Towards Constructing a Flexible Multimedia Environment for Teaching the History of Arts*

Klaus Alfert
Ernst-Erich Doberkat
Corina Kopka
Chair for Software-Technology
University of Dortmund
Germany
{alfert|doberkat|kopka}@ls10.cs.uni-dortmund.de
Phone: +49+231-755-2781
Fax: +49+231-755-2061

Abstract

Multimedia production displays two faces: a multimedia product is the result of programming as well as of publishing. Constructing a multimedia environment for teaching a course in the history of arts suggests that requirements elicitation has many facets and is central to success, hence has to be delt with in a particularly careful way, the more so since we wanted the art historians working in a specific and custom tailored environment. The problem was technically resolved by constructing a dedicated markup language based on XML. We discuss the process of requirements elicitation that led to the markup language and show how this is used as a cornerstone in the development of such an experimental environment.

Keywords and Phrases: Multimedia environment, XML, DTD, Altenberger Dom Markup Language, multimedia in teaching the history of arts, requirements elicitation, translation and interpretation.

*The work reported here has been partially funded by *Universitätsverbund Multimedia Nordrhein-Westfalen*

1 Introduction

Teaching the history of art does not yet fully capitalize on the possibilities offered by multimedia. This is surprising since the history of art is nourished by visual representations. On the other hand, scholarly ideas are presented quite similarly to teaching philosophy, so on second thought the lack of multimedia teaching material may be attributed to not (yet) having a sturdy bridge between ideas and pictures. Of course, teachers in art history have long made use of visual aids, usually displaying slides or pictures. This kind of presenting the necessary visual information is not fully satisfactory to either students or instructors:

- slides and pictures in books are quite static, some of the things to be presented are dynamic, hence change has to be related explicitely. This is achieved usually through a well chosen sequence of pictures, conveying dynamics, albeit only to a limited extent,
- textual and visual information is linked conventionally through narration; students find it difficult reestablishing links when rehearsing the lecture: pictures and explanations (or explications) are gone, taking notes is not easy, in particular when the lecture hall is dark,
- background information is sometime needed, but not alway at the fingertips of instructors or students when a topic is conventionally presented.

A multimedia presentation links text and pictures lively, effectively and in a customizable manner, easing the burden of establishing an mental connection between textual and pictorial representation. It may offer an explanation for the evolution of phenomena, hence capturing dynamics, and it may provide ample background information through hyperlinks. This is our vision.

Certainly we do not want to leave the impression that the conventional mode of teaching is obsolete, bad or plainly unattractive. What we are pleading — and working — for is utilizing multimedia as a device for complementing and embellishing the traditional approach.

We decided entering a cooperation with the Chair for Architectural History at our university. This cooperation aims at the construction of a multimedia representation of the Altenberg Cathedral. The cathedral is a well known and important Gothic church in the Rhineland. It entertains features that are of interest to future architects as well as civil engineers. These students have to take a course in architectural history in which the cathedral features prominently.

From the beginning we understood that both parties would benefit from this project in rather specific ways. The art historians would obtain a multimedial complement to their textbooks, and they would construct a vehicle for teaching their students the in-depth work with the new electronic tools. We as software engineers would help in constructing this vehicle, gaining insight into an intellectually very attractive field together with obtaining a possibility of learning about the architecture of multimedia systems. Most important, we find it exciting to gain some experience in requirements elicitation in a field that has not seen too many computer scientists' footsteps.

The present paper is intended to report on this joint project from the software engineer's perspective. We want to relate where we encountered principal difficulties, how we tried to overcome them, and what we learned from these experiences. In particular we draw the reader's attention to problems of requirements elicitation that emerge in a context where scholars and engineers cooperate. Finally we report on a solution for the construction of a multimedia environment, and we indicate where we endeavour future work.

2 The Perspective

Introducing computer-based training into an art history course is by no means a trivial undertaking. Usually the infrastructure is not comparable to a setting in which engineering courses are taught, but the missing infrastructure may easily be overcome. But the reasons for the non-triviality lie deeper.

Consider these observations we made when cooperating with the group from art history:

- traditionally there is enough pictorial material to be taught from, including graphics used e.g. to analyze a given architectural situation. This material, however, is sometimes not particularly suited for being used in a multimedia environment.

  This is so since it is organized according to principles which are introduced to serve the human user, but not to be applied with a computer; the informal organizational schemes are difficult — if not impossible — to map to a maintainable database design.

- we found it difficult to talk to each other — the way art historians organize their teaching is understandably different from the way computer scientists do, and vice versa.

  Hence requirement elicitation had to be preceded by a phase of learning of mutual understanding.

- the necessity of our doing preparatory technical work was not always immediate to our art historians.

  Hence there was an incentive to have quite early something that could be presented rather in preference to working on seemingly intangible specifications.

We resolved these problems in our cooperation by attempting to agree on the following approach:

1. the target platform is factored out, so that work can be done independent of a particular hardware or software platform,

2. requirements are going to be fixed and described in a sequence of scenarios, which in turn will be translated into a representable form,

3. we intend to construct a development environment that should be user friendly enough so that non-computer scientists will be able to work with it. The environment will eventually be process-based.

3 Related Work

The ZYX model [BK99] is used for semantic modeling of multimedia content. The ZYX multimedia document model has been developed in the context of a database-driven multimedia information system. The model describes multimedia documents based on a tree. The nodes of the tree are presentation elements like videos, images and text or operator elements allowing temporal synchronization, definition of interaction, spatial and audible layouting. The model offers support in structuring the presentation document and combining media objects at layout design level and do not support a didactical design and instantiation phase with logical elements, which should precede synchronization and layouting of media in the software development process of a multimedia teaching system (MMTS).

The process model for the development of multimedia applications in [DEM+99] is based on an analysis and a design phase using the programming model of an authoring system (Director) and on implementing with this specific authoring system. First, an analysis model of the application is constructed and a programming model of the authoring system is derived. Then the relation between these models is analyzed. An instance application model is mapped to an authoring system model using this relation. This approach employs an analysis and design phase trying to fill the gap between the analysis model and implementation using an authoring system. It was developed in parallel to our approach and is rather closely tied to the programming model of an authoring system. It is abstract in the sense that it does not cater for a particular application, and it is quite specific in the use of the programming tools. In contrast to this model, we emphasize the role of domain analysis in the construction of the language we use, and of supporting tools maintaining a platform independent way of representation.

The Relationship Management Methodology *(RMM)* [ISB95], Hypertext Design Model *(HDM)* [GPS93] and Object-Oriented Hypermedia Design Model *(OOHDM)* [SR95] are used for hypermedia or multimedia applications in those areas that display a high degree of domain structure.

Nanard et al. discuss in [FNN96] their SGML-based authoring system PageJockey and present the process between sketches, templates, and final releases of multimedia applications realized in different SGML languages. The technical focus lies on the abstract models and the mapping between them, the methodical on the design process of visual design. They are aiming on an early visual feedback of application derived automatically from sketches and templates.

Morris and Finkelstein [MF92] are discussing a method for constructing a hypertext in the field of history of art, later generalized as a design process integrating the visual designers in an explicit way [MF96,]. It is not clear in how far their method could be supported by a development environment, because the transformation process of media formally described in the model is not fully operational.

4 The Formal Structure of Multimedia Applications

In the methodologies [ISB95, GPS93, SR95] quoted above the structure of the presentation is derived from the structure of the application's domain or content. This is just one point in a spectrum: the pronounced structuring of a domain induces a formal system structure that can be handled with comparable ease. Figure 1 indicates that there are other situations, classified against number of classes vs. number of objects drawn from each class. Generally speaking, many classes in a system arise whenever the application domain admits a model with a highly perceivable formal structure, many instances of a class are generated when a high degree of similarity has to be captured, see e.g. [Jac92, II.7] or [GHJV95, p. 107], in particular the discussion leading to the `Factory` pattern. Now consider Figure 1. It illustrates typical hypermedia and multimedia applications classified in relation to the number of classes and the number of instances of each class represented by the two axes.

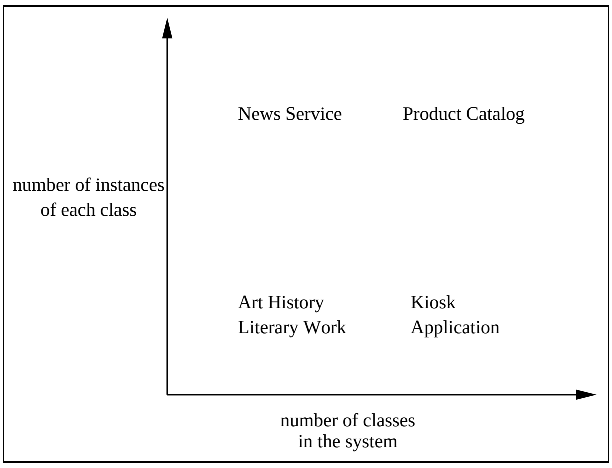

Figure 1: The kinds of hypermedia and multimedia applications

The domain presented in a *product catalog* [SKT+97] is characterized by a large number of product types (classes) and a large number of products for each product type (objects). *Front-ends of databases* are quite similar. The so called *point of information (POI)* as a typical application of *kiosk systems* has to present many different fields (i.e. the classes in this domain) of an enterprise, a city, a railway station, a museum, etc., but may have just one instance for each field or at most a very small number of instances.

On the other hand there are applications with a small number of classes in the domain represented. A *News service* presented by a television channel or a news site presented by an enterprise on the Internet is a type of application that is adequately represented by just one class. This kind of application is characterized by the restriction of constructing only instances of this class.

Our domain in this project is architectural history. This domain has a low formal structure with a small number of classes and a small number of instances, often unique objects. The complexity of the domain is spread out over a few classes and these classes will be very complex. So the usefulness of the mentioned methods for our domain (art history) is low, because these methods base on domain modelling with a supposed regular structure of the domain.

A small number of classes and a small number of instances in the system contains each kind of application presenting artistic work as content, e.g. literary work or art history. These domains, although intrinsically highly structured, present hardly any handle of structuring them formally according to our needs. Thus their presentation in a multimedia application does not depend on the formal content structure (for example, structuring novels into chapters for *Electronic Books* does usually no justice to the artistic quality). A structuring facility for presenting architectural history could e.g. be a presentation form like a *Guided Tour*. Although the construction of such a tour is not the intent of our project, we adopt some of the ideas, as will be seen later.

5 Our Approach to the Development Process of a Multimedia Teaching System

Multimedia applications are large and complex software systems, which should be developed using software engineering methods. Existing authoring systems like Director, Toolbook or Authorware support only the programming phase of the software process, similar to integrated environments like Visual C++, Delphi, etc. They do not support modeling during the analysis and design phase. But modeling should precede implementation in the development of software systems.

The development of a multimedia application is different from the development of software. We want to develop a multimedia presentation (MMP) which turns out to be both a program and a document — this is an important observation for the development that follows. This existence of different views is acknowledged by the parties involved: the art historians perceive the presentation as a document with a course structure and media content. The software engineers see the presentation as a program. By this duality of multimedia presentations as programs and as documents the construction of a MMP necessarily intertwines methods from software construction and from publishing.

Furthermore we had to develop our MMP as a multimedia teaching system (MMTS). For this kind of application the presentation structure depends also heavily on the didactical concept and is in this respect quite independent of the content and its structure. Therefore we need an approach supporting integration between software development, publishing and teaching. It is this approach that will be presented here.

Since markup languages rely on a formal syntax and on the incorporation of narrative text, they are ideally suited for our project once they permit enough flexibility for processing multimedia documents. The class of languages suited to our needs is currently represented by the XML approach [BPSM98], XML instances (or languages) being markup languages similar to SGML instances, HyTime [NKN91] or HTML. These languages are all defined by a document type description (DTD). Hence a particular instance of XML is essentially defined by a DTD. We felt that the conversion of our requirements to a DTD would be the most appropriate way of dealing with the restrictions imposed on us. Documents written in this specification language are converted into an executable program interpreted by an authoring system, the Macromedia Director.

Thus, our approach is divided into two separate parts. The first part deals with the definition of a DTD suitable for our MMTS. The second part makes use of this DTD for writing documents and for building the MMTS itself.

The DTD-Definition Step. This step is subdivided into two sub-steps. First, we establish an analysis phase aiming at the structure, the layout and the capture of the didactical concept of the MMTS. This phase is mostly driven by the art historians consulted by the computer scientists to ensure the technical feasibility of their ideas. The second sub-step formalizes the results of the analysis as an DTD which we call *Altenberger Dom Markup Language* (henceforth abbreviated by ADML; the Altenberg Cathedral is the only church we know of that has a programming language named after itself). This is the work of the computer scientists consulted by the art historians to ensure that the formalization is a valid one. In summary, the first part requires and constitutes an extensive dialog between art historians and computer scientists resulting in layout, structure and didactic concepts as well as their capture in a DTD.

Splitting the Groups. The second part splits the two developer groups. The art historians write documents in ADML, formally instances of the ADML-DTD, create media objects and combine them into scenes. This is the publishing step. The computer scientists realize the technical part of the MMTS and build a converter for the transformation of the ADML documents into an executable multimedia presentation.

The approach to the development process of a multimedia learning system in the Altenberg Cathedral Project is discussed in the following sections and is sketched in Figure 2.

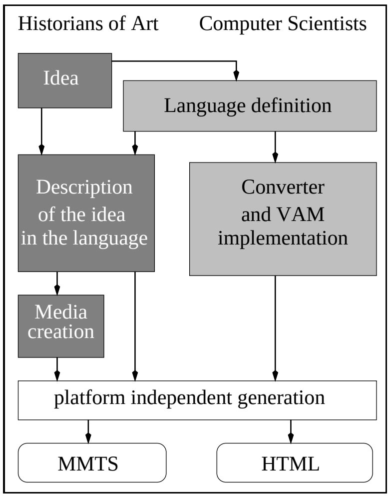

Figure 2: The approach taken in the Altenberg Cathedral Project.

6 Steps towards a DTD-Definition

6.1 The Analysis Step

First the art historians have to establish a suitable presentation for teaching the Gothic architecture. Starting with a vague idea of the layout the logical elements and the relations between them are identified. The description of a sequence of scenarios results in identifying different types of scenes. Scenes are the building blocks for the presentation. They were originally perceived as flat, but turned out later to be structured into *topics* and *subtopics*, thus constituting a hierarchical order. Topics and subtopics combine *media objects*, text with hypertext facility being one of them. On top of this a *navigation* scene serving as a shell yields access to the topics, to a *library* and a *glossary*. This navigation scene is intended to support inductive learning starting from a special visual setting and leading to the general situation explaining architectural aspects. The scene is realized by Quicktime VR [App97]. It centers around a visual representation of the Cathedral's interior and exterior, identifying spots of intense pedagogical interest.

Tackling the problem in this phase of the project consisted in ordering the little chaos in our minds, identifying the important logical elements of the presentation, assembling these elements in a *presentation world* and structuring this world.

6.2 The Model Creation Step

This step maps the presentation structure and elements of the analysis process into a formal representation model using ADML as a specification language.

Each logical element is mapped into linguistic constructs of our emananting language. This applies particularly to the different scene types of the presentation world. Eventually, the set of all these constructs constitutes the core of (the DTD of) ADML. This approach was chosen for two reasons: First, software engineers need formal structures for working in the development process. Second, art historians must work with ADML documents, because they have to construct the presentation at least by providing its content. The language constructed in this manner allows constructing documents at the instance level. The real publishing step is described in the next section.

Summarizing, this phase of the project consists of mapping the art historians' presentation world to an adequate formal world, which in turn is needed for the multimedia development process.

7 Realization and Instantiation Step

The art historians write their educational material, create media objects such as images, videos, animations, audio files, etc., and combine them in ADML documents. Didactical concepts usually cannot be formalized, except for a small part concerning the structure of the presentation. The formalized structure shapes the space for the discourse written by the art historians. Each ADML document is essentially an instance of the structure defined in the DTD and thus a vehicle for implementing the didactical ideas.

There exist in fact some similarities between ADML documents and objects in a program which are instances of classes that are in turn similar to the DTD. In contrast to programming languages, the semantics of ADML instances are not found in the formally defined part but rather are hidden in the discourse written in natural language. Also the class model itself, i.e. the DTD, is not used directly by the authors. As each document is an instance of ADML, the authors always operate directly on these instances of the model. The model defines and provides some kind of frame for their work by helping structuring their thoughts. In all other aspects the models remains passive. Any modifications of the model is to be undertaken by the computer scientists.

This is a substantial difference compared to classical software engineering and displays a prominent example for the *document aspect* of multimedia applications. In classical software engineering instances described in specification parts of program systems are only used and constructed by the end user at run time, but not at development time. Programming may be seen as a refinement of models, thus constituting increasingly detailed levels of modeling. But programming abstracts always from the objects used and deals only with classes (or procedures and functions), never with objects (or procedure calls), which are phenomena of program realization. Considering the other hand the scenario suggested here, such instances of models constructed during development are not modified by the users of the multimedia application. Only few variable data for customizing und storing user profiles are needed; the large amount of information which is presented to the user remains constant. The production of these instance data is integrated into the software development process. Consequently, the classical software engineering methods need to be extended to covering this kind of multimedia applications.

Tool support during the process of instantiation concentrates on the construction of the scenes. While a correct syntax should be guaranteed to ensure the usage in later stages of the software production, the formal bells and whistles itself should not be visible to the user of the tools. This is also different from typical development tools because the formal aspects are important for the technical user thinking about software artifacts. The text orientation of both, ADML and the way the art historians teaches, requires tools more similar to a WYSIWYG text processor than to a bunch of dialog boxes for each element of the ADML.

8 Code Generation

These multimedia applications are built on a very fast changing technical platform. Changes should be anticipated ensuring that the applications remain state-of-the-art. In order to cope with this situation and to ensure maintenability there is a distinct need for decoupling the executable form from the application's content.

Consequently we propose constructing a generic interface to a virtual machine called VAM (Virtual ADML Machine) that is used for rendering. The first implementation of VAM was made in the script language Lingo [Eps98] in Macromedia's Director. The Director gives us the possibility to use both the Mac and the Windows platform without any modifications at source code level. A second, simpler implementation aims at an HTML-Browser like Netscape, thus using HTML as a secondary target platform. The VAM implementations are necessary additions to the ADML since they provide the necessary presentation capabilities. Each VAM implementation is based on the same abstract model of ADML, but each one adds to the static structure of ADML in specific ways. E.g., navigational tools like bookmarks, histories etc. are not part of ADML and thus implemented independently in the VAM.

An ADML converter bridges the gap between the specification in ADML and the VAM implementations. The task of this converter is checking the correctness of the ADML specification and generating VAM initialization code directly usable by the VAM implementations (see Figure 3).

The converter works similar to a compiler by organizing the translation process into an analysis and a synthesis phase, resp. During analysis consistency checks are performed and some properties needed in the synthesis phase are inferred. In the synthesis phase code is generated. Depending on the chosen output format it is either a set of HTML files or initialization and bootstrap scripts in Lingo. The bootstrap scripts are used for an automatic system construction in Lingo. The current turn-around times are ca. 10 min. for a complete system generation process.

We envision that our approach using ADML for content specification, the VAM as abstract presentation and the conversion between them will enable us to perform a simple and smooth transition to other and probably unexpected presentation platforms.

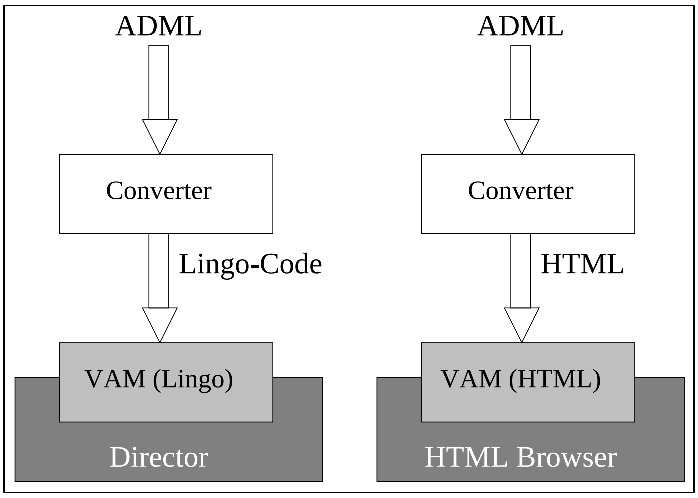

Figure 3: The generation process from ADML to VAM realized in Director's Lingo and in a HTML Browser.

9 Lessons Learned: It Works!

The multimedia presentation of the Altenberg Cathedral is nearly finished. It has been presented sucessfully e.g. to the funding agency that wants to establish and further the use of multimedia in university teaching. Hence we feel to be in a position in which we can relate some of the lessons we learned in this project.

Our first practical insight is that multimedia works for teaching history of art. Exploring the architecture of a building in a multimedial environment and highliting or emphasizing important parts directly and visually provides a new quality for the presentation of the arts. The historians of art see nowadays the multimedia presentation as a viable alternative to classical instruction methods.

Another lesson is that XML works. The separation of structure, content, and the presentation systems works and meets the expectations. The introduction of new media and texts in an ADML document can easily be incorporated into the presentation. The conversion of file formats is done automatically. Thus, the art historians need not bother with the presentation systems, and the computer scientists need not struggle with editorial work in the presentation. But — and this is important — we need more comfortable tools for edititing XML: the historian keep complaining about a clumsy formal syntax. For larger projects it is vital that the authors are concentrating on the content, leaving formalities to those who are concerned. We need WYSIWYG text processors with the capability of syntax oriented editor.

Our third lesson addresses art historians as customers. Their requirements on computer programs are hard to elicitate and even harder to realize. This is due to the lack of (easy to) formalize structures and of course on our side the lacking knowledge of their field. They would appreciate a system managing some kind of *mind maps* and *associative memory* combining literature, works of art and background information in a very flexible and easy to use way. We feel, however, that we did a first step towards realizing these wishes.

10 Future work

We currently construct a concept for a multimedia development environment supporting the multimedia software process model presented here. This is addresses now constructing adequate tool support for art historians in developing using ADML. This includes supporting an analysis and design phase for the development of a didactical presentation concept by art historians, but also supporting a content-authoring phase with ADML. Future work will analyze which fine-grained steps are to be supported in their development work. Finally we will develop a multimedia development environment for formally supporting the processes presented here.

The adequate user-centred support through user-friendly tools is one possible approach in the development of multimedia presentations. Another approach explores the use fuzzy logic [YF94] for the specification of temporal and spatial relationships between multimedia objects. The formal basis of fuzzy logic could be used as the basis for a specification method incorporating natural language to a greater extent.

Acknowledgements. The cooperation with Professor Norbert Nußbaum and Dr. Stefan Hoppe of the Chair of Architectural History has been a pleasure. Their comments are greatly appreciated.

References

[App97] Apple Computer, Inc., Cupertino, California. *Quicktime VR Authoring Studio*, 1997.

[BK99] Susanne Boll and Wolfgang Klas. ZYX-A Semantic Model for Multimedia Documents and Presentations. In *Proceedings of the 8th IFIP Conference on Data Semantics*, January 1999.

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation REC-xml-19980210, World Wide Web Consortium, February 1998.

[DEM+99] Ralph Depke, Gregor Engels, Katharina Mehner, Stefan Sauer, and Annika Wagner. Ein Vorgehensmodell für die Multimedia-Entwicklung mit Autorensystemen. *Informatik-Forschung und Entwicklung*, 14(2):83–94, June 1999.

[Eps98] Bruce A. Epstein. *Lingo in a Nutshell.* O'Reilly, 1998.

[FNN96] S. Fraïsseè, Jocelyne Nanard, and Marc Nanard. Generating hypermedia from specifications by sketching multimedia templates. In *ACM Multimedia 96. The 4th ACM International Multimedia Conference*, pages 353–363, Boston, MA, USA, 1996.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[GPS93] Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM — a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.

[ISB95] Tomås Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, August 1995.

[Jac92] Ivar Jacobson. *Object-Oriented Software Engineering — A Use Case Driven Approach.* ACM Press. Addison-Wesley, 1992.

[MF92] Stephen J. Morris and Anthony C. W. Finkelstein. An experimental hypertext design method and application in the field of art history. *Computers and the History of Art*, 2(Part 2):45–63, 1992.

[MF96] Stephen J. Morris and Anthony C. W. Finkelstein. Integrating design and development in the production of multimedia documents. In Max Mülhäuser and Wolfgang Effelsberg, editors, *MMSD '96: Proceedings of the International Workshop on Multimedia Software Development, March 26/26 1996, Berlin*, pages 98–108, Los Alamitos, CA, March 1996. IEEE Press.

[NKN91] Steven R. Newcomb, Neill A. Kipp, and Victoria T. Newcomb. The hytime hypermedia/time-based document structuring language. *Communications of the ACM*, 34(11):67–83, November 1991.

[SKT+97] J. Schneeberger, N. Koch, A. Turk, R. Lutze, M. Wirsing, H. Fritsche, and P. Closhen. EPK-fix: Software-Engineering und Werkzeuge für elektronische Produktkataloge. In M. Jarke, K. Pasedack, and K. Pohl, editors, *Informatik 97. Informatik als Innovationsmotor*, pages 446–455, Berlin, 1997. Springer.

[SR95] Daniel Schwabe and Gustavo Rossi. The object-oriented hypermedia design model. *Communications of the ACM*, 38(8):45–46, August 1995.

[YF94] Ronald R. Yager and Dimitar P. Filev. *Essentials of Fuzzy Modeling and Control.* John Wiley and Sons, 1994.