{Z} \\
\hline
\#\,\textit{items} \leq 100 \\
\hline
\end{array}
$$

The part above the dividing line defines the state variable *items* (a sequence of integers). The lower part specifies that in each state the number of items should always be equal to or less than 100. For each state schema, one must explicitly define its initial state:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad InitStack \quad}} \\
Stack \\
\hline
items = \langle\, \rangle \\
\hline
\end{array}$$

The upper part includes the *Stack* state schema, so the *items* variable is now inside the current scope. The lower part states that the initial schema holds only if the predicate $items = \langle\, \rangle$ evaluates to true. Note that the lower part is a logical predicate, not an assignment operation. The specification of the stack can be completed by defining the operations (state transitions) that are allowed:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Pop \quad}} \\
\Delta Stack \\
item! : \mathbb{Z} \\
\hline
items \neq \langle\, \rangle \\
items = \langle item! \rangle \frown items' \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Push \quad}} \\
\Delta Stack \\
item? : \mathbb{Z} \\
\hline
items' = \langle item? \rangle \frown items \\
\hline
\end{array}$$

The upper half of the schemas contain the declarations of the variables used in the lower half. The convention of variable *decoration* is used to indicate the role the variable plays. Variable names decorated with an exclamation mark ($item!$) denote the output of an operation, a question mark is used for the input of an operation ($item?$) and a primed variable ($items'$) denotes the value of a state variable after an operation has been carried out. The delta ($\Delta Stack$) in the declaration indicates that the operation modifies the stack, and introduces both the primed and undecorated state variables (in the example: $items'$ and $items$). The lower half contains (a conjunction of) predicates that need to hold for the success of the operation. Predicates without primed variables (such as $items \neq \langle\, \rangle$) are pre-conditions, in this case only the pop operation has a pre-condition. It constrains the stack to be non empty (but does not specify what happens if the pre-condition is false). Predicates with primed variables are post-conditions. For the pop operation, the post-condition states that the sequence *items* before equals the concatenation ($\frown$) of two sequences, one containing the item being popped ($\langle item! \rangle$), the other containing the items after the operation ($items'$). The post-condition of the push operation is similar, it constrains the sequence items after the operation to equal the concatenation of the sequence containing the item being pushed and the sequence containing the items before the operation. Note again that the lower halves of the schemas contain predicates, not assignments. Z schemas generally specify *what* happens, and not *how* this should be implemented.

Object-Z Object-Z is an extension to Z, supporting formal specification in an object-oriented style (see [80, 81, 82]). Object-Z extends Z with the notion of a *class schema*. It is used to group state, initialization and operation schemas into one schema. A class can be used as a type. It behaves as a template for objects: each object of a class has a state conforming to the class's state schema and is subject to state transitions that conform to the class's operations. In Object-Z, the stack, now of a generic type, could be defined as:

Stack[*T*]

$$items : \mathrm{seq}\, T$$

$$\#items \leq 100$$

INIT

$$items = \langle\,\rangle$$

Push

$$\Delta(items)$$
$$item? : T$$

$$items' = \langle item? \rangle \frown items$$

Pop

$$\Delta(items)$$
$$item! : T$$

$$items \neq \langle\,\rangle$$
$$items = \langle item! \rangle \frown items'$$

When compared with the plain Z schemas above, we see that in Object-Z, the initialization schema and the operation schemas no longer need to include the state schema. Instead, the Δ is now used by the operations to indicate which state variables will change when the operation is carried out. In this case, there is only one state variable. But in general, every variable x that is not in the delta list of an operation will remain constant; and the post-condition predicate $x' = x$ is assumed to be implicitly included in the lower half of the schema.

Object-Z has many other features which have not been discussed in the short introduction above, see for example [82] for a more extensive tutorial. Features worth mentioning include inheritance and polymorphism, secondary variables and *history invariants*. History invariants are optional temporal logic predicates over the histories of objects of a particular class.

About the specifications in Part II Most of the schemas included in Part II have been validated by the Object-Z type checker [143]. The only exceptions are the schema of the Dexter composite component on page 118, the Dexter hypertext on page 120 and the session class below. The first schema could not be validated because the type checker did not support the define-after-use style declaration needed for the recursive composite class. The hypertext schema could not be validated because Object-Z does not allow the direct notation we used to include class operations as predicates in other operations in the same class. In particular the *linksToComponent* operation is frequently used in this way. We have used this notation because it is easier to read and very common in plain Z. It is, however, not allowed in Object-Z. In Object-Z, integration of operation schemas is only available via operation calculus. The direct "boxed" notation cannot be used because of potential conflicts with the delta lists during schema expansion[1]. The session class revealed similar problems. Also for reasons of readability, we left out several operations from the Dexter session class on page 126. Below follows the complete schema:

Session

$$
\begin{array}{l}
hypertext : Hypertext \\
history : \mathrm{seq}\, Operation \\
instants : Iid \rightarrowtail\!\!\!\!\!\!\!\;+\; (Instantiation \times Uid) \\
instantiator : Uid \times PresentSpec \rightarrow\!\!\!\!\!+\; Instantiation \\
realizer : Instantiation \rightarrow Component \\
runTimeResolver : ComponentSpec \rightarrow\!\!\!\!\!+\; Uid \\
\hline
head(history) = OPEN \\
hypertext.resolver \subseteq runTimeResolver \\
\forall\, uid : Uid;\ ps : PresentSpec \mid \\
\quad uid \in \mathrm{dom}\, hypertext.accessor \bullet \\
\quad\quad realizer(instantiator(uid, ps)) = hypertext.accessor(uid)
\end{array}
$$

INIT

$$
\begin{array}{l}
hypertext.\mathrm{INIT} \wedge history = \langle OPEN \rangle \wedge instants = \varnothing \\
instantiator = \varnothing \wedge realizer = \varnothing \wedge runTimeResolver = \varnothing
\end{array}
$$

openComponents

$$
\begin{array}{l}
\Delta(history, instants) \\
specs? : \mathbb{F}(Specifier \times PresentSpec) \\
\hline
history' = history \frown \langle PRESENT \rangle \\
\exists\, iids : \mathbb{F}\, Iid;\ newInstants : Iid \rightarrowtail\!\!\!\!\!\!\!\;+\; (Instantiation \times Uid) \bullet \\
\quad iids = \mathrm{dom}\, newInstants \wedge \\
\quad iids \cap \mathrm{dom}\, instants = \varnothing \wedge \\
\quad \#iids = \#specs? \wedge \\
\quad instants' = instants \oplus newInstants \wedge \\
\quad (\forall\, s : specs? \bullet \exists\, iid : iids;\ uid : Uid; \\
\quad\quad cs : ComponentSpec;\ ps : PresentSpec; \\
\quad\quad inst : Instantiation \mid \\
\quad\quad\quad cs = (first(s)).componentSpec \wedge \\
\quad\quad\quad ps = second(s) \wedge \\
\quad\quad\quad uid = runTimeResolver(cs) \wedge \\
\quad\quad\quad inst = instantiator(uid, ps) \bullet \\
\quad\quad\quad\quad newInstants(iid) = (inst, uid))
\end{array}
$$

[1]Thanks to Wendy Johnston for pointing this out.

$$
\begin{array}{l}
\textbf{followLink} \\
\hline
\Delta(history, instants) \\
iid? : Iid \\
linkMarker? : LinkMarker \\
\hline
\exists\, aid? : AnchorId;\ uid? : Uid;\ links! : \mathbb{F}\, Link; \\
specs? : \mathbb{F}(Specifier \times PresentSpec) \mid \\
\quad aid? = (first(instants(iid?))).linkAnchor(linkMarker?) \land \\
\quad uid? = second(instants(iid?)) \land LinksToAnchor \land \\
\quad first(\!|\ specs?\ |\!) = \{s : Specifier \mid \\
\qquad \exists\, link : links! \bullet s \in \mathrm{ran}(link.specifiers)\} \land \\
\quad (\forall\, s : specs? \bullet \\
\qquad (first(s)).direction \in \{TO, BIDIRECT\} \land \\
\qquad second(s) = (first(s)).presentSpec) \bullet openComponents
\end{array}
$$

$$
\begin{array}{l}
\textbf{editInstantiation} \\
\hline
\Delta(history, instants) \\
iid? : Iid \\
inst? : Instantiation \\
\hline
iid? \in \mathrm{dom}\, instants \\
history' = history \frown \langle EDIT \rangle \\
instants' = instants \oplus \{iid? \mapsto (inst?, second(instants(iid?)))\}
\end{array}
$$

$$
\begin{array}{l}
\textbf{newComponent} \\
\hline
\Delta(history, instants) \\
component? : Component \\
ps? : PresentSpec \\
\hline
history' = history \frown \langle CREATE \rangle \\
\exists\, uid : Uid;\ inst : Instantiation;\ iid : Iid \mid \\
\quad uid = hypertext.accessor^{\sim}(component?) \land \\
\quad inst = instantiator(uid, ps?) \land \\
\quad iid \notin \mathrm{dom}\, instants \bullet \\
\quad instants' = instants \oplus \{iid \mapsto (inst, uid)\}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{deleteComponent} \\
\Delta(\mathit{history}, \mathit{instants}) \\
\mathit{iid}? : \mathit{Iid} \\
\mathit{uid}! : \mathit{Uid} \\
\hline
\mathit{iid}? \in \operatorname{dom} \mathit{instants} \\
\mathit{history}' = \mathit{history} \frown \langle \mathit{DELETE} \rangle \\
\exists\, \mathit{uid}! : \mathit{Uid} \mid \mathit{uid}! = \mathit{second}(\mathit{instants}(\mathit{iid}?)) \bullet \\
\quad \mathit{instants}' = \{\mathit{iid}?\} \lhd \mathit{instants} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{realizeEdits} \\
\Delta(\mathit{history}) \\
\mathit{iid}? : \mathit{Iid} \\
\mathit{component}! : \mathit{Component} \\
\mathit{uid}! : \mathit{Uid} \\
\hline
\mathit{history}' = \mathit{history} \frown \langle \mathit{SAVE} \rangle \\
\exists\, \mathit{inst} : \mathit{Instantiation} \bullet \\
\quad (\mathit{inst}, \mathit{uid}!) = \mathit{instants}(\mathit{iid}?) \land \\
\quad \mathit{component}! = \mathit{realizer}(\mathit{inst}) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{unPresent} \\
\Delta(\mathit{history}, \mathit{instants}) \\
\mathit{iid}? : \mathit{Iid} \\
\hline
\mathit{history}' = \mathit{history} \frown \langle \mathit{UNPRESENT} \rangle \\
\mathit{instants}' = \{\mathit{iid}?\} \lhd \mathit{instants} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{closeSession} \\
\Delta(\mathit{history}, \mathit{instants}) \\
\hline
\mathit{history}' = \mathit{history} \frown \langle \mathit{CLOSE} \rangle \\
\mathit{instants}' = \varnothing \\
\hline
\end{array}
$$

$$\Diamond\, \mathit{last}(\mathit{history}) = \mathit{CLOSE}$$

Samenvatting

Het verwerken van gestructureerde hypermedia documenten

Dit proefschrift behandelt het gebruik van gestructureerde documenten voor hypermedia, met de nadruk op toepassingen waarbij verschillende mediafragmenten gesynchroniseerd gepresenteerd moeten worden. Het onderwerp wordt vanuit de verschillende perspectieven van een aantal onderzoeksgebieden belicht.

Het eerste deel van het proefschrift schetst de fundamentele onderzoeksvragen die onderzocht worden in de onderzoeksgebieden *electronic publishing*, *hypertext*, *multimedia* en het *World Wide Web* en de belangrijkste documentmodellen die uit dat onderzoek zijn voortgekomen. Vervolgens geeft het een overzicht van de mogelijkheden om ideeën die in het ene onderzoeksgebied zijn ontwikkeld toe te passen in de andere onderzoeksgebieden, en worden de voor- en nadelen van de verschillende modellen behandeld. De nadruk ligt op een publicatiemodel dat is ontwikkeld in de electronic publishing gemeenschap, en op het toepassen van dit oorspronkelijk voor tekst ontwikkelde model op interactieve multimedia documenten op het World Wide Web. In dit "multiple delivery publishing" model staat het gebruik van gestructureerde documenten en zogenaamde "style sheets" centraal om op een efficiënte manier documenten in meerdere varianten te kunnen publiceren. Het model beoogt bijvoorbeeld het aanpassen van de opmaak van grote hoeveelheden documenten te vereenvoudigen, maar ook het efficiënt aanpassen van documenten aan verschillende software en hardware (denk bijvoorbeeld aan het geschikt maken van dezelfde informatie voor presentatie op papier, op een PC-scherm en op een mobiele telefoon).

In het tweede deel zien we de belangrijkste onderwerpen van deel 1 terug in de formele Object-Z specificatie van het aan het CWI ontwikkelde Amsterdam Hypermedia Model (AHM). Dit model wordt gespecificeerd in termen van het — in de hypertext-gemeenschap bekende — Dexter referentie model. Ook wordt het momenteel meest gangbare, op style sheets gebaseerde, document-transformatie model formeel gespecificieerd. De formele notatie wordt vooral gebruikt om deze modellen eenduidig en op het juiste abstractie niveau te kunnen beschrijven, en om een aantal openstaande kwesties duidelijk te kunnen illustreren. Er wordt inzicht gegeven in de geschiktheid van de gebruikte formele methode voor het modelleren van tijdgebaseerde hypermedia applicaties en de voor- en nadelen van de belangrijkste alternatieve formele methoden.

Het derde deel van het proefschrift behandelt dezelfde onderwerpen nogmaals, maar dan vanuit het perspectief van de architectuur van de software die nodig is om gestructureerde hypermedia documenten te verwerken. Het behandelt de verschillen tussen de architecturen die in de verschillende onderzoeksgemeenschappen zijn ontwikkeld, en de mogelijkheden en problemen die ontstaan wanneer deze architecturen geïntegreerd worden. Twee concrete voorbeelden van software architecturen voor gestructureerde hypermedia documenten worden nader toegelicht. Eerst wordt het aan de VU ontwikkelde DejaVu software raamwerk behandeld, waarbij de nadruk ligt op de toepassing van gestructureerde Web documenten in de context van de vele multimedia componenten van het DejaVu raamwerk. Tenslotte wordt de aan het CWI ontwikkelde Berlage omgeving voor gestructureerde hypermedia document transformaties behandeld.

Bibliography

[1] ACM. *Hypertext '87 Proceedings*, Chaper Hill, North Carolina, November 13-15, 1987. ACM Press.

[2] ACM. *Fifth ACM Conference on Hypertext Proceedings (Hypertext'93)*, Seattle, Washington, November 14-18, 1993. ACM Press.

[3] ACM. *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)*, September 18–23, 1994, Edinburgh, 1994. ACM Press.

[4] ACM. *Proceedings of the 1st ACM international conference on Digital libraries*, Bethesda, MD USA, March 20-23, 1996.

[5] ACM. *Proceedings of the Seventh ACM Conference on Hypertext (Hypertext'96)*, March 1996, Washington D.C., 1996. ACM Press.

[6] ACM. *The Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, Pittsburgh, PA, June 20-24, 1998. ACM Press. Edited by Kaj Grønbæck, Elli Mylonas and Frank M. Shipman III.

[7] ACM. *Proceedings of the 10th ACM conference on Hypertext and Hypermedia*, Darmstadt, Germany, February 21-25, 1999. Edited by Klaus Tochterman, Jorg Westbomke, Uffe K. Will and John J. Leggett.

[8] Adobe Systems Incorporated. Portable Document Format (PDF). See http://www.adobe.com/products/acrobat/adobepdf.html.

[9] Adobe Systems Incorporated. PostScript. See http://www.adobe.com/products/postscript/main.html.

[10] Robert Akscyn, Donald McCracken, and Elise Yoder. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. In *Hypertext '87 Proceedings* [1], pages 1–20.

[11] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–844, November 1983.

[12] Kenneth M. Anderson, Richard N. Taylor, and Jr. E. James Whitehead. Chimera: Hypertext for Heterogeneous Software Environments. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)* [3], pages 94–107.

[13] Keith Andrews, Frank Kappe, and Hermann Maurer. Serving Information to the Web with Hyper-G. In *The Third International World-Wide Web Conference: Technology, Tools and Applications*, Darmstadt, Germany, April 10-14, 1995.

[14] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, and Lauren Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendations are available at http://www.w3.org/TR, October 1, 1998.

[15] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Addison-Wesley Publishing Company, 3rd edition, 2000.

[16] Cristina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A Survey of QoS Architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, 6(3), May 1998.

[17] Jeff Ayars, Dick Bulterman, Aaron Cohen, Ken Day, Erik Hodge, Philipp Hoschka, Eric Hyche, Muriel Jourdan, Kenichi Kubota, Rob Lanphier, Nabil Layaïda, Philippe Le Hégaret, Thierry Michel, Debbie Newman, Jacco van Ossenbruggen, Lloyd Rutledge, Bridie Saccocio, Patrick Schmitz, and Warner ten Kate. Synchronized Multimedia Integration Language (SMIL) 2.0 Specification. Work in progress. W3C Working Drafts are available at http://www.w3.org/TR, 21 September 2000.

[18] Mark Bartel, John Boyer, Barb Fox, and Ed Simon. XML-Signature Core Syntax. W3C Candidate Recommendations are available at http://www.w3.org/TR, 31 October 2000.

[19] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison Wesley, 1998.

[20] Zeev Becker and Daniel Berry. `triroff`, an adaptation of the device-independent `troff` for formatting tri-directional text. *Electronic Publishing—Origination, Dissemination, and Design*, 2(3):119–142, October 1989.

[21] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings of the 11th ICALP*, volume 172 of *Lecture Notesin Computer Science*, pages 82–94, Antwerpen, 1984. Springer-Verlag.

[22] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. RFC 1866 (http://www.ietf.org/rfc/rfc1866.txt), November 1995.

[23] T. Berners-Lee, R Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.

[24] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, August 1998. RFC 2396.

[25] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL), December 1994. RFC 1738, updated by RFC 2396.

[26] Tim Berners-Lee. Information Management: A Proposal. Available at http://www.w3.org/History/1989/proposal.html, May 1990.

[27] Michael Bieber and Tomás Isakowitz (Guest editors). Special issue on designing hypermedia applications. *Communications of the ACM*, 38(8), August 1995.

[28] Gordon Blair, Lynne Blair, Howard Bowman, and Amanda Chetwynd. *Formal Specification of Distributed Multimedia Systems*. UCL Press, 1998.

[29] Dirk Bolier and Anton Eliëns. Sim — a C++ library for discrete event simulation. Technical Report IR-367, Vrije Universiteit, Amsterdam, November 1994.

[30] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[31] T. Bolognesi, F. Lucidi, and S. Trigila. From Timed Petri Nets to timed LOTOS. In *Protocol Specification, Testing and Verification, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium*, pages 395–408, Ottawa, Ont., Canada, 1990. North-Holland.

[32] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One, September 1993. RFC-1521, obsoletes RFC-1341.

[33] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendations are available at http://www.w3.org/TR, May 12, 1998.

[34] Rodrigo A. Botafogo, Ehud Rivlin, and Ben Shneiderman. Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics. *ACM Transactions on Information Systems*, 10(2):142–180, April 1992.

[35] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (http://www.ietf.org/rfc/rfc2205.txt), September 1997.

[36] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. International computer science series. Addison-Wesley, 2nd edition, 1990.

[37] Tim Bray, Dave Hollander, and Andrew Layman (Editors). Namespaces in XML. W3C Recommendations are available at http://www.w3.org/TR, Januari, 14, 1999.

[38] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 Specification, February 10, 1998. W3C Recommendations are available at http://www.w3.org/TR.

[39] Pearl Brereton, David Budgen, and Geoff Hamilton. Hypertext: The Next Maintenance Mountain. *Computer*, pages 49–55, December 1998.

[40] S.D. Brookes, C.A.R. Hoare, , and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[41] P. J. Brown. Using logical objects to control hypertext appearance. *Electronic Publishing—Origination, Dissemination, and Design*, 4(2):109–118, June 1991.

[42] P. J. Brown and Heather Brown. Embedded or separate hypertext mark-up: is it an issue? *Electronic Publishing—Origination, Dissemination, and Design*, 8(1):1–13, March 1995.

[43] Peter J. Brown. Turning Ideas into Products: The Guide System. In *Hypertext '87 Proceedings* [1], pages 33–40.

[44] P Brusilovsky. Efficient techniques for adaptive hypermedia. *Intelligent hypertext: Advanced techniques for the World Wide Web. Lecture Notes in Computer Science*, 1326:12–30, 1997. Edited by C. Nicholas and J. Mayfield.

[45] John F. Buford. Evaluation of a Query Language for Structured Hypermedia Documents. In *Proceedings of DAGS 95 — Electronic Publishing and the Information Superhighway*, Boston, MA, May 30 - June 2, 1995.

[46] John F. Buford. Evaluating HyTime: an examination and implementation experience. In *Proceedings of the Seventh ACM Conference on Hypertext (Hypertext'96)* [5], pages 105 – 115.

[47] John F. Buford and Lloyd Rutledge. Third-Generation Distributed Hypermedia Systems. In William I. Grosky, Ramesh Jain, and Rajiv Mehrotra, editors, *Handbook of Multimedia Information Management*, pages 449–473. Prentice Hall, 1997.

[48] John F. Buford, Lloyd Rutledge, John Rutledge, and Can Keskin. HyOctane: A HyTime Engine for an MMIS. In *ACM Multimedia*, pages 129–136, Anaheim, CA, 1993.

[49] John F. Buford, Lloyd Rutledge, John Rutledge, and Can Keskin. HyOctane: A HyTime Engine for an MMIS. *Multimedia Systems Journal*, 1(4), February 1994.

[50] John F. Koegel Buford, editor. *Multimedia Systems*. ACM SIGGRAPH Book Series. ACM Press Books/Addison-Wesley Publishing Company, 1994.

[51] D.C.A. Bulterman, L. Hardman, J. Jansen, K.S. Mullender, and L. Rutledge. GRiNS: A GRaphical INterface for Creating and Playing SMIL Documents. In *Seventh International World Wide Web Conference*, Brisbane, Australia, April 14-18, 1998.

[52] Dick C.A. Bulterman, Guido van Rossum, and Robert van Liere. A Structure for Transportable, Dynamic Multimedia Documents. In *Proceedings of the Summer 1991 USENIX Conference*, pages 137–155, Nashville, TN, 1991.

[53] V. Bush. As we may think. *Atlantic Monthly*, July 1945.

[54] Michael Casey and Paris Smaragdis. NetSound. In *Proceedings of the International Computer Music Conference*, 1996.

[55] Warwick Cathro. Metadata: An Overview. In *Standards Australia Seminar: Matching Discovery and Recovery*, August 1997.

[56] James Clark. SP — An SGML parser. Available at http://www.jclark.com/sp/, October 1995.

[57] James Clark. Jade — James' DSSSL Engine. Available at http://www.jclark.com/jade/, 1997.

[58] James Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendations are available at http://www.w3.org/TR/, November, 16, 1999.

[59] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendations are available at http://www.w3.org/TR/, Noveber,16, 1999.

[60] W. Clinger and J. Rees (Editors). Revised(4) Report on the Algorithmic Language Scheme (R4RS). http://www.cs.indiana.edu/scheme-repository/R4RS/r4rs_toc.html, 1991. Published as ACM Lisp Pointers IV, July-September 1991, and also as MIT AI Memo 848b.

[61] Ken A. L. Coar and D.R.T. Robinson. The WWW Common Gateway Interface Version 1.1. Internet Draft, work in progress. Available at: http://www.ietf.org/internet-drafts/draft-coar-cgi-v11-02.txt, April 2, 1999.

[62] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9), September 1987.

[63] Daniel W. Connolly. An Evaluation of the World Wide Web with respect to Engelbart's Requirements. http://www.w3.org/Architecture/NOTE-ioh-arch, February 1998.

[64] A. Coombes. An Interval Logic for Modelling Time in Z. Technical report, University of York, Dept. Computer Science, 1990.

[65] Copernican Solutions Incorporated. DAE SDK Version 1.0. See http://www.copsol.com.

[66] James O. Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley, 1992.

[67] Davenport Group. Documentation Version 1.0 for DocBook 3.0, 1997. See http://www.ora.com/davenport/dbdoc/.

[68] Hugh Davis. To Embed or Not to Embed... In *Communications of the ACM* [27], pages 108–109.

[69] Hugh Davis, Wendy Hall, Ian Heath, Gary Hill, and Rob Wilkins. Towards an Integrated Information Environment with Open Hypermedia Systems. In *Proceedings of the ACM European Conference on Hypertext (ECHT'92)*, pages 181–190, Milano, Italy, November 30 - December 4, 1992. ACM, ACM Press.

[70] Hugh C. Davis, Simon Knight, and Wendy Hall. Light Hypermedia Link Services: A Study of Third Party Application Integration. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)* [3], pages 41–50.

[71] L. Hardman D.C.A. Bulterman, L. Rutledge and J. van Ossenbruggen. Supporting Adaptive and Adaptable Hypermedia Presentation Semantics. In *The 8th IFIP 2.6 Working Conference on Database Semantics (DS-8): Semantic Issues in Multimedia Systems*, Rotorua, New Zealand, 5-8 January 1999, 1999.

[72] Paul de Bra, Geert-Jan Houben, and Hongjing Wu. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *Proceedings of the 10th ACM conference on Hypertext and Hypermedia* [7], pages 147–156. Edited by Klaus Tochterman, Jorg Westbomke, Uffe K. Will and John J. Leggett.

[73] Stephen Deach. Extensible Stylesheet Language (XSL) Specification. W3C Candidate Recommendations are available at http://www.w3.org/TR, 21 November 200.

[74] Serge Demeyer, Theo Dirk Meijler, Oscar Nierstrasz, and Patrick Steyaert. Design Guidelines for 'Tailorable' Frameworks. In *Communications of the ACM* [97], pages 60–64.

[75] Steve DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink). W3C Proposed Recommendations are available at http://www.w3.org/TR, 20 December 2000.

[76] Steve DeRose, Eve Maler, and Jr. Ron Daniel. XML Pointer Language (XPointer) Version 1.0. W3C Candidate Recommendations are available at http://www.w3.org/TR, 8 January 2001.

[77] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. W3C Notes are available at http://www.w3.org/TR, August, 19, 1998.

[78] Michel Diaz and Patrick Sénac. Time Stream Petri Nets: A Model for Multimedia Streams Synchronization. In *Proceedings of Multimedia Modelling '93, Singapore*, pages 257–273, November 1993.

[79] D.A. Duce, D.J. Duke, F.Faconti, I. Herman, and M. Massink. PREMO: A case study in formal methods and multimedia system specification. Technical Report INS-R9708, CWI, November 1997.

[80] Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z Specification Language: Version 1. Technical Report 91-1, Software Verification Research Centre, Department of Computer Science, the University of Queensland, Australia, April 1991. Reprinted by ISO JTC1 WG7 as document number 372.

[81] Roger Duke and Gordon Rose. Modelling Object Identity. Technical Report 92-11, Software Verification Research Centre, Department of Computer Science, the University of Queensland, Australia, 1992.

[82] Roger Duke, Gordon Rose, and Graeme Smith. Object-Z: a Specification Language Advocated for the Description of Standards. Technical Report 94-45, Software Verification Research Centre, Department of Computer Science, the University of Queensland, Australia, December 1994.

[83] David Durand and Paul Kahn. MAPA: A System for Inducing and Visualizing Hierarchy Websites. In *The Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia* [6], pages 66–76. Edited by Kaj Grønbæck, Elli Mylonas and Frank M. Shipman III.

[84] Martin J. Dürst. Coordinate-independent font description using Kanji as an example. *Electronic Publishing—Origination, Dissemination, and Design*, 6(3):133–143, September 1993.

[85] Jr. E. James Whitehead and Meredith Wiggins. WEBDAV: IETF Standard for Collaborative Authoring on the Web. *IEEE Internet Computing*, pages 34–40, September/October 1998.

[86] Anton Eliëns. *DLP — A Language for Distributed Logic Programming*. Wiley & Sons, 1992.

[87] Anton Eliëns. Hush — a C++ API for Tcl/Tk. Technical Report IR-366, Vrije Universiteit, Amsterdam, November 1994.

[88] Anton Eliëns. Hush: A C++ API for Tcl/Tk. *The X Resource*, (14):111–155, April 1995.

[89] Anton Eliëns. *Principles of Object-Oriented Software Development*. Addison-Wesley, 1995.

[90] Anton Eliëns. *Principles of Object-Oriented Software Development*. Addison-Wesley, 2nd edition, 2000.

[91] Anton Eliëns, Frank Niessink, Bastiaan Schönhage, Jacco van Ossenbruggen, and Paul Nash. Support for Business Process Redesign – Simulation, Hypermedia and the Web. In *Euromedia 96: Telematics in a Multimedia Environment, December 19-21, London. United Kingdom*, pages 193–200. The Society for Computer Simulation International, December 1996.

[92] Anton Eliëns, Jacco van Ossenbruggen, and Bastiaan Schönhage. Animating the Web — An SGML-based Approach. In *Proceedings of the International Conference on 3D and Multimedia on the Internet, WWW and Networks, 17-18 April 1996, Bradford.* British Computer Society, April 1996.

[93] Anton Eliëns, Jacco van Ossenbruggen, and Bastiaan Schönhage. Animating the Web — An SGML-based Approach. In Rae Earnshaw and John Vince, editors, *The Internet in 3D — Information, Images and Interaction*, pages 75–96. Academic Press, 1997.

[94] Anton Eliëns, Martijn van Welie, Jacco van Ossenbruggen, and Bastiaan Schönhage. Jamming (on) the Web. In *Proceedings of the 6th International World Wide Web Conference — Everone, Everything Connected*, pages 419–426. O'Reilly and Associates, Inc., April 1997. April 7-11, 1997, Santa Clara, California USA.

[95] D.C. Engelbart. A Conceptual Framework for the Augmentation of Man's Intellect. *Vistas in Information Handling*, 1(9), 1963.

[96] Douglas C. Engelbart. Knowledge-Domain Interoperability and an Open Hypermedia System. In *Proceedings of the Conference on Computer-Supported Collaborative Work*, pages 143–156, Los Angeles, CA, October, 7-10, 1990.

[97] Mohamed E. Fayad and Douglas C. Schmidt (Guest Editors). Special Issue on Object-Oriented Application Frameworks. *Communications of the ACM*, 40(10):32–87, October 1997.

[98] Jon Ferraiolo. Scalable Vector Graphics (SVG) 1.0 Specification. W3C Candidate Recommendations are available at http://www.w3.org/TR, 2 November 2000.

[99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 - IETF Draft Standard, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[100] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steel, and P. Yanker. Query by Image and Video Content: The QBIC Sytem. *IEEE Computer*, 28(9), 1995.

[101] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, E. Sink, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, June 1999. Available at http://www.ietf.org/rfc/rfc2617.txt.

[102] C.J. Fridge. Specification and Verification of Real-Time Behaviour Using Z and RTL. In J. Vytopil, editor, *Proceedings of the Second International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 393–409. Springer, 1992.

[103] Richard K. Furuta. Important Papers in the History of Document Preparation Systems: Basic Sources. *Electronic Publishing — Origination, Dissemination and Design*, 5(1):19–44, March 1992.

[104] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1994.

[105] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Inc., 1st edition edition, December 1994.

[106] Franca Garzotto, Luca Mainetti, and Paolo Paolini. Hypermedia Design, Analysis, and Evaluation Issues. In *Communications of the ACM* [27], pages 74–86.

[107] Google, Inc. Google Search. See http://www.google.com/.

[108] K. Grønbæck and R. Trigg. Design Issues for a Dexter-Based Hypermedia System. *Communications of the ACM*, 37(2):40–49, February 1994.

[109] Kaj Grønbæck. OHS Interoperability — Issues Beyond the Protocol. In Wiil [240]. Proceedings are published as Technical Report CS-98-01 of Aalborg University, Denmark.

[110] IETF/W3C XML-Signature Working Group. More information on http://www.w3.org/Signature/.

[111] W3C SYMM Working Group. W3C Activity on Synchronized Multimedia. More information at http://www.w3.org/AudioVideo/.

[112] F. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[113] F. Halasz and M. Schwarz. The Dexter Hypertext Reference Model. In *NIST Hypertext Standardization Workshop*, pages 95–133, January 1990.

[114] F. Halasz and M. Schwarz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994. Edited by K. Grønbæck and R. Trigg.

[115] Frank G. Halasz. ”Seven Issues”: Revisited. Keynote Address, Hypertext ’91 Conference, San Antonio, Texas, December 18, 1991. Transcript available at: http://www.parc.xerox.com/spl/projects/halasz-keynote/.

[116] Yannis Haralambous. Typesetting Khmer. *Electronic Publishing—Origination, Dissemination, and Design*, 7(4):197–215, December 1994.

[117] Yannis Haralambous. The traditional Arabic typecase extended to the Unicode set of glyphs. *Electronic Publishing—Origination, Dissemination, and Design*, 8(2/3):111–123, June/September 1995.

[118] L. Hardman and D. C. A. Bulterman. Document Model Issues for Hypermedia. In William I. Grosky, Ramesh Jain, and Rajiv Mehrotra, editors, *Handbook of Multimedia Information Management*, pages 39 – 68. Prentice Hall, 1997.

[119] L. Hardman, D. C. A. Bulterman, and G.van Rossum. The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *Communications of the ACM*, 37(2):50–62, February 1994.

[120] L. Hardman, D.C.A. Bulterman, and G. van Rossum. Links in hypermedia: the requirement for context. In *Fifth ACM Conference on Hypertext Proceedings (Hypertext’93)* [2], pages 183–191.

[121] Lynda Hardman. *Modelling and Authoring Hypermedia Documents*. PhD thesis, University of Amsterdam, 1998. ISBN: 90-74795-93-5, also available at http://www.cwi.nl/∼lynda/thesis/.

[122] Lynda Hardman, Dick C. A. Bulterman, and Guido van Rossum. The Amsterdam Hypermedia Model: Extending Hypertext to Support REAL Multimedia. *Hypermedia Journal*, 5(1):47–69, July 1993. Revised version of CWI Report CS-R9306.

[123] Lynda Hardman, Jacco van Ossenbruggen, Lloyd Rutledge, K. Sjoerd Mullender, and Dick C. A. Bulterman. Do You Have the Time? Composition and Linking in Time-based Hypermedia. In *Proceedings of the 10th ACM conference on Hypertext and Hypermedia* [7], pages 189–196. Edited by Klaus Tochterman, Jorg Westbomke, Uffe K. Will and John J. Leggett.

[124] Lynda Hardman, Guido van Rossum, and Dick C. A. Bulterman. Structured Multimedia Authoring. In *ACM Multimedia*, pages 283 – 289, Anaheim, CA, 1993. Revised version of CWI Report CS-R9304.

[125] Gary Hill and Wendy Hall. Extending the Microcosm Model to a Distributed Environment. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)* [3], pages 32–40.

[126] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–667, 1978.

[127] Philipp Hoschka. Known Errors in the SMIL 1.0 Specification. Available at http://www.w3.org/AudioVideo/SMIL/errata, 2000.

[128] Yang hua Chu, Philip DesAutels, Brian LaMacchia, and Peter Lipp. PICS Signed Labels (DSig) 1.0 Specification. http://www.w3.org/TR/REC-DSig-label/, May 27, 1998. W3C Recommendations are available at http://www.w3.org/TR.

[129] International Consultative Committee for Telephony and Telegraphy. Recommendation Z.100 — CCITT specification and description language (SDL), 1988.

[130] International Organization for Standardization. Information Processing — Text and Office Information Systems — Standard Generalized Markup Language (SGML), 1986. International Standard ISO 8879:1986.

[131] International Organization for Standardization. Information Processing — Text and Office Information Systems — Office Document Architecture (ODA) and Interchange Format, 1989. International Standard ISO 8613, Parts 1–8.

[132] International Organization for Standardization. Information processing systems — Open systems interconnection — Estelle — A formal description technique based on an extended state transition model, 1989. International Standard ISO 9074:1989.

[133] International Organization for Standardization. Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, 1989. International Standard ISO 8807:1989.

[134] International Organization for Standardization. Information Technology — Hypermedia/Time-based Structuring Language (HyTime), August 1997. International Standard ISO 10744:1997 (HyTime Second Edition).

[135] International Organization for Standardization/International Electrotechnical Commission . Information technology – Processing languages – Standard Page Description Language: ISO/IEC 10180:1995 - SPDL, 1995.

[136] International Organization for Standardization/International Electrotechnical Commission. Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL), 1996. International Standard ISO/IEC 10179:1996.

[137] International Organization for Standardization/International Electrotechnical Commission. Information technology – Coding of multimedia and hypermedia information – Part 5: Support for base-level interactive applications, 1997. International Standard ISO/IEC 13522-5:1997 (MHEG-5).

[138] International Organization for Standardization/International Electrotechnical Commission. MPEG-7: Context and Objectives, 1998. Work in progress.

[139] International Organization for Standardization/International Electrotechnical Commission. Information Technology — Document Description and Processing Languages: ISO/IEC FCD 13250:1999 - Topic Maps, 1999.

[140] International Organization for Standardization/International Electrotechnical Commission. Information technology – Coding of moving pictures and audio, 1999. International Standard ISO/IEC 14496:1999 (MPEG-4).

[141] Tomás Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, August 1995.

[142] Ralph E. Johnson. Frameworks = (components + patterns). In *Communications of the ACM* [97], pages 39–42.

[143] Wendy Johnston. A Type Checker for Object-Z. Technical Report 96-24, Software Verification Research Centre, School of Information Technology, the University of Queensland, Brisbane 4072. Australia, July 1996.

[144] C. B. Jones. *Systematic Software Development using VDM*. Prentie-Hall, Englewood Cliffs, New Jersey, 1990.

[145] M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismaïl, and L. Tardif. Madeus, an Authoring Environment for Interactive Multimedia Documents. In *Proceedings of ACM Multimedia '98*, Bristol UK, 1998.

[146] F. Kappe, H. Maurer, and N. Sherbakov. Hyper-G - A Universal Hypermedia System. *Journal of Educational Multimedia and Hypermedia*, 2(1):39–66, 1993.

[147] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Inc., 2nd edition, 1988.

[148] M.Y. Kim and J. Song. Multimedia Documents with Elastic Time. In *Proceedings of ACM Multimedia '95*, pages 143–154, San Francisco CA, 1995.

[149] Donald E. Knuth. *TeX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley Publishing Company, 1986.

[150] Jukka Korpela. Lurching Toward Babel: HTML, CSS, and XML. *IEEE Computer*, 31(7), July 1998.

[151] R. Koymans. Specifying Real-time Properties with Metric Temporal Logic. *Real-Time Systems*, 2:255–299, 1990.

[152] Daniel LaLiberte and Alan Braverman. A Protocol for Scalable Group and Public Annotations. In *The Third International World-Wide Web Conference: Technology, Tools and Applications*, Darmstadt, Germany, April 10-14, 1995.

[153] Leslie Lamport. *LaTeX - A Document Preparation System*. Addison-Wesley Publishing Company, 1985.

[154] Kevin Lano and Howard Haughton, editors. *Object-Oriented Specification Case Studies*. The Object-Oriented series. Prentice Hall International, 1994.

[155] Ora Lassila and Ralph R. Swick (Editors). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendations are available at http://www.w3.org/TR, February, 22, 1999.

[156] Håkon W. Lie and Bert Bos. Cascading Style Sheets, level 1. W3C Recommendations are available at http://www.w3.org/TR, December 1996.

[157] Mark A. Linton, John M. Vlissides, and Paul R. Calder. Composing User Interfaces with InterViews. *IEEE Computer*, 22(2):8–22, 1989.

[158] T. D. C. Little and A. Ghafor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[159] Thomas D. C. Little. Time-based Media Representation and Delivery. In Buford [50], pages 175–200.

[160] L. Logrippo, M. Faci, and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23:325–342, 1992.

[161] Mark Lutz. *Programming Python*. O'Reilly & Associates, Inc., 1st edition, 1996.

[162] Macromedia. Director. http://www.macromedia.com/software/director/.

[163] William C. Mann, Christian M. I. M. Matthiesen, and Sandara A. Thompson. Rhetorical Structure Theory and Text Analysis. Technical Report ISI/RR-89-242, Information Sciences Institute, University of Southern California, November 1989.

[164] C. Marshall, F. Shipman, and J. Coombs. VIKI: Spatial Hypertext Supporting Emergent Structure. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)* [3], pages 13–23.

[165] H. Maurer. *HyperWave: The Next Generation Web Solution*. Addison-Wesley, UK, 1996.

[166] Sebastiaan A. Megens. More Music in Hush. Master's thesis, Vrije Universiteit, Amsterdam, August 1996.

[167] Norman Meyrowitz. Intermedia: The architecture and construction of an object-oriented hypemedia system and applications framework. In *Conference proceedings on Object-oriented programming systems, languages and applications*, pages 186–201, Portland, OR USA, September 29 - October 2, 1986.

[168] Jim Miller, Paul Resnick, and David Singer. Rating Services and Rating Systems (and Their Machine Readable Descriptions), Version 1.1, October, 31, 1996. W3C Recommendations are available at http://www.w3.org/TR.

[169] R. Milner. A Calculus of Communicating Systems. *Lecture Notes in Computer Science*, 92, 1980.

[170] R. Moats. URN Syntax, May 1997. RFC 2141 (http://www.ietf.org/rfc/rfc2141.txt).

[171] Jocelyne Nanard and Marc Nanard. Should Anchors Be Typed Too — An Experiment with MacWeb. In *Fifth ACM Conference on Hypertext Proceedings (Hypertext'93)* [2], pages 51–62.

[172] National Center for Supercomputing Applications, University of Illinois. NCSA Mosaic (TM) for the X Window System. http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/.

[173] Steven R. Newcomb. GroveMinder. In *The 4th International HyTime Conference*, August 19-20, 1997, Montreal, Quebec, Canada, 1997.

[174] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. W3C Note for discussion only. Available at http://www.w3.org/Protocols/HTTP/Performance/Pipeline, 1997.

[175] M. Noik. Exploring Large Hyperdocuments: Fisheye Views of Nested Networks. In *Fifth ACM Conference on Hypertext Proceedings (Hypertext'93)* [2], pages 192–201.

[176] Peter J. Nürnberg, John J. Leggett, and Uffe K. Will. An Agenda for Open Hypermedia Research. In *The Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia* [6]. Edited by Kaj Grønbæck, Elli Mylonas and Frank M. Shipman III.

[177] Open Hypermedia Systems Working Group. See http://www.csdl.tamu.edu/ohs/.

[178] Hidetaka Ohto and Johan Hjelm. CC/PP exchange protocol based on HTTP Extension Framework. W3C Note for discussion only. Available at http://www.w3.org/TR/NOTE-CCPPexchange, 24 June 1999.

[179] J.K. Ousterhout. Tcl: An Embeddable Command Language. In *USENIX*, 1990.

[180] J.K. Ousterhout. An X11 Toolkit Based on the Tcl Language. In *USENIX*, 1991.

[181] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science*, pages 46–57, Los Alamitos, CA, 1977. IEEE Computer Society Press.

[182] J. Postel and J. Reynolds. FILE TRANSFER PROTOCOL (FTP). RFC 959, October 1985.

[183] Wolfgang Pree. *Design Patterns for Object-Oriented Software Development*. ACM Press Books. Addison-Wesley, 1995.

[184] D. Raggett. Document Type Definition for the HyperText Markup Language (HTML DTD). Expired Internet Draft, Part of the HyperText Markup Language Specification Version 3.0, March 1995.

[185] Dave Raggett. HTML 3.2 Reference Specification. W3C Recommendations are available at http://www.w3.org/TR, Januari, 14, 1997.

[186] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendations are available at http://www.w3.org/TR, April, 24, 1998.

[187] C. Ramchandani. Analysis of Asynchronous Concurrent Systems by Times Petri Nets. Technical Report TR 120 (Project MAC), MIT, February 1974.

[188] RealNetworks, Inc. RealPlayer G2. See http://www.real.com/products/player/.

[189] L. Rutledge, L. Hardman, and J. van Ossenbruggen. Evaluating SMIL: Three User Case Studies. In *Proceedings of ACM Multimedia*, Orlando, Florida, USA, November 1999. Poster Paper.

[190] L. Rutledge, L. Hardman, and J. van Ossenbruggen. The Use of SMIL: Multimedia Research Currently Applied on a Global Scale. In *Proceedings of Multimedia Modeling 99 (MMM 99)*, pages 1–17, Ottawa, Canada, October 4-6, 1999. World Scientific.

[191] Lloyd Rutledge. Evaluations of and Extentsions to a Second Generation Hypermedia Model. Sc.D. thesis, University of Massachusetts Lowell, 1996.

[192] Lloyd Rutledge, Brian Bailey, Jacco van Ossenbruggen, Lynda Hardman, and Joost Geurts. Generating Presentation Constraints from Rhetorical Structure. In *Proceedings of the 11th ACM conference on Hypertext and Hypermedia*, pages 19–28, San Antonio, Texas, USA, May 30 – June 3, 2000. ACM.

[193] Lloyd Rutledge, Lynda Hardman, Jacco van Ossenbruggen, and Dick C.A. Bulterman. Implementing Adaptability in the Standard Reference Model for Intelligent Multimedia Presentation Systems. In *The International Conference on Multimedia Modeling*, pages 12–20, 12-15 October 1998.

[194] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C. A. Bulterman. Mix'n'Match: Exchangeable Modules of Hypermedia Style. In *Proceedings of the 10th ACM conference on Hypertext and Hypermedia* [7], pages 179–188. Edited by Klaus Tochterman, Jorg Westbomke, Uffe K. Will and John J. Leggett.

[195] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. A Framework for Generating Adaptable Hypermedia Documents. In *Proceedings of ACM Multimedia*, pages 121–130, Seattle, Washington, November 1997. ACM Press.

[196] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Cooparative use of MHEG and HyTime in Hypermedia Environments. In Jean-Pierre Balbe, Alain Lelu, Marc Nanard, and Imad Saleh, editors, *Actes de la 4e conférence internationale Hypertextes et Hypermédias — réalisations, outils & méthodes*, number (1)2-3-4 in Hypertextes et Hypermédias, pages 57–73, Paris, September 1997. Editions HERMES.

[197] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Generic Hypermedia Structure and Presentation Specification. In *ICCC/IFIP Conference — Electronic Publishing '97*, April 1997.

[198] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Practical Application of Existing Hypermedia Standards and Tools. In *ACM Digital Libraries (DL'98)*, pages 191–199, June 1998.

[199] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Structural Distinctions Between Hypermedia Storage and Presentation. In *Proceedings of ACM Multimedia*, pages 145–150. ACM Press, November 1998.

[200] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, and Dick C.A. Bulterman. Anticipating SMIL 2.0: The Developing Cooperative Infrastructure for Multimedia on the Web. In *Proceedings of The Eighth International World Wide Web Conference (WWW8)*, May 1999.

[201] Hans Albrecht Schmid. Systematic Framework Design by Generalization. In *Communications of the ACM* [97], pages 48–51.

[202] Douglas C. Schmidt. *Pattern Languages of Program Design*, chapter Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching. Addison-Wesley, 1 edition, 1995. Editied by James O. Coplien and Douglas C. Schmidt.

[203] Patrick Schmitz and Aaron Cohen. SMIL Animation. Work in progress. W3C Working Drafts are available at http://www.w3.org/TR, 31 July 2000.

[204] Patrick Schmitz, Jin Yu, and Peter Santangeli. Timed Interactive Multimedia Extensions for HTML (HTML+TIME): Extending SMIL into the Web Browser. W3C Note are available at http://www.w3.org/TR, September 1998.

[205] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889 (http://www.ietf.org/rfc/rfc1889.txt), January, 25, 1996.

[206] H. Schulzrinne, A. Rao, and R. Lanphier. Real-Time Streaming Protocol (RTSP). RFC 2326 (http://www.ietf.org/rfc/rfc2326.txt), April 1998.

[207] D. Schwabe, G. Rossi, and S.D.J. Barbosa. Systematic Application design with OOHDM. In *Seventh ACM Conference on Hypertext (Hypertext '96) Washington DC, March 16-20 1996*, pages 116–128, 1996.

[208] Frank M. Shipman, III, Catherine C. Marshall, and Mark LeMere. Beyond Location Hypertext Workspaces and Non-Linear Views. In *Proceedings of the 10th ACM conference on Hypertext and Hypermedia* [7], pages 121–130. Edited by Klaus Tochterman, Jorg Westbomke, Uffe K. Will and John J. Leggett.

[209] Ben Shneiderman. User Interface Design for the Hyperties Electronic Encyclopedia. In *Hypertext '87 Proceedings* [1], pages 189–194.

[210] Graeme Smith and Ian Hayes. Towards Real-Time Object-Z. Technical Report 99-10, School of Information Technology, the University of Queensland, Brisbane 4072, Australia, February 1999.

[211] Society of Automotive Engineers. Draft SAE Standard Q1. See http://www.mcs.net/∼dken/sae.htm, 1995.

[212] K. Sollins. Architectural Principles of Uniform Resource Name Resolution. RFC 2276 (http://www.ietf.org/rfc/rfc2276.txt), January 1998.

[213] C. M. Sperberg-McQueen and Robert F. Goldstein. HTML to the Max — A Manifesto for Adding SGML Intelligence to the World-Wide Web. In *Proceedings of the Second International World Wide Web Conference '94: Mosaic and the Web*, October 1994.

[214] J. M. Spivey. *The Z Notation: A Reference Manual*. Series in Computer Science. Prentice Hall International, 2nd edition, 1992.

[215] H. Stenzel, K. Kansy, I. Herman, and S. Carson. Premo — An Architecture for presentation of Multimedia Objects in a Open Environment. In *Proceedings of the first Eurographics Symposium on Multimedia, Graz, Austria*, 1994.

[216] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2 edition, 1991.

[217] W.R.T ten Kate, P.J. Deunhouwer, D.C.A. Bulterman, L. Hardman, and L. Rutledge. Presenting Multimedia on the Web and in TV broadcast. In *3rd European Conference on Multimedia Applications, Services and Techniques*, Berlin-Germany, May, 26-28, 1998.

[218] The Productivity Works. LpPlayer. See http://www.prodworks.com/lpplayer.htm.

[219] Randall Trigg. *A Network-Based Approach to Text Handling for the Online Scientific Community*. PhD thesis, University of Maryland, November 1983. University of Maryland Technical Report, TR-1346.

[220] Matthijs van Doorn and Anton Eliëns. Integrating WWW and Applications, November 1994. ERCIM/W4G — International Workshop on WWW Design Issues, Amsterdam.

[221] Matthijs van Doorn and Anton Eliëns. Integrating Applications and the World Wide Web. *Computer Networks and ISDN Systems*, 27(6):1105–1110, April 1995. Special issue: Proceedings of the Third International World-Wide Web Conference — Technology, Tools and Applications, April 10-14, Darmstadt, Germany.

[222] Jacco van Ossenbruggen and Anton Eliëns. Music in Time-based Hypermedia. In *ECHT'94, The European Conference on Hypermedia Technology*, pages 224–270, September 1994.

[223] Jacco van Ossenbruggen and Anton Eliëns. Bringing Music to the Web. In *Proceedings of the Fourth International World Wide Web Conference — The Web Revolution*, pages 309–314. O'Reilly and Associates, Inc., December 1995.

[224] Jacco van Ossenbruggen, Anton Eliëns, and Lloyd Rutledge. The Role of XML in Open Hypermedia Systems. In Wiil [240]. Proceedings are published as Technical Report CS-98-01 of Aalborg University, Denmark.

[225] Jacco van Ossenbruggen, Anton Eliëns, Lloyd Rutledge, and Lynda Hardman. Requirements for Multimedia Markup and Style Sheets on the World Wide Web. In *The Seventh International World Wide Web Conference (WWW7)*, volume 30 of

Computer Networks and ISDN Systems, pages 694–696. Elsevier Science B.V., April 1998.

[226] Jacco van Ossenbruggen, Anton Eliëns, and Bastiaan Schönhage. Web Applications and SGML. In David F. Brailsford and Richard K. Furuta, editors, *Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography*, volume 8(2 & 3) June and September 1995 of *Electronic Publishing — Origination, Dissemination and Design*, pages 51–62. John Wiley & Sons, Ltd., 1996.

[227] Jacco van Ossenbruggen, Lynda Hardman, Lloyd Rutledge, and Anton Eliëns. Style Sheet Support for Hypermedia Documents. In *The Proceedings of the Eighth ACM Conference on Hypertext and Hypermedia*, pages 216–217, Southampton, UK, April 1997. ACM, ACM Press.

[228] G. van Rossum, J. Jansen, K. S. Mullender, and D.C.A. Bulterman. CMIFed: A Presentation Environment for Portable Hypermedia Documents. In *The First ACM International Conference on Multimedia*, pages 183–188, August 1993.

[229] Barry Vercoe. *Csound, A Manual for the Audio Processing System and Supporting Programs with Tutorials*, 1993. Available via ftp://cecelia.media.mit.edu/pub/Csound/Csound.man.ps.Z.

[230] Philippe Vijghen. Experience of EDI for Documents: The Role of SGML. In *Conference Proceedings of SGML'97*, pages 213–218, December 1997.

[231] W3C. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, June 15, 1998. Edited by Philipp Hoschka. W3C Recommendations are available at http://www.w3.org/TR.

[232] W3C. XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0. W3C Recommendations are available at http://www.w3.org/TR/, January 26, 2000.

[233] W3C. XHTML 1.1 - Module-based XHTML. Work in progress. W3C Working Drafts are available at http://www.w3.org/TR, January 5, 2000.

[234] W3C HTTP-NG activity. Hypertext Transfer Protocol - Next Generation. http://www.w3.org/Protocols/HTTP-NG/.

[235] B. Walter. Timed Petri Nets for Modelling and Analising Protocols with Time. In H. Rudin and C. West, editors, *IFIP Conference on Protocol Specification, Testing and Verification III*, 1983.

[236] Weigang Wang and Jörg M. Haake. Implementation Issues on OHS-based Workflow Services. In *5th Workshop on Open Hypermedia Systems, Hypertext '99, Darmstadt, Germany, February 21-25, 1999*, 1999.

[237] Stuart L. Weibel, Erik Jul, and Keith Shafer. PURLS: Persistent Uniform Resource Locators. Available via http://purl.oclc.org/oclc/purl/summary, 1996.

[238] R. Weiss, A. Duda, and D.K. Gifford. Composition and Search with a Video Algebra. *IEEE Multimedia*, 2(1):12–25, 1995.

[239] U. K. Wiil and J. J. Leggett. The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of the Seventh ACM Conference on Hypertext (Hypertext'96)* [5], pages 140–148.

[240] Uffe Kock Wiil, editor. *4th Workshop on Open Hypermedia Systems*, ACM Hypertext '98, June 20-24, 1998, Pittsburgh, PA, 1998. Proceedings are published as Technical Report CS-98-01 of Aalborg University, Denmark.

[241] N. Yankelovich, N. Meyrowitz, and A. van Dam. Reading and Writing the Electronic Book. *IEEE Computer*, 18(10):15–30, 1985.

Index