

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Geurts, Jozef Petrus Theodorus Maria

A Document Engineering Model and Processing Framework for Multimedia Documents / door
Jozef Petrus Theodorus Maria Geurts. -
Eindhoven: Technische Universiteit Eindhoven, 2010.
Proefschrift. - ISBN 978-90-386-2106-7
NUR 983

Subject headings: document engineering / information presentation / multimedia / hypermedia / Web technology
CR Subject Classification (1998) : H.3.5., H.5.1., H.5.4, I.7.2., I.7.4, I.2.4.



SIKS Dissertation Series No. 2010-03

The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design: Aida Fernández Sánchez (www.summerdesign.es)

Printed by GVO drukkers & vormgevers B.V. | Ponsen & Looijen, Ede, the Netherlands.

Copyright © 2010 by J. Geurts, Eindhoven, the Netherlands.

All rights reserved. No part of this thesis publication may be reproduced, stored in retrieval systems, or transmitted in any form by any means, mechanical, photocopying, recording, or otherwise, without written consent of the author.

A Document Engineering Model and Processing Framework for Multimedia Documents

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de rector magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College voor Promoties
in het openbaar te verdedigen
op woensdag 3 februari 2010 om 16.00 uur

door

Jozef Petrus Theodorus Maria Geurts

geboren te Gouda

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. L. Hardman

Copromotor:

dr. J. van Ossenbruggen

Contents

1	Introduction	11
1.1	Scope	13
1.1.1	Document engineering	14
1.1.2	Knowledge engineering	16
1.1.3	Software engineering	17
1.2	Research questions	18
1.3	Contributions	18
1.4	Outline	18
2	Related Work	21
2.1	Document engineering	21
2.1.1	Historic overview	22
2.1.2	Authoring hypermedia documents	24
2.1.3	Document engineering model	26
2.1.4	Discussion	31
2.2	Knowledge engineering	32
2.2.1	Issues with multimedia annotation	32
2.2.2	Multimedia vocabularies	35
2.2.3	Semantic Web	38
2.2.4	Discussion	41
2.3	Software engineering	43
2.3.1	Software architectures for document engineering	43
2.3.2	Generating multimedia	46
2.3.3	Intelligent multimedia systems on the web	52
2.3.4	Discussion	55
2.4	Summary	55
3	Requirements	57
3.1	Document engineering principles	57
3.1.1	Preliminary requirements	58
3.1.2	Reuse of authoring and design effort	58
3.1.3	Implicit assumption: formatting satisfies constraints of delivery context	59
3.2	Stylesheet vocabulary	60
3.2.1	Representing form conventions	60

3.2.2	Implicit assumption: default style rule adapts form while preserving function	62
3.2.3	Implicit assumption: formatting always succeeds	62
3.3	Structured document vocabulary	63
3.4	Form vocabulary	64
3.4.1	Representing form	65
3.4.2	Form properties to detect constraint violations	67
3.5	Practical requirements	67
3.5.1	Optimize for reuse	67
3.5.2	Web compliant	68
3.6	Conclusion	69
4	Modeling	71
4.1	Modeling the document engineering paradigm	71
4.1.1	Scope of the model	73
4.1.2	Explicit modeling of delivery context	73
4.1.3	Explicit parametrization of the style sheet	74
4.1.4	Explicit modeling of metadata	74
4.2	Modeling the stylesheet	75
4.2.1	Multiple default style rules	76
4.2.2	Detecting constraint violations	77
4.2.3	Selecting alternative style rules	78
4.2.4	Discussion: soft constraints	78
4.3	Modeling the structured document	79
4.3.1	Explicit representation of media items	80
4.3.2	Representing grouping, ordering and priorities	80
4.4	Modeling the document form	81
4.4.1	Three dimensional bounding box	83
4.4.2	Discussion: the containment hierarchy	84
4.5	Summary and Conclusion	85
5	Cuypers document engineering framework	87
5.1	Overview of the Cuypers framework architecture	87
5.1.1	The five steps of the Cuypers transformation chain	89
5.1.2	Embedding the Cuypers chain into a Web server	91
5.1.3	Discussion: The Cuypers versus the traditional transformation chain	92
5.1.4	Summary	93
5.2	Cuypers vocabularies	93
5.2.1	Delivery context	93
5.2.2	Presentation Structures	94
5.2.3	Hypermedia Formatting Objects	98
5.2.4	Style rules	104
5.2.5	Summary	107
5.3	The Cuypers formatter	108
5.3.1	Formatting process	108
5.3.2	Resolving constraints	110
5.3.3	Labeling of constraint variables	112

5.3.4	Summary	113
5.4	Conclusion	113
6	Evaluation scenarios	115
6.1	Method	116
6.2	ScalAR	117
6.2.1	Aggregation	118
6.2.2	Normalization	119
6.2.3	Formatting	121
6.2.4	Serialization	125
6.2.5	Standardization	125
6.2.6	Discussion	126
6.2.7	Conclusion	129
6.3	SEMINF	129
6.3.1	Aggregation	130
6.3.2	Normalization	132
6.3.3	Formatting	133
6.3.4	Serialization	136
6.3.5	Standardization	136
6.3.6	Discussion	136
6.4	DISC	138
6.4.1	Aggregation	139
6.4.2	Normalization	141
6.4.3	Formatting	141
6.4.4	Serialization	141
6.4.5	Standardization	141
6.4.6	Discussion	141
6.5	Performance analysis	144
6.5.1	Automatic adaptation to the delivery context	144
6.5.2	Reuse of style	147
6.5.3	Comparison of the three scenarios	148
6.6	Conclusion	150
7	Conclusion	151
7.1	The research questions revisited	151
7.2	Lessons learned	154
7.3	Discussion and remaining challenges	154
A	Hypermedia Formatting Objects	157
A.1	Style attributes	157
A.2	Delivery context attributes	158
B	Performance Statistics	159
	Summary	169
	References	170

About the author	183
SIKS Dissertation Series	185

Preface

The morning before my fellow student Thijs and myself went for an internship interview to CWI, we went for the same reason to one of the bigger IT consulting firms that was also located in Amsterdam. At the time, the millennium bug and the Internet bubble were thriving well leading to a situation in which two nearly finished IT students were a rare commodity that should be treasured. Hence, a taxi was arranged for us to bring us from the station to our appointment. A high-heeled woman welcomed us with coffee, cakes and an elaborate tour around the premises specifically pointing out the (suspiciously new) recreational facilities. Although at the end of the meeting we did not manage to discuss our project, we were impressed and happy with our newly acquired t-shirts and coffee mugs.

Our second meeting was at CWI, which we reached after a healthy half hour walk through a refreshing Amsterdam drizzle rain. We met with Lynda Hardman and Marcel Worrying who had a vague, but exciting idea of automatically generating multimedia presentations personalized for each individual user. During the 4 months foreseen for our project we managed nothing but scratch the surface of such a system. However, it turned out to be a more than fascinating project going beyond computer science having links to many interesting areas including, web, design, art, cinematography, discourse and artificial intelligence.

It got me hooked and, with pleasure and excitement, I returned to work on it during the summer holiday months. And I kept coming back, first as a part-time research student (combined with my AI studies at the university of Amsterdam) then to fulfill the final Master project and finally it became the topic of my PhD research.

The fun and excitement has always remained and I am grateful for that in the first place to Lynda. She managed to create a quality group with an open minded atmosphere where novel ideas were stimulated and controversial views were welcomed. To me this has always felt as what research should be like. Lynda, on a more personal note, I thank you for your trust, support and patience. No amount of chocolate could satisfactorily express my gratitude, which I am sure you won't understand.

Secondly, I am indebted to Jacco who I appreciate most for his ability to truly follow one's thoughts and recognize and shape a potential idea (provided it is there). At many times when juggling too many dependencies, the "extra hands" proved invaluable and lead the way out. Jacco, I have enjoyed working together a lot and thank you for everything you thought me (that excludes tasting whisky for which I feel I need some additional practice).

Thirdly, I am grateful to all colleagues at INS2 who made the group a versatile and sparkling place to work. In particular, Frank for brightening the day through kind words, or chocolate cakes. Lloyd for patiently answering all my SMIL, HyTime and general Web-related questions. Stefano, Katya and Yulia shared the PhD adventure with me I'd like to thank them for their friendship and support. Stefano, you know what they say about donkeys? I am sure one day

you'll get your priorities straightened out as well. Katya, fighting air conditioning settings, invasion of fluffy animals and the debatable quality of Russian pop songs has been a lot of fun! Yulia, I am still not sure whether you would be qualified best as a delayed owl, or an early early bird. What would you like to be? Raphaël, merci for the much appreciated French support. Thanks to Alia and Michiel for sharing an office and withstanding my bad habits. Thanks to Željko for being a UML modeling master.

During my PhD I have had the privilege to visit two research institutes. The first was in 2002, when I joined the Maenad group of Jane Hunter at DSTC in Brisbane, Australia. Thanks to Suzanne Little and Jane I have had heaps of fun in Brisbane! I still haven't figured out how Tim Tams and Vegemite both can be considered delicious, though. The second visit was in 2004 at the Garage Cinema Research group of Marc Davis. Like the first, this visit turned out to be very valuable as well. Our weekly discussions on just about every topic were very inspiring and thought me a lot.

Over the years, many people contributed to the Cuypers framework. Brian Bailey implemented a constraint based presentation generation system, which could be considered the predecessor of our Cuypers system. Frank Cornelissen implemented (almost overnight) the first version of the Cocoon based framework, which is still in use today. Oscar Rosell made the code comprehensible by reimplementing the formatter using the object-oriented Prolog extension, Logtalk. Furthermore, the Cuypers system relies heavily on a number of open source projects. In particular I'd like to thank Jan Wielemaker (SWI-Prolog), Markus Triska (finite domain CLP library) and Paulo Moura (Logtalk) for their software and support.

I owe gratitude to the members of my doctorate committee, in particular Jane Hunter, Bruno Bachimont and Geert-Jan Houben. Furthermore, I'd like to thank Aida and Sigrid for making the cover of my thesis and Geert and Jan for being my paranimfen. The Rijksmuseum in Amsterdam kindly permitted the use of digital representations of their artworks for the Cuypers demonstrators and the cover of this thesis. The research reported in this thesis was partly funded by the NASH project (NWO projectnumber: 612.060.112).

Finally, I expect it will come to no surprise that writing this thesis was not only intellectually challenging, but at times also emotionally demanding. For the later I owe gratitude to family and friends whose patience, understanding and support throughout the years has been of great help. Laurence, thanks for putting back the smile back on my face whenever it had gone.

Chapter 1

Introduction

People often associate the term *multimedia* with lively presentations combining film, animation, images and music. It is typically considered entertaining, interactive and great for playing games. It is also used to indicate advanced technology. Sometimes it is art. Although all of these are valid qualifications, multimedia is foremost a very confusing term. For example, film is considered multimedia, but technically so is a newspaper article with a picture, or even your neighbor showing his holiday photographs.

In this thesis we consider multimedia in the context of an electronic document that attempts to convey a certain message to a reader. Because of the ambiguity of the term multimedia we define it as a document that has the following properties:

Heterogeneous media types Unlike text-based documents, a multimedia document does not have a dominant media type but is composed of multiple media items using different media types, such as, image, text, audio and video. The author of a multimedia document uses media items that are, either specifically created, or (re)used from existing resources, to represent the message she intends to convey.

Spatio-Temporal dimensions A multimedia document has, besides two spatial dimensions, a temporal dimension. Consequently, the author of a multimedia document should, in addition to the spatial layout, synchronize media items in a meaningful way.

Figure 1.1 presents three screenshots of multimedia documents that were designed for various screen sizes¹. The first two documents (A and B) are about the painting technique “Chiaroscuro” in the work of “Rembrandt”. Both presentations contain a text explaining the term *chiaroscuro*, which is accompanied by a synthesized voice-over reading the text. The image of Rembrandt is the first of eight examples, presented in sequence, illustrating his use of *chiaroscuro*. The third screenshot (C) is about “Genre paintings” in the work of “Johannes Vermeer”. Similarly, to A and B, a text explaining the term “Genre” painting is accompanied by a sequence of illustrative examples.

Authoring such multimedia documents is in multiple ways different from authoring a text-based electronic document. First, modern text processors allow an author to abstract from type-

¹A copy of these documents can be found at
<http://www.cwi.nl/~media/cuypers/generated/>

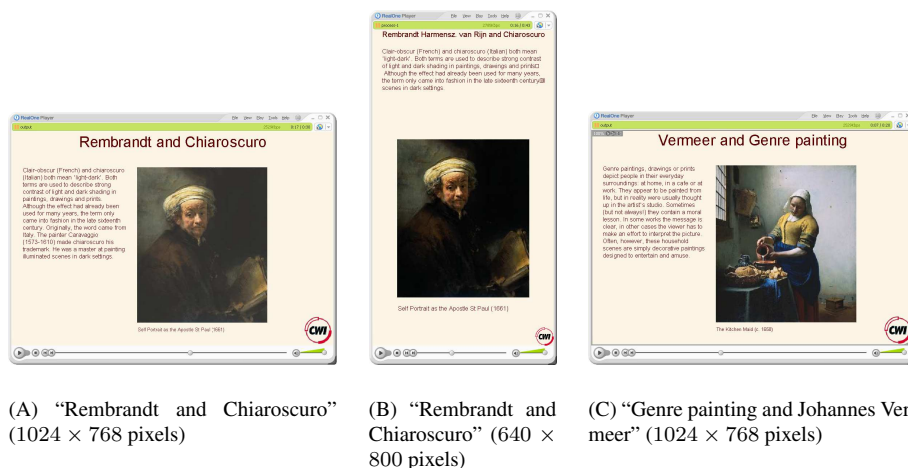


Figure 1.1: Three examples of multimedia documents.

setting details, such as hyphenation, kerning and leading². The word processor automatically formats the text in such a way that it fits within the designated area, such as a page or screen. In contrast, the author of a multimedia document carefully designs a multimedia document so that it exactly fits the screen size the document is designed for. For example, presentation ‘A’, shown in figure 1.1, was created for a screen with a width of 1024 pixels and a height of 768 pixels, whereas presentation ‘B’, which conveys an identical message, is specifically created for a screen with a width of 640 pixels and a height of 800 pixels.

Secondly, modern text processors often have the ability to include predefined styles (e.g. corporate identity), which allows an author to abstract from the styling of the document. Consequently, an author does not require design expertise to ensure a consistently formatted and aesthetically pleasing document. In contrast, modern authoring tools for multimedia documents require an author to make both authoring and design decisions. For example, the three presentations in figure 1.1 have a common style. However, since the function of each media item is implicit for the authoring tool, an author should design all three presentations individually.

The reason that authoring and design are intertwined in the production of multimedia documents is that the spatial layout and temporal synchronization between media items is semantically significant. Unlike text, where a sentence or word may be split to continue on the next line or page, breaking the spatio-temporal relations between media items in a multimedia document typically alters the message conveyed by the document. For example, the text explaining chiaroscuro in figure 1.1 is top aligned and placed directly next to a painting using chiaroscuro. The author did this intentionally to indicate a relationship between the two. When the presentation does not fit the screen, the author carefully redesigns the presentation in order to maintain these relationships. A possible alternative would be to first present the text, since it explains the concept of “Chiaroscuro”, and then present the example paintings. Consequently, the author of a

²Typesetting, hyphenation, kerning and leading are typographic terms that originate from the manual work of compositors who used metal casts of letters to construct an image of a page used in a mechanical press.

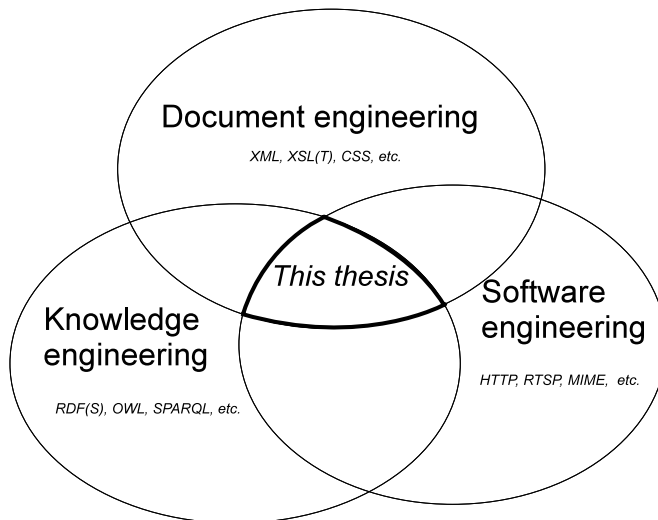


Figure 1.2: Venn diagram of disciplines and associated technologies related to our research.

multimedia document should understand the impact the presentation may have on its semantics.

Although a multimedia document can be adapted to a particular context, and multiple multimedia documents can be consistently styled, this typically requires significant human investment. The costs involved in authoring and designing multimedia documents are therefore relatively high compared to textual documents. As a result, the production of multimedia documents is only viable in specific cases, which is unfortunate because multimedia documents are typically effective to convey a particular message.

This thesis reports on our research aiming to reduce the effort involved in the authoring and design of multimedia documents by automating part of the production process. In the subsequent sections we elaborate on the scope of this work, we specify the investigated research questions and summarize the main contribution. Finally, we present a brief outline of the thesis.

1.1 Scope

Research involving multimedia is inherently cross-disciplinary, having relations to various established research domains. Figure 1.2 illustrates three areas in computer science relevant for our research. Foremost, *document engineering*, which investigates systems for creating, managing and maintaining electronic documents. Secondly, *knowledge engineering*, which studies acquisition and formal representation of knowledge. Thirdly, *software engineering*, which studies the creation and maintenance of software architectures. In the next sections we elaborate on the relation between these disciplines and our work.

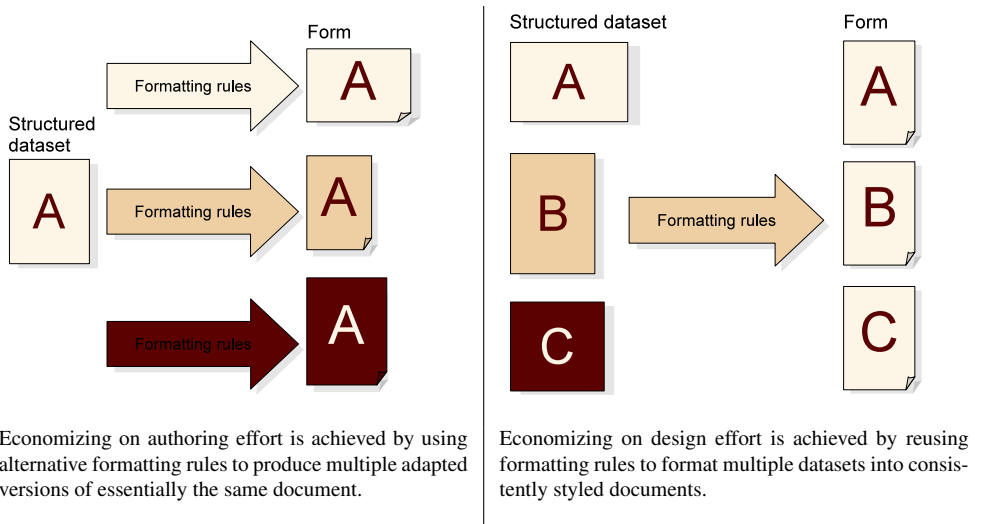


Figure 1.3: The document engineering paradigm

1.1.1 Document engineering

For many ages the term “document” referred to the physical object that carried a representation of the message the author wanted to convey to the reader. This could be a piece of paper, a papyrus scroll or even a clay tablet. Electronic documents are different as they do not have an inherent physical form. Only when interpreted and rendered by appropriate software does the document become perceivable on, for example, a computer screen or a print-out. Roger T. Pédaque [119]³ defines an *electronic document* as a dataset organized in a stable structure associated with formatting rules to allow it to be read both by its designer and readers.

Because rendering is essential to perceive an electronic document, the perceivable form of an electronic document may be adapted by adjusting the rendering process. This notion of separating the dataset from its presentation is known in literature as *the document engineering paradigm*, also known as *the multiple delivery publishing model* [30, 61, 139], or *separation of content and style* [28, 154]. It is used in most modern word processors, including L^AT_EX [95], Microsoft Word [107] and OpenOffice.org Writer [117]. In the following sections we elaborate on the document engineering paradigm and the reduction of production costs it achieves.

Economizing authoring effort

The document engineering paradigm allows a document to be authored once and automatically adapt it to a specific presentation environment. The left hand side of figure 1.3 illustrates adaptation of the form of an electronic document by using multiple sets of formatting rules to present a single dataset. Effectively, the author of document “A” automatically generates multiple different

³“Roger T. Pédaque” is a pseudonym used by a multi disciplinary group of French researchers.

versions of the form of the same dataset. Some practical applications of automatic adaptation include:

Adaptation to size Adaptation is used to ensure that the presentation of a document meets the characteristics of a specific device or medium. For example, the layout of an HTML page is optimized to fit the size of the browser window. If the browser window is resized, the layout automatically adapts to fit the new size.

Adaptation to the user Adaptation may be used to meet requirements and preferences of an individual user. For example, a user with a visual impairment may use a set of formatting rules that ensure the presentation of the document is rendered using large fonts. In addition, adaptation can be used to adapt the dataset for a particular presentation context. For example, a teacher authors a test, for which she includes both questions and answers. Using this dataset she creates two versions, one for during the exam, excluding the answers, and one for after the exam, including them.

Export to different file format Adaptation of electronic documents may be used for generating multiple file formats for essentially the same document. For example, an author creates a dataset once and uses different sets of formatting rules to generate a PDF version, a PostScript version and an HTML version.

Economizing design effort

The document engineering paradigm allows an author to reuse styling of a document. The right hand side of figure 1.3 shows that the same set of formatting rules can be used for multiple datasets. Effectively, a designer creates a set of transformation rules once. The authors of documents “A”, “B” and “C” all use this set of formatting rules. Some practical application of reusing formatting rules include:

Professional design For effectively conveying the intended message of an author, a document should be clearly readable and attractive to perceive. Design is therefore an important aspect of a document, which is illustrated by companies hiring professional designers for their publications. However, in many cases, the author of a document is also the designer (since a professional designer is often not available). Reuse of formatting rules allows a professional designer to formulate a set of transformation rules, which can be applied to multiple documents, effectively allowing an author to focus on the intellectual content of the document while using the professional expertise of a designer for the presentation.

Consistency Reuse of formatting rules proves to be effective in scenarios where multiple authors work on parts of the same document. For example, the editor of a collection of scientific publications, provides all (independent) authors with the same set of formatting rules. Since all documents will use the same consistent style multiple contributions can relatively easily be merged into a single document.

Dynamic content Due to the separation of dataset and formatting rules, electronic documents are not as static as traditional documents. Examples include constantly updated websites presenting news, stock information and weather conditions. The reuse of formatting rules allows automatic updating of the dataset, without the need for an author to redesign the presentation of the document.

1.1.2 Knowledge engineering

The author (and designer) of a multimedia document use different types of knowledge to represent the message she intends to convey. In order to automatically adapt a document to a specific context, while preserving the intended message, part of this knowledge should be made explicit. In the following sections we elaborate on the specific types of knowledge used during the production of a multimedia document.

Domain knowledge

The author of a multimedia document either creates or selects existing media items to convey the message she intends to convey. For example, in figure 1.1 the topic of the document is Rembrandt and his use of Chiaroscuro. The media items representing paintings are specifically selected (or created) because they are made by Rembrandt and illustrate the use of the technique. However, the same media item may be used in a different context, conveying a different message. For example, the portrait of Rembrandt could just as well be used in a presentation about the apostle Paul (in this painting Rembrandt portrays himself as the apostle Paul). Similarly, the explanatory text about chiaroscuro can be used in a context independently of Rembrandt. Consequently, the context in which a media item is used influences its conveyed message. Therefore, an author should understand the domain represented by a media item in context of the multimedia document.

Discourse knowledge

The author of a multimedia document structures the media items to represent a particular message. This is typically referred to as *discourse*. The document portrayed in Figure 1.1 attempts to show the use of chiaroscuro in the work of Rembrandt. The author deliberately presents the text explaining chiaroscuro before the example paintings. This way, a reader views the image while being aware of the concept chiaroscuro and (hopefully) recognizes its use in the painting. If the example paintings were presented before the explanatory text, a reader would have more difficulties recognizing the link⁴.

The media items in a multimedia document can participate in multiple discourse relationships, which are conveyed through the layout and style. These can be one-to-one relationships, such as the caption with the painting in figure 1.1, or a one-to-many relationship, such as the sequence of example paintings and the explanatory text in figure 1.1. When one of these relationships is broken, for example, because of limited screen space, an author carefully redesigns (parts of) the presentation to make sure the intended discourse relationships are conveyed correctly. As a result, the author of a multimedia document needs to understand the different types of relationships between media items and how they can be communicated using layout and style.

Design knowledge

For successful communication, the author and reader need to share a system of design conventions, which allows them to interpret the message expressed by the layout and style. In figure 1.1, the author uses a large bold font for the text, centered at the top of the document, to indicate the

⁴We do not propose this particular example as an educational strategy, but it serves as an illustration of the type of knowledge an author uses to structure her message.

title. Typically, the reader of the document interprets this as a title and understands that it represents the topic of the document. If the same text had been presented using a small font and a different position a reader probably would have difficulties recognizing it as a title.

Besides the use of layout and style to convey relationships between media items, a designer is also concerned with the constraints of the device used for presenting (e.g. the size of the screen, whether the device can show colors and the available bandwidth). Finally, a designer is concerned with the aesthetics of the multimedia document, which includes clearly visible fonts, sufficiently large images and an overall aesthetically pleasing design that is appropriate for the conveyed message. Typically, these aspects of design are not unrelated. For example, a large font may increase clarity but also requires more spatial resources. A designer of multimedia documents needs to understand the implications of design and invariably needs to make trade-offs.

1.1.3 Software engineering

The World Wide Web [22] (web) is a software framework which contains billions of electronic documents (i.e. resources). Since the web has many different users and many different presentation contexts, adaptivity is particularly relevant in a web context. Document engineering technology, such as HTML, CSS, XML and XSL, are actively used on the web and are partly responsible for its success. In addition, the open architecture of the web stimulates reuse of resources, which includes media items but also the reuse of knowledge and software components. In the next sections we elaborate on the advantages of using web technology and the requirements that are imposed by the web architecture.

The web as a resource

The web architecture strives for uniformity between applications. This, on the one hand, allows reuse of resources, such as media items, which can be included in a document by referring to their web addresses (URLs). On the other hand, uniformity allows reuse of software components, which may result in sophisticated applications with relatively little investment. For example, most professional websites use complex modularized technology, such as content management systems that dynamically generate the requested page from external resources.

The web as a delivery platform

The web is based on a client-server architecture where the server is typically unaware of the client, and the client is *a priori* unaware of the presented document. This independence is key for the scalability of the web. Hence, an author writes a document once, which can potentially be accessed by billions of clients. Moreover, a client can potentially access billions of documents using the same browser software.

However, the uniformity of the web architecture also imposes constraints on the software architecture of a web application, which limit the possible functionality. For example, communication on the web is *stateless*. This means that client and server do not have access to the history of their transactions. If history is relevant, such as for applications using interactive forms (e.g. on-line shops), the history needs to be communicated with each transaction.

1.2 Research questions

This research was originally motivated by failed attempts to apply document engineering techniques when dealing with multimedia documents (e.g. SRM-IMMPS [27], HyTime [84]). Hence, the benefits of reduced authoring and design costs do not apply to multimedia documents. This discrepancy is the topic of this thesis. More precisely we research the following questions:

Research question 1 (REQUIREMENTS). *What are the requirements for an extended document engineering model and processing framework that include support for multimedia documents?*

Research question 2 (MODEL). *What are the properties of such an extended document engineering model?*

Research question 3 (FRAMEWORK). *What are the properties of a software architecture that implements the formatter of the extended document engineering model, and fulfills the requirements imposed by a web architecture?*

1.3 Contributions

The research described in this thesis led to the following tangible results:

Contribution a (EXTENDED DOCUMENT ENGINEERING MODEL). A document engineering model for multimedia documents, which economizes on authoring and design effort by separating the presentation of a document from the intended message of the author, and makes the relevant dependencies and trade-offs within and between them explicit.

Contribution b (HYPERMEDIA FORMATTING OBJECTS). A vocabulary for hypermedia formatting objects, analogous to formatting objects for textual documents, which describes the presentation of multimedia documents.

Contribution c (MULTIMEDIA FORMATTER). A multimedia document engineering formatter based on the EXTENDED DOCUMENT ENGINEERING MODEL (Contribution a) and the HYPERMEDIA FORMATTING OBJECTS (Contribution b) vocabulary. This formatter can be used to experiment with dependencies and trade-offs in document engineering for multimedia documents.

Contribution d (MULTIMEDIA DOCUMENT ENGINEERING FRAMEWORK). An open and reusable software framework, which is used to automatically generate multimedia documents using the MULTIMEDIA FORMATTER (Contribution c). It assumes that multimedia assets are annotated and combines a stateless web architecture with a modularized knowledge architecture, using existing web and semantic web technology.

1.4 Outline

This chapter informally introduces the questions researched in this thesis. The remaining part of this thesis is structured as follows:

- Chapter 2** gives an overview of relevant technology and elaborates on the current state of the art concerning document engineering, knowledge engineering and software engineering. We will show that currently available document engineering technology is insufficient for multimedia documents.
- Chapter 3** identifies implicit assumptions in the traditional document engineering model. Because multimedia documents do not fulfill these assumptions the traditional document engineering model does not apply. Based on the discrepancy in the traditional model we derive requirements for an extended document engineering model that includes multimedia documents. Additionally, we state requirements on the software architecture, which ensures the model is practically implementable using currently available technology.
- Chapter 4** describes the extended document engineering model that fulfills the requirements related to the model (contribution a). We use the model to clarify trade-offs that are less apparent in the traditional model, but are inherent to multimedia document engineering. Furthermore, the model serves as a reference for the implementation of a document engineering framework and formatter that includes multimedia documents.
- Chapter 5** describes our document engineering framework called Cuypers⁵ that implements the extended document engineering model while satisfying the architecture requirements imposed by the web architecture (contributions b, c and d).
- Chapter 6** illustrates and evaluates our model and the Cuypers framework by describing the implementation of three document engineering scenarios.
- Chapter 7** summarizes and discuss the results described in this thesis.

⁵Named after the Dutch architect Pierre Cuypers (1827-1921) whose designs include the Rijksmuseum in Amsterdam.

Chapter 2

Related Work

In chapter 1 we introduced the high-level concepts of document engineering, knowledge engineering and software engineering in relation to multimedia documents. In this chapter we elaborate on these areas and discuss related work relevant for our research.

The first section describes the basic principles of document engineering and ends with the current state of affairs for document engineering for multimedia documents.

The second section elaborates on knowledge engineering and describes issues with describing multimedia data. Furthermore, it introduces the semantic web, which is a framework for describing and reusing knowledge on the web.

The third section introduces software architectures for document engineering systems. Furthermore, it describes related systems and architectures comparable to our own.

The fourth section summarizes the main observations, which we use to derive requirements for an extended document engineering model, the topic of the next chapter.

2.1 Document engineering

The difference between authoring textual documents and multimedia documents has not always been so apparent. Before the age of electronic documents, authoring a textual document, in a way, was comparable to authoring a multimedia document today. For printing, a professional typesetter was needed to place metal or wood casts of glyphs (i.e. letters) on a press for each individual page. Although many copies could be printed this way, the typesetter needed to redo the formatting from scratch for a different paper size. After the introduction of the computer for printing documents, the glyph casts were replaced by control codes to instruct a machine to print the glyph. Although printing became a little less time consuming, each document still needed to be prepared for a specific paper size. And, to make things worse, the control codes used by the printing machines typically were different between brands and even types. Consequently, electronic documents could not be distributed in electronic form, unless the sender was sure the receiver had access to exactly the same brand and type of printing machine for which the electronic document was prepared.

This section introduces related work in the field of document engineering. It first gives historical background and describes the state of the art. Based on this, we then describe the

concepts of the document engineering model that are relevant for this thesis. Finally, we discuss related work in the area of document engineering for multimedia documents.

2.1.1 Historic overview

In the late sixties electronic documents typically contained hardware specific control codes which formatted a document in a certain way for particular hardware. These codes were often subject to change when different machinery was used to render the document. William Tunnicliffe, chairman of the Graphic Communications Association (GCA), gave a presentation in 1967 about the separation of the intellectual content of documents from their presentation. This is what many consider the start of the document engineering paradigm [69] and the document engineering discipline.

Separating content from presentation (SGML, DSSSL)

In the late sixties, Charles Goldfarb and others at IBM defined the Generalized Markup Language [69] (GML). GML described a document but abstracted over the specific control codes used by a printing machine. In this way, an electronic document could be sent to a receiver who used transformation software to produce a document that was processable by the receiver's hardware. Later, in 1986, this work formed the basis of the ISO standard SGML [83] which was the predecessor of currently well known document formats such as HTML [151] and XML [29].

SGML specifies the structure of a document and, following document engineering principles, abstracts from presentation details. A marked-up document, however, still needs to be formatted before it can be presented to a reader. DSSSL [85] was standardized as the language for specifying stylesheets for SGML documents (10 years after SGML). In addition to standardizing the language which describes the transformation, it also introduced the concept of formatting objects.

Typesetting vocabulary (T_EX, XSL-FO)

Formatting objects are device independent objects that describe the form of the document exactly, but abstract over the rendering (i.e. the presentation of the document on a specific medium/device), which is realized by a typesetter, such as T_EX [91] and Extensible Stylesheet Language Formatting Objects (XSL-FO) [154].

T_EX is a typesetting system developed by Donald Knuth specifically designed for technical texts containing formulae. It was one of the first systems which allowed authors to produce high quality documents comparable to the quality achieved by manual typesetters dealing with typography issues such as typefaces, hyphenation, kerning, leading and the placing of figures. Note that these algorithms are specified independent of a specific document. Furthermore, note that the automatic formatting of a textual document is a complex process which, in addition to ten years of work by Knuth, resulted in several PhD theses (Frank Liang, Michael Plass, John Hobby) [92].

Similar to T_EX, XSL formatting objects (XSL-FO) [154] is a vocabulary, to represent the form of a text-based document. (XSL-FO is part of Extensible Stylesheet Language (XSL), which we describe in the following section.) In general, the vocabulary used to represent the document form abstracts from a proprietary file format. For example, T_EX generates a DVI file [60], which can be relatively straightforwardly transformed to document formats such as PostScript [2] or PDF [1]. Note that in order to support multiple document formats with different

features the formatting vocabulary is needs to be sufficient expressive to represent the combined features of the proprietary formats.

Abstract from form (\LaTeX , HTML and XML)

One of the motivations of Knuth to create \TeX was to allow an author to denote formulae in an intuitive way (in contrast to writing formulae using a typewriter). Although \TeX was quite successful in that respect, the content and style were mixed. Consequently, a document designed for a particular paper size could not easily be adapted to other formats. These were essentially the same problems document engineering tried to solve by separating style from content. \LaTeX , HTML and XML were designed to address these problems.

\LaTeX [95] is based on the philosophy that an author should focus on the intellectual content of the document and should not be bothered by formatting details. It was originally written in 1984 by Leslie Lamport as an extension to the typesetting system \TeX (\LaTeX is written in the macro language of \TeX). \LaTeX provides an author with additional constructs for automating common tasks in document authoring such as creation of enumerated lists, indexes, cross-references, bibliographies etc. Note that, if desired, an author can still express explicit formatting by using \TeX commands within a \LaTeX document.

\LaTeX was specifically designed for technical documents. In contrast, the HyperText Markup Language (HTML) [122] was designed to describe documents accessible on the web. It was defined by Tim Berners-Lee to create hypertext documents that are portable across different platforms. Together with Cascading Style Sheets (CSS) [28], which describes rules to associate styling information to an HTML marked-up document, it implements the document engineering paradigm for the web.

Although HTML positively influenced the popularity of the web, it was not suited for describing information not intended for human consumption (e.g. database records). The eXtensible Markup Language (XML) [29] is a subset of SGML specially designed for the web. Although XML was developed after the first versions of HTML, it is considered the foundation of the web [20]. XML is used to serialize various languages including RDF(S) [150, 156], SVG [56] and SMIL [149]. Due to the adoption of XML as a data representation language, generic tools can be used that simplify the development process and facilitate the interchange of data between different programs.

The Extensible Stylesheet Language (XSL) [154] is the stylesheet language for XML documents (comparable to DSSSL, the stylesheet language for SGML). It consists of two parts: 1) XSL Transformations (XSLT) [35] which specify the transformation between XML documents and 2) XSL-FO which defines a formatting vocabulary to represent the presentation of the document. In contrast to CSS, which complements an input structure with formatting specification, XSL allows an author to adapt the input structure during the transformation.

Document engineering for multimedia documents (HyTime)

Document engineering technology is currently actively used for the processing of textual documents. In contrast, for authoring multimedia documents, similar levels of abstraction do not exist. HyTime was an attempt to provide similar functionality for documents based on temporal flow. HyTime [84] allows an author to define relationships between objects by hypertext linking, scheduling and alignment.

HyTime, like SGML, is not a presentation format, but a meta language to describe a class of documents. Subsequently, HyTime needed transformation software for producing the final

presentation. The level of abstraction in HyTime documents turned out to be too high. Specific transformation software, designed for a specific class of documents, was required for producing the final form document, making HyTime practically unusable [32] for general use.

Despite its failure to achieve the goal it was designed for, HyTime provided many valuable insights that influenced current technology (e.g. XLink [43], XPath [36], XPointer [44]).

HyTime was designed as a meta language for describing multimedia documents. In contrast, the Amsterdam Hypermedia model, and its derivative SMIL, focuses on a presentation format for multimedia documents.

2.1.2 Authoring hypermedia documents

In the early nineties of the previous century, technical advances in hardware and bandwidth increased the production of multimedia data. Although individual media items could be transmitted successfully, no document format suitable for the web allowed the specification of multimedia documents. Since existing document models and authoring paradigms were not sufficient for multimedia documents on the web, Hardman developed the the Amsterdam Hypermedia Model (AHM) [76].

The model we propose in chapter 4 for describing the form of a multimedia document is based on the AHM. In addition, the SMIL output format generated by our Cuypers engine, described in chapter 5, is based on the AHM.

Amsterdam Hypermedia Model

The AHM is based on the Dexter hypertext reference model [74] and describes a model for authoring time-based hypermedia documents and defines four components:

Atomic An atomic component refers to the media content used in a hypermedia presentation. Atomic components have an explicit duration and extent, style attributes relevant to the media item and a reference to a channel which denotes the spatial position of the media item.

Channel A channel is a conduit which defines a spatial region in the hypermedia presentation. It is referenced by an atomic components and used to position the media item spatially. A channel is associated with a media type (text, image, video audio etc.) has style attributes and can be (re)used by more then one atomic component.

Composite A composite component is used to describe the spatio-temporal structure of the presentation. A composite can be temporal (in which case they have a duration) or atemporal. It can have style attributes which are inherited by descendant components.

Link A link component is used to define relationships between components in the presentation. This includes the specification of an anchor, a target resource and specifiers that describe the behavior of the (activated) link, such as whether the link should be opened withing the current presentation or in an external application.

The AHM expresses a balance for multimedia documents that on the one hand allows an author to abstract over presentation details that do not significantly influence the presentation. On the other hand, it is sufficiently expressive for an application to interpret the specification and render the final form presentation corresponding to the intention of the author. This is comparable

with the textual formatting model described previously. Note however that the AHM is optimized for manual authoring.

SMIL

The SMIL [149] specification for multimedia documents on the web was influenced by the AHM. Although efforts were made to make the language readable and convenient for manual authoring, it is commonly accepted that for authoring multimedia documents an authoring tool is highly advisable. GRiNS [118] (previously known as CMIFed [145]) is an authoring tool for multimedia documents. It is, like SMIL, based on the AHM and shows multiple views of a multimedia document, including a view of the hierarchical structure of the document, a view of the spatial layout and a view of the temporal synchronization.

SMIL was designed with some support for adaptation. In SMIL this is expressed by means of the `SWITCH` element. An author of a SMIL document uses the `SWITCH` element to specify alternative presentation for part of the document. When the document is played and reaches the `SWITCH` element, the player selects one of the alternatives based on a specified parameter retrieved from the delivery context. For example, a presentation specifies a video with synchronized subtitles in several languages. Based on the value for `system-language`, which is defined by the delivery context, the player selects the subtitle in an appropriate language. Other values for adaptation include `system-bitrate` and `system-screen-size`.

Although SMIL has some support for adaptation, it still has some problems. For example, the style of a SMIL presentation is embedded in the document and can not be applied to other multimedia documents. Furthermore, adaptation is based on document instances, which consequently means that an author needs to specify all possible adaptations within the SMIL document even if they are not used in a particular delivery context.

Roisin [24] proposes a language for formatting multimedia documents which in case of a constraint failure gives hints to the formatter how to choose the most appropriate resolution strategy. These hints include priorities on media items which can cause less relevant media items to be omitted and fall-back rules in case a particular formatting strategy fails. Disadvantage of this approach is that the server, just like for adaptation in SMIL, needs to provide resolution strategies for a device it possibly does not know.

Scalable MSTI (Media, Spatial, Temporal and Interactive) is a multi-device authoring approach that, based on a minimal base document allows an author to specify a range of extensions exploiting specific characteristics of a specific device [120]. The advantage of this approach is that a document is optimally adapted to a specific device. However, the authoring of document is a rather complex task for which dedicated tools are necessary.

Flash

Flash [100] is a proprietary format to author and present multimedia documents on the web. One of the success features of Flash is its support for different media types, including bitmap images, vector images, text, audio and video, which are all integrated in the format. Provided that the client-side supports the Flash format, this guarantees that the document form can be presented as intended by the author. Another success feature of Flash is its sophisticated scripting facilities, which allows development of rich Internet applications.

However, Flash (implicitly) states minimal requirements on the delivery context necessary to implement the Flash player. If these requirements are not met and, subsequently, there is no Flash player available, the document cannot be presented. Consequently, Flash does not allow an

author to abstract from the properties of the delivery context, comparable to text-based document engineering technology. Since it does not separate authoring effort from design effort, it cannot be considered a viable implementation of the document engineering paradigm for multimedia documents.

Exhibit

Exhibit is a publishing framework designed for the visualization and management of structured data collections [82]. This is of particular interest to maintainers of dynamic data repositories, such as digital libraries, as it allows them to abstract from the visualization of the data and instead reuse existing widget that are optimized to convey particular types of information. For example, by indicating the temporal properties of a data resource, the *Timeline* widget automatically visualizes the resources on an interactive timeline. Similar, generic visualizations widgets are available for geographic and numerical data. In addition, Exhibit allows a user to interactively sort and filter the data that is associated with one or more visualization widgets. As a result a user can focus on a particular point of interest and reveal hidden relationships in the data collection.

Arguably, the focus in Exhibit is on visualizing data collections in an objective manner. However, if an author wishes to exploit certain specific domain semantics, for which no widget exists yet, a dedicated widget should be implemented. To justify the additional development effort, the data collection having these specific domain semantics should be sufficiently large. Because the development effort of a widget is relatively high, the threshold to develop domain specific widgets is typically low. This compares to the trade-off in document engineering between, the class of documents that can be transformed by a stylesheet, and the specificity of form conventions represented in the stylesheet(see section 2.1.3).

2.1.3 Document engineering model

When we read a document, often without realizing, we immediately recognize part of the structure of the document by the way it is presented. This includes the division in chapters, sections and paragraphs, which we recognize by the distinctively formatted headers. These form conventions are exploited by document engineering. A *form convention* is a shared understanding between an independent author and reader about how a particular function can be communicated using a particular form.

Form conventions can be shared between a large group of people, such as the use of a large bold font for a chapter title that almost everybody would recognize. There are, however, also form conventions that apply to more specific domains. H_2O is a symbolic convention in chemistry that denotes water and 11-10-1975 is used to indicate a date (11th of October in Europe, November 10th in Northern America). These distinctively formatted form conventions are immediately recognizable to a reader familiar with the domain.

In addition to existing conventions, an author can introduce conventions that apply within a single document. For example, in chapter 3 of this thesis, we introduce a form convention for presenting requirements. Since all requirements are formatted in a consistent way a reader recognizes a requirement by the formatting.

The document engineering paradigm abstracts from the perceivable form of a document by making the function and corresponding form of form conventions explicit.

The *function* of an electronic document represents the message an author intends to convey to a reader. The *form* of a document represents the perceivable part of an electronic document

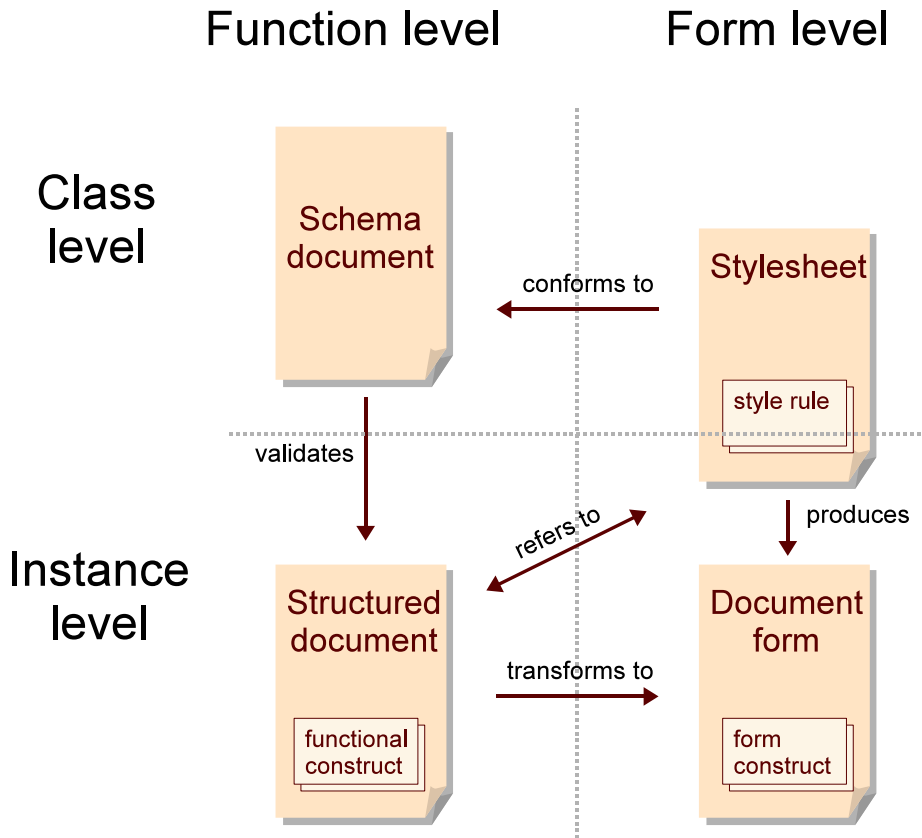


Figure 2.1: Conceptual abstractions relevant for the document engineering paradigm.

that attempts to convey the function to reader¹.

Figure 2.1 illustrates the four conceptual abstractions relevant for the document engineering model. On the left hand side *function level* denotes the level of abstraction that is concerned with the message an author intends to convey. On the right hand side, *form level* represents the abstraction level concerned with representation of the form of the document. At the top, *class level* denotes the level of abstraction that is concerned with the modeling of an abstract document that defines common characteristics of multiple documents. At the bottom, *instance level* represents the abstraction level that is concerned with a specific instance of a document.

The *structured document*, in the bottom-left quadrant, is a representation of a specific document function. A structured document contains media items (or references to media items), such

¹Semiotics is a theory of signs and symbols. It makes a distinction between *signifier* and *signified* that compares to the distinction between form and function. In terms of semiotics, the document as a whole is defined as a *sign*. The document form is the *signifier* and the document function the *signified* [129].

as text and figures, that are explicitly structured. In document engineering technology, such as L^AT_EX [95] and HTML [123], this represents the source document as specified by the author.

The *schema document*, in the top-left quadrant, abstracts over specific document instances and allows an author to specify constraints on the structure for a class of structured documents. A document structure can be validated against the schema document to verify that it conforms to the specifications. A validation helps to ensure the document is well structured, however, in most cases the schema document is implicit.

The *stylesheet*, in the top-right quadrant, specifies the transformation from a structured document to its perceivable document form. A stylesheet is defined independently of a structured document, and is responsible for the advantages of document engineering: automatic adaptation and reuse of style (see chapter 1). In the remainder of this thesis we refer to the *designer* of the stylesheet to denote the creator of the stylesheet who makes decisions concerning the document form. We refer to the *author* of the structured document to denote the creator of the structured document who makes decisions concerning the document function. Note that in practice, however, there is often not such a clear cut separation between the tasks of an author and the task of a designer. Finally, the *reader* of the document form denotes the perceiver, who interprets the document form.

A stylesheet applies to a class of documents. The size of the class is constrained by the specificity of the encoded form conventions. If a stylesheet encodes generic form conventions, such as title, the stylesheet applies to a large class of documents. If however the stylesheet encodes specific domain dependent form conventions, the class of documents becomes smaller. Subsequently, the designer of a stylesheet needs to find a balance between the document function that is explicitly represented in a structured document and the document function that remains implicit.

Although a stylesheet defines the formatting of a class of documents, an author might want to define exceptions that apply to a specific document. Subsequently, a stylesheet can specify formatting on both class level as well as instance level, indicated in figure 2.1.

The *document form*, in the bottom-right quadrant, represents the perceivable document that conveys the structured document as specified by the stylesheet.

A form convention is represented by a style rule as part of a stylesheet. A *style rule* specifies a mapping between a functional construct and a form construct on class level or instance level.

A *functional construct* makes part of the structured document explicit so that a style rule can select it and transform it to its corresponding form construct

Subsequently, a *form construct* represents part of the document form that is produced by a style rule.

A *formatter* is a computer application which, independent of a particular document, applies formatting rules to a structured document and produces the form of a document. The formatting rules are described in a stylesheet. If the stylesheet conforms to a schema document and the schema document validates the structured document, then the stylesheet can be used to transform the structured document to document form.

For the formatter a structured document is a tree structured list of functional constructs. The document form is a tree structured list of form constructs. Abstracting from functional constructs and form constructs, the transformation, concerns an input tree that is transformed to an output tree. Starting at the root node, which becomes the active node, the formatter finds all style rules, of which the selector matches with the active node. From these matching style rules, the formatter selects one, according to a specific resolution strategy, and executes it. The result, which is a tree fragment, becomes the top of the output tree. Typically, the descriptor of the

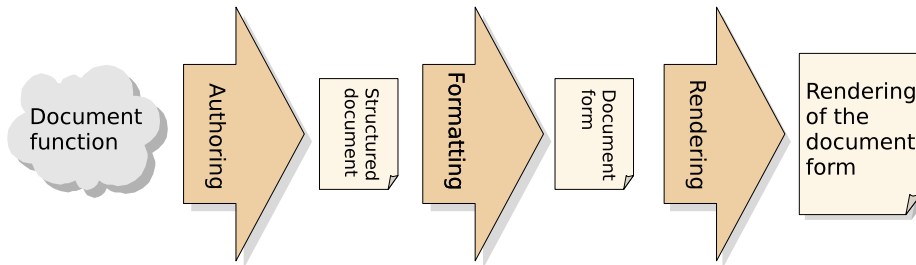


Figure 2.2: Transformation chain consisting of three transformation steps.

matching style rule also contains instructions that specify the continuation of the transformation process for the child nodes of the active node. The process continues recursively until there are no more nodes in the input tree to transform.

Defining the transformation chain

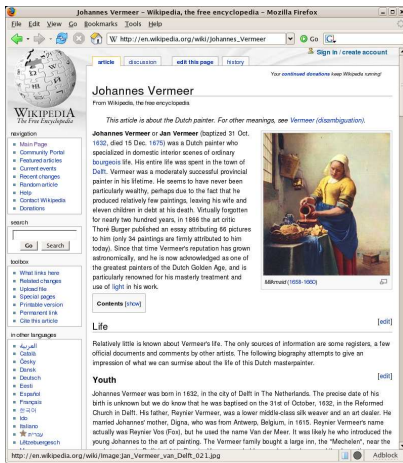
In section 1.1.1 we defined an electronic document as a structured dataset that required formatting rules to become perceivable. This is subtly different from separating function from form, since the dataset can be an encoding of the *form*. A facsimile format such as PDF is a structured dataset. However, it represents the form of a document, while the function remains implicit. In practice, the document engineering paradigm (transforming function to form) is often implemented by several sequential transformations where the output of the previous transformation is used as input for the next. This is referred to as the *transformation chain*, an individual transformation in the transformation is referred to as a *transformation step*. Although there is no maximum in the number of transformation steps in a transformation chain, most modern document engineering applications (e.g. \LaTeX , HTML) use a three stage document production process that consists of two (automatic) transformations². These three stages, illustrated in figure 2.2, are: authoring, formatting and rendering [61].

Authoring Document function in figure 2.2 refers to the intended message an author wants to convey. During the authoring phase an author attempts to materialize this message, while abstracting from the form used to convey this message. For example, an author indicates the intended function of a text, such as emphasis, without specifying the formatting used to convey emphasis in the document form. The result of the authoring phase is an explicitly structured dataset, which is often referred to as the structured document.

Formatting The form of a document is determined during the formatting phase. Based on the explicit function in the structured document, the form of a document is adapted to a particular delivery context. Effectively, perceivable properties are assigned to the explicit function described in the structured document.

The text that is designated as “emphasis” may, for example, be formatted using an italic font, which is traditionally used to convey emphasis. Alternatively, for a differently styled

²One can interpret the authoring of a structured document as a transformation from the document function as it resided in the mind of the author to the structured document.



Screenshot of a Firefox browser showing the Wikipedia entry for “Johannes Vermeer”.



Screenshot of RealPlayer showing a multimedia document about “Johannes Vermeer” and “genre painting”.

Figure 2.3: The rendered document form as perceived by the reader consists of both interface artefacts of the application rendering the document form, and the document form conveying the document function.

document, emphasis may be communicated using a bold font. The result of this phase is a specification of the document form.

Rendering The rendering phase is typically a straight-forward transformation from document form to the rendering of the document form that does not require design decisions that could alter the perceived document function. This can be a paper document, in which case the rendering involves sending formatting instructions to the printer. If it is a computer screen, rendering involves sending pixels to the screen. The result of this phase is the perceivable manifestation of the document form to a reader.

Defining document form

Figure 2.3 presents two screenshots that illustrate the document form. On the left, a text-based document about “Johannes Vermeer” is shown using the Firefox browser [40], on the right, a multimedia document about “Johannes Vermeer” and “genre paintings” is presented using RealPlayer [124].

The document form includes interface components, such as window borders, the title bar and interaction widgets that are perceivable, yet they do not convey the document function an author intends to convey. The interface of an application is often beyond the control of the author and therefore does not play an active role in the document engineering paradigm.

Roger T. Pédaque makes a comparable distinction for non-electronic documents. He defines

a non-electronic document as the inscription (e.g. text, figures) and the medium (e.g. paper), which “carries” the inscription [119].

Although medium and inscription are not defined for electronic form, the constraints imposed by the delivery context can be compared to the constraints imposed by the medium. For example, they both have a notion of available real estate for the form (e.g. paper size, screen size). Although we are mostly concerned with electronic documents, the notion of medium and inscription is a useful metaphor for this type of document as well. For our purposes we therefore extend the definitions of medium and inscription to include electronic documents.

The *medium* is the part of document form that is independent of the document function and defined by the delivery context. The *inscription* is the part of electronic form that is dependent on the document function³.

Note that according to this distinction, the navigational arrows below the Vermeer painting in figure 2.3 are part of the inscription. In contrast, the arrows indicating the start and end of the document are part of the interface and therefore part of the medium.

The inscription of a document traditionally includes handwritten or printed text and images. For electronic documents, the inscription may, in addition to text and images, consist of temporal media such as audio and video. A *media item* is a unit of data, identified and retrieved as one from a repository, which, using appropriate software, can be represented in a perceivable form (adopted from [76]). This definition includes spatial-temporal media items such as audio-clips, images and videos, as well as textual media items. In a textual document media items are atomic text fragments such as a paragraph or the title. Note that we are referring to the text only, thus without styling.

In addition to media items, the inscription of a document form consists of layout, style and optionally hyperlinks (based on the AHM [76]). We define the *layout* of an electronic document to define the spatial regions and/or temporal regions that are part of the inscription. This includes borders, margins, paddings (and if applicable, pauses and synchronization). The *style* of an electronic document defines perceivable properties of the document that can be adapted. This includes fonts, colors, alignment and transition effects. Style is used to optimize perception and improve the aesthetic value of a document. Note that some of these properties, such as font type or font size influence the layout. The *hyperlinks* of an electronic document define perceivable elements within the inscription that define explicit references with associated behavior to (parts of) one or more electronic documents.

2.1.4 Discussion

The advantages of document engineering, adaptation to the delivery context and reuse of style apply to both textual documents as well as multimedia documents. The document engineering algorithms developed for textual documents, however, cannot be applied to multimedia documents. The problems HyTime revealed concerning the tight relation of content and style in time-based media is still a topic of research today. In many ways, our own research addresses some of the problems initially raised by the HyTime community.

³We are aware of the inherent differences between electronic documents and non electronic documents and the problematic implications this has for the concepts of medium and inscription. For example, the implementers of the Firefox browser have decided to display the document title “Johannes Vermeer” in the titlebar, this is not a choice of the author. The definition of the <title> element in HTML [151] states that the title should be perceivable. Consequently, the titlebar is part of the medium but dependent on the function of the document.

Van Ossenbruggen [139] describes the overlap between electronic publishing and time-based hypermedia models. Although these share some technology, in general the developed models are not compatible. Van Ossenbruggen states two incompatibilities that also occur in our own work.

First, the document engineering paradigm advocates a source document that makes no explicit statements about the presentation of the document. The source document thus contains the intellectual content of the document but abstracts over its presentation. However, the intellectual content of a time-based hypermedia document is often established by combinations of media items with specific spatio-temporal relations between them. In this case the presentation is part of the intellectual content.

Second, structured document models for text-based documents are all based on text-flow. Once a line of text exceeds the margin of the page it is broken-up and continues on the beginning of the next line. Similarly, at the end of a page, once a line does not fit the current page it is automatically moved to the beginning of the next page, or a scroll-bar is added. Multimedia documents use temporal flow, which, in general, cannot be broken at arbitrary places as it can destroy the coherence of the document.

2.2 Knowledge engineering

When a multimedia item is perceived by a human perceiver she attaches particular semantics to the collection of bits that encode the media item. Although the media item encodes semantics it is typically hard to access for a machine. Nevertheless, some structures can be derived (partly) automatically. For example, for text, algorithms exist that can detect word bases and the language used. This can be used for automatic hyphenation in an electronic document [92]. For video, algorithms exist that detect shot boundaries and to some extent scene boundaries automatically, which facilitate navigation through a video [132]. Furthermore, image analysis algorithms allow retrieval of images based on visual features [4].

There is, however, a trade-off between the size of the domain these algorithms can be applied to and the detail of the automatically detected semantics. This trade-off is often referred to as the semantic gap [131].

For applications that require knowledge about the media that cannot be automatically detected, additional descriptions (i.e. metadata) are necessary.

Symbolic AI and knowledge engineering (or classic AI) is a branch of artificial intelligence that is concerned with representing (human) knowledge and semantics in a declarative form suitable for further processing by intelligent applications [45].

In the remainder of this section we first focus on the inherent problems of multimedia annotation. Part of these problems are addressed by the formalized vocabularies that are designed to encode particular types of knowledge that are relevant for our domain. Finally, we describe the semantic web, which is a framework designed for combining and reusing knowledge in a web context.

2.2.1 Issues with multimedia annotation

Although for some applications the requirement on metadata for media items is inevitable, there are some serious issues with describing media items. These include issues imposed by the person annotating the object, the object itself and the vocabulary used to annotate the object.

The “Metadata Twist”

Haase [73] argues for the importance of high quality metadata. He states metadata can often not be generated automatically and needs to be authored manually for a specific task. This leads to what Haase calls the *Metadata Twist*, “While the economic significance of metadata increases with the size of the available content, the average economic value of the content must, at the same time, decrease with the amount of available content.”

Van Ossenbruggen et al. [144] mention the following problems with metadata for multimedia documents.

Costs High quality metadata typically is expensive as a human author is needed to provide them. Although some automatic annotation is possible using feature extraction algorithms, most applications require high-level annotations for which human labor still is needed.

Subjectivity Annotations are typically highly subjective. Human annotators focus on different things and therefore interpret a media item differently. Furthermore, an annotator typically imagines a context for which the media item can be used, which influences the perspective of the annotation.

Restrictivity Formalized metadata schemata may partially limit the subjectivity of the annotation, they also limit the expressiveness of a human annotator. Subsequently, an annotator might not be able to express subtleties she finds important.

Longevity Annotation schema that can be used for current usage as well as usage in the long run are hard to define. This is because the context of the annotator and intended use of the media items typically change over time.

Standardization Annotations need to be made accessible to end-users. Standards are needed to ensure that the intended meaning is communicated.

Privacy Metadata might include personal or security sensitive information that should not be disclosed to other parties.

Granularity The scope of annotations might be different as some address the whole media item, while others refer to part of the media item. For example, the title of a film relates to the whole media item, while the location of action might change during the film.

In addition to these “general” issues concerning metadata, the association between the metadata and the digital artefact is not always made explicit [71]. In the next section we elaborate on the possible mismatches between a media item and its metadata.

Relation between topic, media asset and annotations

In multimedia, the digital artefact itself is often also “about” another (often non-digital) artefact. This results in a complex, at least four-way relationship between (1) the concept, (2) its annotation, (3) the digital media item and (4), its annotation. Figure 2.4 gives an example, which shows three distinguishable conceptual levels of media item and its annotation. On the upper left, we see the primary topic Saskia. The middle left image represents the painting, as it is hanging on the wall in the museum. On the lower left, there is a series of digital media items that are directly related to the painting: digital images of the painting in different sizes and resolutions,

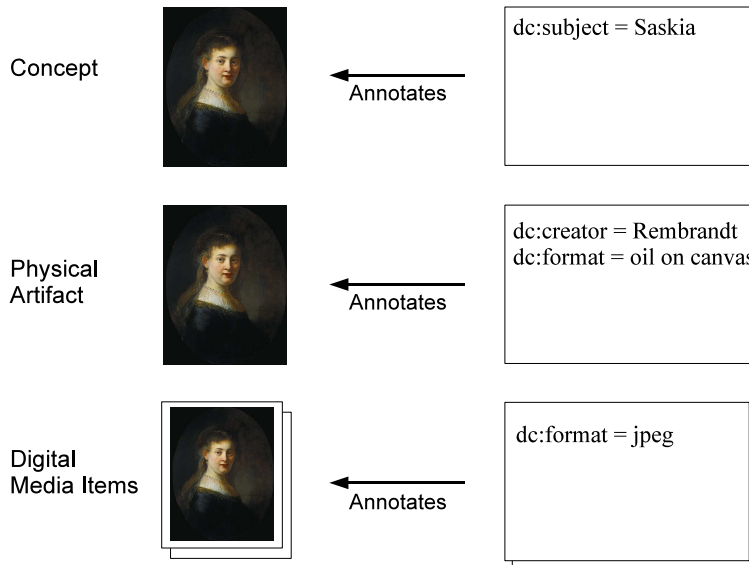


Figure 2.4: Media item and annotation record

X-ray images of the same painting, etc. The right half represents the metadata⁴. At the top, the annotations that describe Saskia might include her birth year, the place she lived, and that she was the wife of Rembrandt. In the middle, annotations about the painting might include the date of creation, the artist who created it, the physical dimensions and the room in the Rijksmuseum where it is currently hanging. Finally, the annotations about the digital media item might include the media type of the media item, the width and height in pixels and possibly copyright notifications.

All multimedia applications need to deal with the lower part of this figure to be able to show the media items (bottom left) and to process the different media types (bottom right). Most of them also need access to information about the topic of discussion, for example to make decisions about the relevancy of the media items. Unfortunately, the distinction between the different conceptual levels is not always made, and, if it is, the relationship between the conceptual levels has not been standardized or is even not made explicit at all [75].

These problems fall into three categories. First, the relationships need to be made explicit between real world objects and digital media assets representing them, and between metadata expressing concepts in the domain and metadata about the media assets. Second, multimedia applications require a distinction between content-level annotations and (technical) descriptions of the media asset. Third, multimedia applications often require metadata from a variety of different vocabularies [13, 67].

⁴Note that in some sense, the media items can also be considered metadata about the painting. Here, we stick with the more conventional definition of metadata.

2.2.2 Multimedia vocabularies

Controlled vocabularies are designed with a particular purpose in mind. Typically they are established within a community to achieve a level of consistency or agreement on the terms used to annotate. Depending on the task the vocabulary was designed for it has a particular level of formalization. Dublin Core [46], for example, provides a set of basic relationships that are typical within the digital libraries domain. However, since it specifies very few restrictions in the standard, the values allocated to a relationship are often ambiguous and therefore hard to process automatically by a machine. In contrast, CC/PP [152] is a vocabulary describing device profiles that are processable by a machine. Subsequently, the syntax and semantics are specified in great detail.

In the remainder of this section we present a number of vocabularies that are used in our work. Other relevant vocabularies, such as MPEG-21 [87], Visual Resources Association (VRA) [148], The Art and Architecture Thesaurus (AAT) [64], Iconclass [137], Media Streams [41] and PREMO [47] we do not describe here in detail.

MPEG-7

MPEG-7 [88] provides a standardized way to describe audiovisual data for searching, browsing and filtering in order to be used by third party applications. MPEG-7 specifically focuses on machine readable media descriptions rather than human accessible descriptions. It provides a set of audiovisual descriptors and description schemata describing the structure of a metadata element, which a user can use to describe audiovisual material. The standard is divided in eight parts, each focusing on different aspects of audiovisual content (adopted from [111, 112, 144]).

Systems specifies the binary encoding of MPEG-7 descriptions for efficient transport. These can be delivered independently, or included with the media they describe.

The Description Definition Language (DLL) specifies the syntax of the language for describing MPEG-7 descriptors.

Visual specifies descriptors that cover basic features of visual data, such as, colors, texture, shapes and motions.

Audio specifies a set of descriptors covering audio features, such as, spectral, parametric and temporal features.

Multimedia Description Schemes (MDS) specifies the generic descriptors for multimedia content (including visual and audio).

Reference Software specification of reference software that implements the standard, which can be used to validate compliant MPEG-7 implementations.

Conformance specifies guidelines and procedures for testing conformance of the standard.

Extraction and use specifies informative material on the use of descriptors.

MPEG-7 attempts to keep the application domain as broad as possible. This results in an elaborate standard in which a number of fields ranging from low level encoding scheme descriptors to high-level content descriptors are merged. Because there is no explicitly assumed use

case, MPEG-7 supports multiple ways of structuring annotations. In addition, domain dependent descriptors can be added if the default descriptors do not provide enough detail.

MPEG-7 is one of the few attempts to standardize multimedia metadata. Although we did not substantially use the MPEG-7 standard in our work, for reasons we will discuss later⁵, it did serve as a reference and inspiration.

Dublin Core

The Dublin Core Element Set (DC) [46] is an ISO standard for cross-domain information resource description. There are no fundamental restrictions to the types of resources to which Dublin Core metadata can be assigned. Dublin Core is a commonly used annotation scheme across different domains. It is small set of relations, identified by domain experts in the field of digital libraries.

Dublin Core currently identifies 15 relations, including `Title`, `Creator`, `Subject`, `Description` and `Date`. All of the relations are explicitly optional and there are no explicit constraints on the values of the relations. There are, however, suggested controlled vocabularies for some of the Dublin Core elements such as `Date`. Finally, Dublin Core does not make explicit statements on the syntax and language for denoting Dublin Core annotations. There are, however, recommendations about encoding Dublin Core annotations in XML, RDF and XHTML.

Some of the knowledge resources we use in our work use a subset of the Dublin Core schema as part of their annotation schema. Furthermore, in part of our work we use the Dublin Core schema for inferring additional relationships between media items (see chapter 5).

MIME

MIME [59] was initially designed to allow mail tools to identify the media type of email content, but is now also used by web browsers and many other multimedia applications to identify the type of a media item.

MIME specifies a MIME-type which consists of a `type` and `subtype`. For example, `image/JPEG`, has type `image` and subtype `JPEG`, referring to an image which is JPEG encoded.

We use the MIME standard to infer the type of a media item and what properties are defined for it. For example, every `image` has a `width` and a `height`, every `audio` file has a `duration` and every `text` has a `length`. Furthermore, we map the MIME type to the basic modalities where these are not explicitly defined for a media item⁶.

Cascading Style Sheets

Cascading Style Sheets (CSS) [28] is a style sheet language that allows authors and users to attach style (e.g. fonts, spacing, and aural cues) to structured documents (e.g. HTML documents and XML applications).

A style sheet rule consists of two parts, a *selector* that identifies the part of the document the rule applies to, and a list of *property-value pairs* defining the formatting of the selected part. For example, `H1: { font-size: 18pt }` denotes that headers(H1) use a font-size of 18pt.

Although not specifically meant for multimedia we use CSS (and XSL Formatting Objects) as inspiration for our hypermedia formatting objects (see chapter 5).

⁵In [98] we use MPEG-7 relations for mapping semantics to spatio-temporal relations.

⁶Modality refers here to the term used in semiotics, which denotes the way in which information is encoded for the presentation to humans [162].

Composite Capabilities/Preference Profiles

Composite Capabilities/Preference Profile (CC/PP) [152]) describes the structure and vocabulary of profiles for describing device capabilities and user preferences. It is mainly used for content adaptation in mobile environments.

In our work we use part of the CC/PP vocabulary for describing the delivery context.

Modality Theory

Modality Theory [23] aims at providing a taxonomy of output modalities that are used for creating multi-modal output in human computer interfaces. Modality theory is used to combine media assets in an effective way for the user. For example, an audio file that contains music and an audio file that contains speech may be played in parallel, whereas two audio files that contain music are better presented in sequence.

Based on modality knowledge in chapter 5 we define rules to dynamically combine media items. We use the following modalities to describe the properties of a media item:

Graphic (gr) modalities are perceived by the visual input sense of a user. Examples of graphic modalities include images, text and hieroglyphs.

Acoustic (ac) modalities are perceived by the hearing sense of a user. Examples include sound, music and voices.

Haptic (ha) modalities are perceived by the touch sense of a user.

Linguistic (li) modalities are based on existing syntactic-semantic-pragmatic systems of meaning. They can represent or express in principle anything, but require focus and consequently omit detail. For example the string “My neighbor” denotes a particular person but omits (irrelevant) details on what he looks like.

Analogue (an) modalities (sometimes called iconic) are complementary to linguistic modalities. They have the property of being specific but lack abstract focus. For example a picture of a man will precisely show what he looks like. However, stating that this man is my neighbor is hard to express using analogue modalities.

Non-Arbitrary (na) modalities rely on an already existing system of meaning and representation. Examples include natural language or mathematical notions which do not need further explanation because their semantics are shared between the communicating parties.

Arbitrary (ar) modalities must be accompanied with appropriate representational conventions in order to understand the representation. Some arbitrary representations become non-arbitrary as they acquire common use.

Static (st) modalities may be decoded by a user at any time and as long as desired. For example a photograph or statue.

Dynamic (dy) representations do not allow the user freedom of inspection but have a prescribed way of being communicated to the user, e.g., video.

2.2.3 Semantic Web

Most controlled vocabularies (including most of those described above) define their own syntax and semantics, preventing reuse of the vocabulary. DAML+OIL [138] was developed in the beginning of this century by the symbolic branch of the AI community, for providing uniform syntax and semantics for reuse of knowledge descriptions and tools processing this knowledge. Although DAML+OIL served as a generic knowledge description language it was, outside AI research labs, not often used.

In order to stimulate the exchange of metadata, the Open Archives Initiative (OAI) [94] defined an interoperability framework, the Open Archives Metadata Harvesting Protocol [38], to facilitate the sharing of metadata. Using this protocol, *data providers* are able to make metadata about their collections available for harvesting through an HTTP-based protocol. *Service providers* then use this metadata to create value added services. To facilitate interoperability, data providers are required to supply metadata which complies to a common schema, the unqualified Dublin Core Metadata Element Set [46]. Additional schemas are also allowed and are distinguished through the use of a metadata prefix. In addition, the Open Archives Initiative Object Reuse and Exchange (OAI-ORE) defines a vocabulary to describe aggregations of Web resources that may combine multiple media types [39]. This information may be used for authoring, deposit, exchange, visualization, reuse, and preservation. We used the OAI protocol to harvest metadata for the SEMINF use case described in chapter 5.

On a larger scale, the *Semantic Web* initiative aims to make data on the web accessible to machines. In this way, both people and machines can interchange information on the web [22]. Furthermore, relationships between data resources can be made, similar to hyperlinking mechanism commonly used on the “traditional” Web [21]. This way, available data may be reused (possibly for purposes it was not primarily designed for) and combined with other data to create “new” information.

Although the web for people is already a success, the web for machines is relatively young. In the remainder of this section we first explain the basic vocabularies used to describe knowledge on the semantic web. Although these vocabularies are suited for machine processing they are not very well suited to be interpreted by humans. Consequently, a sub area of the semantic web initiative intends to access and present this information to humans.

RDF, RDF(S) and OWL

The Resource Description Framework (RDF) [150] is a data model, standardized by the World Wide Web Consortium (W3C) (<http://www.w3c.org>). RDF is a language to describe a graph of statements. The semantics of RDF are formally defined in the standard [78]. In practice, however, they are often not sufficient for modeling a domain.

RDF Schema (RDFS) [156] adds additional formal semantics to a number of RDF statements. It defines the concepts of a *class* and *instance* that allow conceptual modeling of a domain.

The semantics supported by RDF Schema, however, are relatively simple. For more complex domain modeling more sophisticated semantics are needed, which are provided by the W3C OWL recommendation [157].

Figure 2.5 illustrates the modeling of a domain of *artists* and the *artefacts* they create using RDF(S). The interrupted line separates *Class* descriptions from *Instance* descriptions. *Class* descriptions model the domain conceptually by defining relations between concepts. For ex-

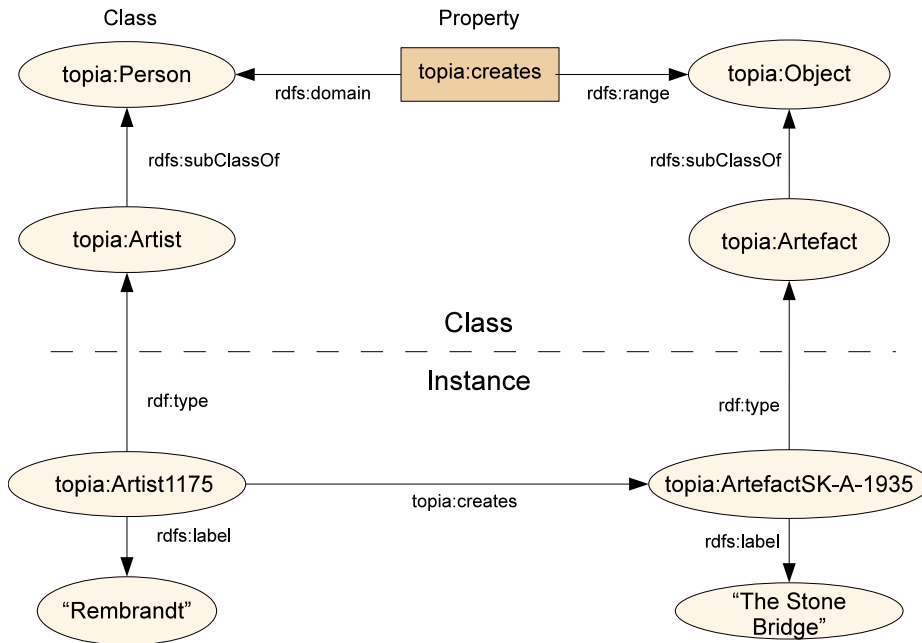


Figure 2.5: Graph representation of a RDF(S) graph modeling artists who create artefacts. (Graphical representation of part of the domain model used for the Topia project [126].)

ample, the concept `topia:Artist`⁷ is a special type of `topia:Person`, as specified by the `rdfs:subClassOf` relationship. Informally explained, the model states that persons create objects, and artefacts are special types of objects. Since artists are persons and artefacts are objects we can infer that artists can create artefacts. This is illustrated by the *Instance* descriptions, specified below the interrupted line. That the artist “Rembrandt” created the artefact “The Stone Bridge” is a valid statement in the model.

An RDF graph (which may include RDF(S) and OWL) can be encoded as an XML document, which can be stored on a server and downloaded by a client. Third party tools are available to process the knowledge encoded in the RDF file. We used Protégé⁸, which is an ontology editor, to define alignments between ontologies we used. Furthermore we used SWI-Prolog⁹, which is a Prolog implementation that allows reasoning with knowledge encoded in RDF, RDF(S) and

⁷On the (semantic) web an entity (or in web terminology a *resource*) is denoted by a Universal Resource Identifier (URI) [19]. A URI can refer to a document, such as <http://www.w3.org/Addressing/>, which explains addressing resources on the web. It can, however, also be used to denote an abstract entity, such as `topia:Artist`. In this case, the prefix (or namespace) `topia` is a short notation to denote <http://www.telin.nl/rdf/topia#>. The resource `topia:Artist` thus actually denotes <http://www.telin.nl/rdf/topia#Artist>.

⁸<http://protege.stanford.edu/>

⁹<http://www.swi-prolog.org/>

partly OWL. Finally, we use Sesame¹⁰, which is an database application for knowledge encoded in RDF and RDF(S).

Although the semantic web encodes explicit knowledge that is accessible and processable by a machine, this information is, in general, hard to interpret by humans. In the next section we elaborate on systems developed to present this knowledge in a format better suited for humans.

Presenting the semantic web

The semantic web aims at making resources accessible for machines. The intention is to provide more sophisticated technology for processing information. However, the graph structure and formal semantics are not easily accessible to human users. Therefore, in addition to encoding knowledge to be accessible for machines, the reverse, presenting knowledge to a human user is also necessary [63, 142].

The data structure of an RDF repository represents a directed graph. Subsequently, an RDF repository can be visualized by a graph, as illustrated in figure 2.5. The W3C RDF validator service [121] provides the option to automatically generate a visual graph of an arbitrary RDF file. This type of visualization works reasonably well for small graphs. To visualize larger graphs more customizable visualization tools are required [58].

Besides visualizing the RDF graph, which is arguably the most complete, there are other visualization tools that attempt to facilitate access to the information contained within the graph:

Noadster [128] generates hypermedia presentations in a domain-independent way, using clustering techniques. Figure 2.6 shows a presentation generated by Noadster about “Rembrandt”. Noadster presents a *structure view* (left side of Figure 2.6) which represents a higher-level structural view based on the clusters found, and *local view* (right side in Figure 2.6) which presents information about an individual resource.

/facet [79], illustrated in figure 2.7 supports browsing through the graph structure by means of dynamically selecting relevant properties (facets). Additionally, the interface allows the inclusion of facet-specific display options that go beyond the hierarchical navigation that characterizes current facet browsing. Foremost /facet is a tool for Semantic Web developers as it provides an intuitive interface to their complete dataset. However, the automatic facet configuration generated by the system can be further refined to configure it as a tool for end users as well.

Fresnel [25] defines a vocabulary for specifying how RDF graphs should be presented using existing style languages such as CSS. It uses the concepts of lenses and formats. A *lens* identifies the relevant properties of a `Resource` within a specific context. A *format* specifies how the selected information is presented.

ClioPatria [130] is a toolkit for Semantic Web applications, which provides search and presentation facilities using standard Web components. It is based on SWI Prolog [160], which provides a powerful search and exploration back-end of RDF data. The obtained result set may be visualized using off-the-shelf interactive widgets that exploit the semantic characteristics of the results, for example, clusters of image thumbnails, charts of aggregated property values, or the timeline or geographical views that are part of Simile Exhibit [82].

¹⁰<http://www.openrdf.org/>

Results for "Rembrandt"

The Company of Frans Banning Cocq and Willem van Ruytenburch, known as the 'Night Watch' [10]
Rijn
http://www.telin.nl/rdf/topia#Frame583_1_1
Here come the guards
Original location
http://www.telin.nl/rdf/topia#Frame583_2_2
Who is in the Night Watch?
Symbols
Utterly new: a moment in time!
Rendering of texture
Troublesome foreshortening
Self Portrait at an Early Age [8]
http://www.telin.nl/rdf/topia#Page380_6_2
Rembrandt and Lievens
http://www.telin.nl/rdf/topia#Frame380_6_2
http://www.telin.nl/rdf/topia#Frame380_1_1
Light and shade
Points of light and scratches
Famous Dutchman
Rembrandt in Leiden
The Sampling Officials [8]
http://www.telin.nl/rdf/topia#Page587_4_2
Preparatory drawings
http://www.telin.nl/rdf/topia#Frame587_4_2
http://www.telin.nl/rdf/topia#Page587_5_2
Signature
http://www.telin.nl/rdf/topia#Frame587_5_2
http://www.telin.nl/rdf/topia#Frame587_1_1
Background facts

Here come the guards

But where are they going? Although the militiamen in the Night Watch may appear to be positioned at random, Rembrandt has constructed the composition with great care. The drawing shows the positions and movement of the figures through the space, seen from above. The militiamen are coming out of the gate and moving


Property/Subject	Related Resource
Frame Image Description	The direction of the militiamen
Aspect	The Company of Frans Banning Cocq and Willem van Ruytenburch, known as the 'Night Watch'
Page Frame	http://www.telin.nl/rdf/topia#Page583_11
Frame Image URL	
RDF Type	Frame, RDFS Resource
http://www.telin.nl/rdf/topia#Frame583_11_1	

Figure 2.6: Visualization of RDF generated by Noadster [128]

Welkin [106] visualizes an RDF graph by graphically representing its nodes and the relationships between them. It is primarily meant as a tool for professional users of metadata repositories to quickly get an overview of a specific dataset. Although for large graphs (larger than 1MB of RDF) cluttering becomes inevitable, Welkin proposes several interactive selection and clustering mechanisms to untangle the graph as much as possible.

2.2.4 Discussion

The MPEG-7 standardization committee chose to use XML Schema for describing the MPEG-7 descriptors and description schemes because of its ability to express the constraints required for MPEG-7. Although the syntax can be automatically verified, the semantics of the descriptors cannot be processed by a machine. Hunter [81] proposes RDF and RDF Schema to express the semantics of MPEG-7 descriptors and provides an initial version of an MPEG-7 ontology. Since the semantic expressivity of RDF(S) turned out to be insufficient, the additionally required semantic constraints were obtained by adopting the DAML+OIL schema [138], the predecessor of OWL [80].

Troncy et al. [13, 135, 136] criticize the limitation in semantic extensibility of MPEG-7. A multimedia document can be described from a structural perspective (e.g. interview, shot, frame) and a semantic perspective (e.g. "Paris-Nice cycling race"). In practice, however, both are needed for search and retrieval of video fragments. For this reason, a multimedia description vocabulary

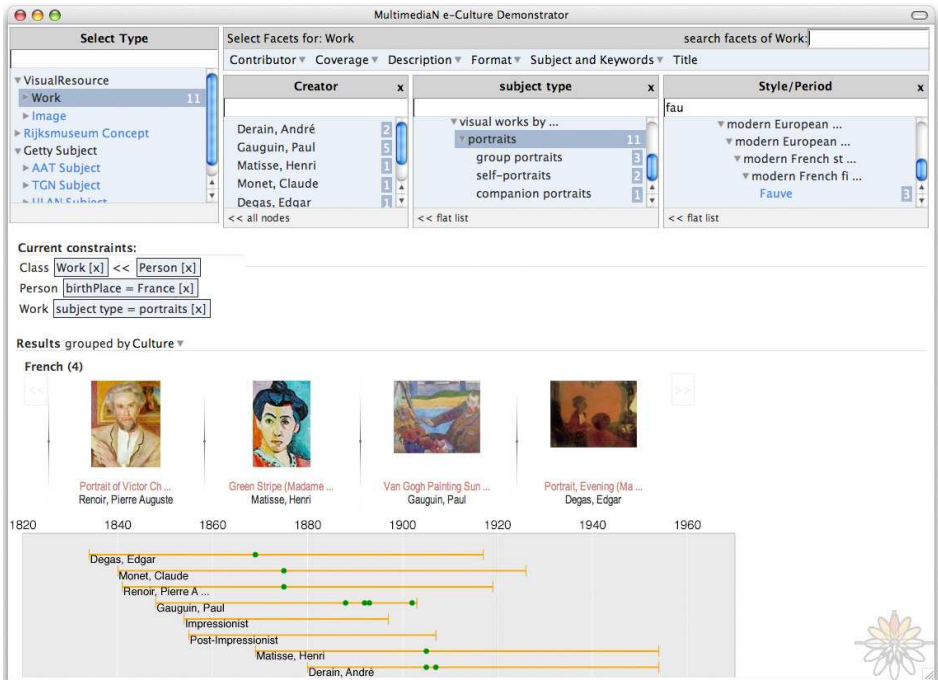


Figure 2.7: Faceted browsing RDF in \facet [79]

integrating both perspectives is desirable. Although MPEG-7 [88] is a logical candidate, it has some limitations: firstly, the expressivity of the semantics is insufficient for simple retrieval tasks, secondly the semantics of the MPEG-7 descriptors are not formally defined. Troncy proposes an architecture that uses MPEG-7 to describe the structural semantics of an audio visual document and semantic web languages (RDF(S), OWL) for describing the ontological semantics (see figure 2.4 on page 34 for an example of different conceptual levels in multimedia annotations).

Document engineering considers metadata to be outside the scope of the model. As a result, the treatment of metadata is relatively heterogeneous. Some document formats, such as PDF or Microsoft Word, allow *embedding* of metadata in the document form. In contrast, other document formats, such as Postscript, include no metadata. In addition, the metadata vocabulary is typically dependent on the document format used. Alternatively, an annotator may store metadata *external* to the document by referencing it. The document form, however, is typically considered a final product, and suitable “hooks” for attaching annotations are therefore often not apparent. There are exceptions, such as HTML, which allow an author to define anchors, which an annotator may use to attach an annotation. A disadvantage of this approach is that an author needs to identify *a priori* the relevant parts of a document that an annotator may want to annotate. Note that although the examples above are based on textual documents, the same issues apply to other media types, such as image, audio and video [14, 67].

Furthermore, annotation is typically considered a tedious and time consuming task that is

only worth the effort if there is a clear need or advantage. Even then, due to the large amount of data, manual annotation is, in many cases, no longer an option. Numerous media analysis algorithms have thus been developed to facilitate this task [131]. However, some of these algorithms detect information that is explicitly available during the production of the media. For example, the title and authors of a document are often made explicit in the structured document. Likewise, Nack et al. [113] propose a method to preserve the metadata that is available during the production process. For example, instead of transcribing the film after it is produced, the script (if available) can be used to annotate it. Furthermore, the information that denotes the starting and ending of a shot or scene is explicitly available in the tools used to montage the film. The tools and devices that are used to produce the film need to be metadata aware and include and synchronize metadata during the the intermediate stages of production.

The semantic web provides the framework for sharing knowledge resources. Valuable pre-semantic web knowledge repositories, which have proved their value in practical usage (e.g. Wordnet [55], Arts and Architecture Thesaurus [64]), however exist that were not designed for a semantic web context. Note that porting these repositories to the semantic web is not trivial as shown by Wielinga et al. [161] and our own work [68].

2.3 Software engineering

The term software engineering was used as a title for a conference organized by NATO in the late sixties on the production of large and complex computer systems [114]. Up until that time there was no general consensus on how to build and control complex software systems. Nowadays, software engineering is concerned with the development, operation, and maintenance of software [17].

This section introduces relevant work on software engineering in the context of document engineering and the generation of multimedia documents.

2.3.1 Software architectures for document engineering

Document engineering applies transformation rules to a source document to produce a target document. In most cases multiple transformations (e.g. authoring-formatting, formatting-rendering) are needed to get to the desired output format. In document engineering this is called the transformation chain (figure 2.2 on page 29), which is typically implemented using a pipe-line architecture.

Pipe and filters

A *pipe and filters* (i.e. *pipe-line*) architecture in software engineering is used to describe a system which consists of two or more sequential processing steps. A processing step is typically called a filter. The output of a previous filter is used as input for the next filter, a pipe is used to connect the two. A filter reads its input, performs a transformation and produces output. A pipeline architecture requires that both the input and output are well described. The filter is considered to be a self contained black box, which can be implemented independently from other filters. The main advantage of the pipe and filters architecture is that pipes and filters can be combined independently. Because of its modularizing nature, it is relatively easy to extend or modify a system by adding or removing filters. In addition, it allows a complex process to be broken up

by multiple less complex processes, which reduces errors and improves maintainability of the system [17, 62].

Since the transformations in a transformation chain are sequential, the pipe-line architecture perfectly fits a document engineering application.

XML and XSL(T)

Document engineering technology used on the web, such as XSL(T) stylesheets, specify the transformation from an input XML document, to an output document, which is also specified in XML. Since input and output are the same format, the output of a transformation can be used as the input of the next transformation. This way, transformations can be chained, modularizing the transformation process in reusable components.

Transformation languages, such as XSL(T), exploit the pipe-line software architecture by using a functional specification of the transformation. In contrast to imperative languages, functional languages have no side effects influencing the execution of the program. As a result, a formatter can transform a document more efficiently, by transforming part of document, which is sent immediately to the next transformation, independently of the rest of the document.

Cocoon [134] is a web based framework which can be seen as an intelligent manager of the transformation chain. It adapts the form of a document on the fly by influencing the path it traverses through the transformation chain. Cocoon exploits the advantages of a pipe-line architecture by reusing filters and using the concepts of pipes to link them together. Lemmens [96] achieves a similar result by automatically generating XSL(T) stylesheets adapted to a specific delivery context.

Representational State Transfer (REST)

In a web-based context the document transformation chain, and consequently the software architecture, is split across a network. Roy Fielding describes, in his thesis *Architectural Styles and the Design of Network-based Software Architectures* [57], the consequences of applying the document engineering paradigm on the web. He presents a software architectural style, Representational State Transfer (REST), that is optimized for large scaled distributed hypermedia systems such as the web.

An important concept in the REST architecture is the notion of a *resource* denoting a specific source of information, which is uniquely identified by a *URI*. A client (or server) can access a resource by referring to its URI. Furthermore, the REST architecture states that server and client should conform to a *well defined protocol* to allow communication between an arbitrary server and an arbitrary client. Furthermore, due to the scale of the web it is impossible for a server to keep track of the individual transactions with a client. Therefore, the communication is *stateless*, which means that every transaction stands on its own. If history is needed it should be sent along with the transaction.

In addition, since server and client are in principle independent, the communicated data-structure should conform to a *commonly accepted format*.

Software architectures used in Cuypers

For the implementation of Cuypers, our multimedia document document engineering framework described in chapter 5, we used a combination of constraint programming, logic programming and object orient programming.

Constraint Programming (CP) [10] is a programming paradigm which, instead of solving a problem using a procedural approach like imperative programming (e.g. Java [15], Python [99]), allows a programmer to declaratively specify the problem in terms of formalized constraints. A constraint solver then automatically uses *domain reduction* algorithms to efficiently compute an answer to the problem at hand. Constraint programming is often applied to domains which can be modeled using arithmetic constraints as there are numerous off-the-shelf arithmetic constraint solvers available. For non-arithmetic domains CP can be useful too, in which case the domain reduction rules need to be specified explicitly.

Logic Programming (LP) [115] is a declarative programming paradigm using *unification* and logic reasoning to find variable bindings that satisfy a certain statement. Unification is a mechanism that binds variables to values based on the equivalence of statements. For example, the statements $f(X)$ and $f(a)$ are unifiable, since the value a can be assigned to the variable X . Note that $f(X)$ is a generalization of the term $f(a)$ since there are many other statements, such as $f(b)$, $f(c)$ and $f(d)$ that can also be unified with $f(X)$. In contrast, the statements $f(a)$ and $g(X)$ cannot be unified, since there is no variable binding possible that makes $f(a)$ equal to $g(X)$. There can be multiple possible unifications for a statement which are enumerated in a *search-tree*, where every branch represents a possible unification. Prolog, a logic programming language, uses a technique called *backtracking* when the Prolog interpreter discovers an inconsistency. If an inconsistency is detected it returns, in a depth-first manner, to the last unification which has multiple alternatives, and selects the next possible unification. The program then continues with the new unification and tries to prove the statement.

Logtalk [42] is an Object Oriented Programming (OOP) [50] extension on top of the Prolog language. It provides most of the features of an Object Oriented language, such as inheritance, separation of interface and implementation. It is meant to improve the software engineering capabilities of Prolog [109].

Constraint Logic Programming (CLP) [12] combines Logic Programming and Constraint Programming by allowing *constraint variables* to be treated as *logical variables*. The difference between a logic variable and a constraint variable is that a logic variable does not have an explicit domain whereas a constraint variable does have an explicit domain. By adding a constraint, the domain of a constraint variable is reduced so that it only consists of values that can participate in a solution. For example, consider two integer constraint variables with their respective domains: $X \in \{1, 2, 3, 4\}$, $Y \in \{3, 4\}$. The constraint $X > Y$ states that the value of X is larger than the value of Y . However, since the smallest value of Y is 3, the value $X \in \{1, 2, 3\}$ will never lead to a solution, consequently they can be removed from the domain leaving $X = 4$, $Y \in \{3, 4\}$. Since the constraint is defined for two variables, in addition to reducing the domain of X , the domain for Y can also be reduced. In this case, the value of $X = 4$ needs to be larger than $Y \in \{3, 4\}$. The only value that qualifies is 3, leaving $X = 4$, $Y = 3$. In this case both variables have been reduced to a single value, and can thus be treated as “normal” logic unifications. However, a constraint variable can also be reduced to an empty domain. For example, $X \in \{1, 2, 3, 4\}$ with the constraint $X > 5$ reduces to $X \in \{\}$. This means there is no value which can satisfy the constraint and thus leads to a constraint violation. In a traditional Constraint Programming language the result of the program would therefore be something like “There is no solution to the constraints you imposed.” Constraint Logic Programming, in contrast, treats a constraint violation¹¹ as an inconsistent unification and backtracks to the last unification to try an alternative value, as would be the case in a Logic Programming language. Consequently, an alternative set

¹¹In this thesis we also use the terms conflicting constraints, inconsistent constraints, and constraint failure to denote constraint violations.

of constraints can be imposed which might lead to satisfactory solution.

Several consistency notions for constraints exist in CLP (and CP), which relate to the efficiency with which a constraint satisfaction problem (CSP) can be solved. A CSP is *arc-consistent* if all domain values of all variables in the CSP participate in a solution to the CSP. For example, the CSP: $X \in \{1, 2, 3\}, Y \in \{3, 4\}, X < Y$ is arc-consistent, since all domain values participate in a solution. In contrast, the CSP: $X \in \{1, 2, 3, 4\}, Y \in \{3, 4\}, X < Y$ is not arc-consistent since $X = 4$ does not participate in any solution. For CLP arc-consistency is an important notion since it guarantees that a solution to a CSP can be found without backtracking.

The domain of a variable get reduced by participating in one or more constraints. However, after all the constraints have been applied there are often multiple values remaining in the domain. In order to solve the CSP the formatter assigns a domain value for each variable. In CLP this process is called *labeling*. Just like a constraint, an assignment reduces the domain of the variable (to a single value), which is propagated to the domains of other variables that share a constraint with the labeled variable. For example, consider the CSP $X \in \{1, 2, 3\}, Y \in \{3, 4, 5\}, X < Y$. If the labeling assigns $X = 3$, the domain of Y is reduced to $Y \in \{4, 5\}$, which are the only values that can participate in a solution. Alternatively, if the labeling procedure assigns $Y = 3$, the domain of X is reduced to $X \in \{1, 2\}$. The ordering in which variables are labeled thus influences the solution to a CSP. There are different strategies to label a constraint variable, for example, minimal labeling assigns the smallest value from a domain to a variable, whereas maximum labeling assigns the largest value from a domain.

Note that labeling of an arc-consistent CLP, in contrast to a CLP that is not arc-consistent, is guaranteed to succeed without revising (i.e. backtracking over) an assignment.

In Cuyppers we implement the detection and resolution of constraints using CLP technology.

2.3.2 Generating multimedia

In the early nineties of the previous century the field of automatically generating multimedia presentations and interfaces was quite active. This was mostly due to the technical advances of computer systems that were capable of presenting multimedia content. Despite the development of authoring tools, authoring multimedia presentations remains more complex than authoring textual documents. This is mostly because a multimedia author not only needs to possess skills to represent information using different media types, these media types also need to work coherently together, from a technical perspective, a user perspective and a design perspective. The second generation of authoring tools, such as COMET [54], WIP/PPP [7, 8], DArt_{bio} [18] and the work on relational grammars of Weitzman and Wittenburg [159], which we describe next, made an attempt to overcome this problem by allowing an author to express the intended message on a higher level.

Relational grammars for multimedia documents

Weitzman and Wittenburg [159] acknowledge the need for higher level authoring. They argue that for automatically adapting a multimedia document to the context it is played in, a different authoring paradigm for multimedia documents is needed. They propose relational grammars, which are an extension of string-based grammars. This way, part of the design knowledge used by an author to create a multimedia document can be described and reused for other documents. The grammar specifies rules which match relationships between objects, and outputs a set of constraints that specify how the objects should be presented. For example, figure 2.8 presents a grammar rule that states that an `Article` (0) is produced by a number of different objects,

including Text object (1), Text object (2) that has an Author-Of relation with Text object(1), a third Text object (3), which is a Description-Of Text object(1). These relationships are conveyed graphically using the constraints defined at the bottom of the rule. The grammar consists of multiple of these sets of rules, each of which adapts to a specific delivery context.

```

;;; Article(0) -> Text(1) Text(2) Text(3) Number(4) Image(5)

(Defrule (Make-Article The-Grammar)

;;; Object      Relation

(0 Article)
(1 Text)
(2 Text      (Author-Of 2 1))
(3 Text      (Description-Of 3 1) )
(4 Number    (Page-Of 4 1))
(5 Image     (Image-Of 5 1))

;;; Geometric Constraints

:OUT
( (right-of 1 5)
  (right-of 2 5)
  (right-of 3 5)
  (right-of 5 4)
  (top-aligned 1 5)
  (top-aligned 5 4 )
  (spaced-below 2 1)
  (spaced-below 3 2 )
  (set-font 1 10pt :bold)
  (set-font 2 8pt :italic)
  (set-font 3 8pt :plain)
  (set-font 4 10pt :plain) ) )

```

Figure 2.8: Example of a relational grammar rule which produces an “article” [159] (comments and indentation added).

The rules system of Weitzman and Wittenburg showed that (semantic) relationships between media items can be mapped to formatting constructs, and that, depending on the delivery context, alternative rule sets can be used.

The advantage of this approach is that an author can provide multiple rule sets that provide alternative presentations of the same information. The main objection, however, is that for every distinct delivery context an alternative rule set needs to be formulated. Furthermore, the rules are domain dependent and require a relatively predictable input stream.

Despite this, we use a comparable grammar for mapping media items based on modality properties to presentation constructs (see chapter 5). However, since there is a finite set of modalities that can be associated with a media item (section 2.2.2), the input stream is relatively predictable and therefore this restriction does not cause problems.

Weitzman and Wittenburg focus on the automatic adaptation of a document to different delivery contexts. It assumes that the function of the presentation is given. This is different from

other approaches at that time, which typically included automatic construction of the function as well (e.g. COMET, WIP/PPP).

COMET

The fully automated AI approach to generating multimedia documents focuses on documents for specialized domains. For this type of document the investment of a human multimedia author would not pay-off and therefore generated presentations are a viable alternative. Examples include technical maintenance manuals that give instructions to a mechanic how to perform certain operations.

The COMET system generates “multimedia explanations” (the authors prefer to use the term “multimedia explanation” rather than multimedia document) that explain how to perform diagnostic tests for a particular radio receiver-transmitter [54]. The resulting documents include step-wise textual instructions, accompanied by example illustrations, of the action to be performed.

In response to a user request, the COMET system first determines *what* to say. A planner¹² queries three knowledge bases, which encode domain objects and actions, a diagnostic rule-set and detailed geometric knowledge, to produce a formalized representation of the sequence of actions the mechanic has to perform. Then COMET determines *how* to express this using multimedia content.

Both the textual explanations as well as the example images are automatically generated. These processes, however, cannot be independent since there are inter-dependencies between the media items and the layout of the page. For example, a reference number or symbol in the text should correspond to the embedded symbol in the image. Or the width of an image determines where a line-break should occur in the explanatory text. Furthermore, there is a two way dependency between the generation of media items and the generation of the overall presentation. For example, the dimension of a media item is dependent on the available size set by the delivery context, whereas the size of the overall presentation is dependent on the generated media items.

Although the primary objective of the COMET system is to make a particular knowledge base accessible to a user, the steps involved in producing a multimedia explanation are similar to producing a multimedia document. Based on this, we can make a number of observations on the COMET system.

Firstly, the advantage of generating media items is that it exactly matches the requirements of the (automatic) author. On the downside, generating media items is far from trivial. It requires extensive knowledge and significant computational resources (five workstations) and produces, compared to human labor, media items of mediocre presentation quality.

Secondly, In addition to expert domain knowledge for answering the query of the user, COMET uses extensive knowledge bases for describing the generation of media items and the generation of the overall presentation. Because of the tight integration of the knowledge bases and the explicit links between the knowledge bases, the COMET framework is not optimized for generating presentations for different domains.

Finally, COMET showed there are two-way dependencies for which there exists no straightforward strategy that leads to the best possible solution. Similar dependencies exist in our own work, which we discuss in chapter 4.

¹²A planner is an algorithm that given a particular goal and constraints produces the actions that when carried out will lead to desired goal [125].

WIP/PPP

WIP/PPP is comparable to COMET in the sense that it generates multimedia presentations explaining the use, or assembly, of electronic devices [7, 8]. In contrast to COMET, WIP specifically aims at an architecture which is capable of adapting the presentation to specific requirements, which include the user (language, expertise, preferred medium), the delivery context (space limitations, capable of playing audio) and the domain. Consequently, the architecture of the WIP system identifies generic phases in the authoring process, which are independent of a specific domain, user or device, which are considered as external inputs to the system.

The WIP/PPP system first selects and retrieves relevant knowledge from one or more knowledge-bases. The selected knowledge is grouped and ordered to produce a coherent semantic structure. The system then selects modalities that are most appropriate for communicating the intended information in terms of media content, taking into account the preference of the user and capabilities of the delivery context. After this, the media content is generated. Finally, the generated media items are spatially arranged and temporally coordinated in a layout and presented to the user.

WIP/PPP, just as COMET, generates most of the required media items. Although the WIP system is capable of generating presentations for multiple domains, there are significant knowledge requirements that need to be explicitly described by a domain expert.

DArt_{bio}

DArt_{bio} automatically generates biographies based on data provided by a knowledge-base. In addition to the automatic generation of textual descriptions, DArt_{bio} specifically focuses on the automatic generation of layout. Bateman et al. [18] divide the construction of page layout (i.e. typographic design) into three parts: *Micro-typography*, which represent the formatting of the smallest units in a document (e.g. characters, spaces, words etc.); *Macro-typography*, which describes the formatting of the largest units in a document (e.g. columns of text, placement of figures etc.); and *Style*, which represents the perceivable properties of a document (e.g. font, color). Unlike COMET and WIP/PPP, which focused on micro-typography (i.e. the generation of media) the approach taken by Bateman et al. is more concerned with macro-typography. Bateman argues that the way elements are presented on a page is partly responsible for the effectiveness of the message conveyed by the document. However, the design of a page-layout currently depends largely on the creativity of the designer.

Bateman's hypothesis is that a correspondence exists between the rhetorical structure of the document and the layout used to present it. By manually analyzing existing page layouts and relating them to the rhetorical structure of the text, they developed a set of heuristics that allowed them to automatically decompose an RST tree into hierarchical layout units¹³. An example heuristic states that a nucleus and its satellites should be part of the same logical unit. These layout units are formatted bottom-up, expressing an RST relationship by a spatial proximity relation between the layout units.

¹³Rhetorical Structure Theory [102], although originally intended as a model for generating text, is often used by linguists for analyzing the discourse structure within a text. RST defines relationships between spans of text. This can be a binary relation, in which case the *Nucleus* denotes the target span, relative to the source span, which is called the *Satellite*. There are 21 relations defined for *Nucleus-Satellite* relations, which include *Elaboration*, *Condition*, *Motivation*. There are also *Multi-nuclear* relations which define a relationship between multiple spans of text, for example *Sequence* and *Contrast*.

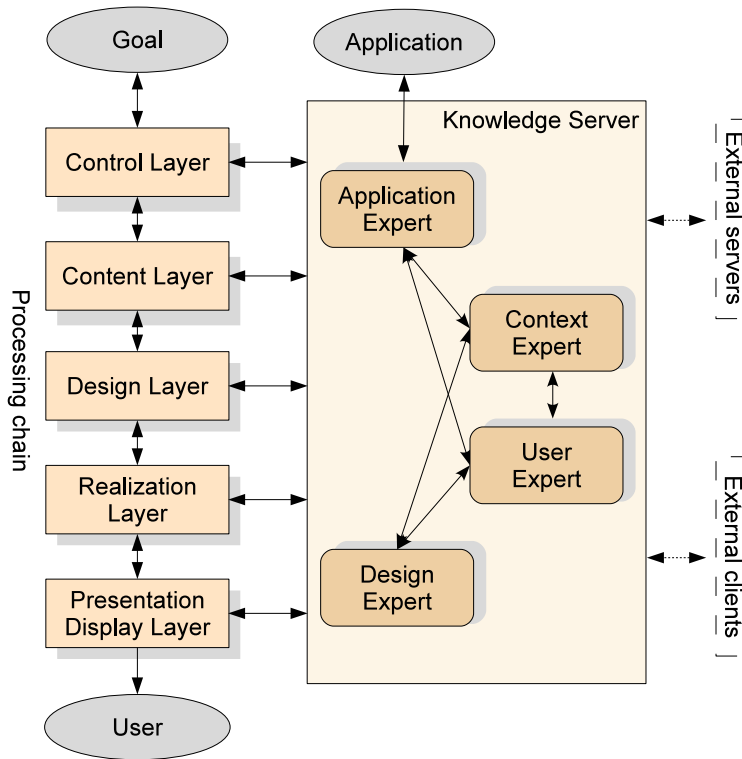


Figure 2.9: The Standard Reference Architecture for IMMPSs [27]

Although Bateman’s work is primarily designed for text-based documents, we used RST representations in one of our evaluation scenarios (see section 6.2 on page 117).

The Standard Reference Model for Intelligent Multimedia Presentation Systems

The Standard Reference Model for Intelligent Multimedia Presentation Systems (SRM-IMMPS) is a joint effort of various projects active in the development of IMMPSs, such as the COMET and the WIP/PPP systems. Its goal was to define a commonly agreed on reference model [27].

In the late nineties of the previous century, the need for such a model was motivated by practical problems the community was dealing with. These include the unintentional and inefficient replication of results due to the lack of a vocabulary to identify common parts in IMMPSs architectures. Furthermore, existing solutions were hard to reuse for other projects because of the use of different terminology and abstractions. Finally, the absence of a reference model limited the synergy between similar projects as systems were hard to compare.

Figure 2.9 represents the architecture graphically. On the left hand side are the processing layers. On the right hand side, the knowledge server that contains the knowledge required by the

processing layers. The *goal formulation* and *application* provide the initial input to the system¹⁴. The presentation is generated for a *user*, who is presented at the bottom¹⁵.

The bi-directional arrows between the processing layers indicate that the system occasionally needs to backtrack to a previous layer to resolve a problem in the current layer. Note the difference with the single processing direction in text based document engineering.

Control Layer The first layer of the SRM translates the incoming presentation goal, as specified by the user, to a format understandable by the IMMPS.

Content Layer The formalized presentation goal is specified at a relatively high-level of abstraction (e.g. “Give information about X”). The content layer extends and refines the high-level goal to a number of subgoals which can be processed by parts of the IMMPS (e.g. “Formulate a query which retrieves media related to X”, “Formulate a query which retrieves knowledge about X”, “Formulate a query which retrieves device characteristics.”).

These subgoals are executed by the system and relevant information and data is retrieved from the *Knowledge Server*. Furthermore, the system chooses what combination of modalities to use in order to communicate this information to the user, keeping in mind the restrictions of the device and the characteristics of the user. The output of this layer are *media communicative acts*. These are multimedia equivalents of communicative-acts, which achieve a particular effect on the user (e.g. “explain concept X”) [105].

Finally, the content layer determines the structure and order in which the selected information should be presented to the user.

Design Layer The design layer performs two tasks in parallel, planning the generation of media items for the *communicative acts*, and the design of the layout which communicates the order and structure of the information. The generation of media depends on layout (e.g. the media items need to fit within the space allocated by the layout). Conversely, the design of layout is dependent on the available media items. Since there is no justification why one should precede the other they need to be performed in parallel, negotiating choices which are mutually dependent. Depending on the application at hand and a preference for either layout or media, the system can define an order between the two.

The output of this layer are *realization plans*, which, when executed, relatively straight forwardly produce media and layout.

Realization Layer The fourth layer performs the realization of both the media and layout *realization plans*.

The output of this layer is a specification of presentable media objects together with layout information. It contains all the information necessary to play the presentation. The format of this phase is a good candidate for standardization since it would allow independent third party players to present the document.

¹⁴In practice the application and the goal formulation are often combined in a single component. Here they are separated to distinguish between the availability of application knowledge and the use of application knowledge to satisfy the presentation goal.

¹⁵Typically a user would interact with the system by formulating the goal and then obtain the result in the form of a generated presentation.

Presentation Display Layer The final layer is responsible for rendering the presentation specification produced in the previous layer, so that it is perceivable by the user.

The *Knowledge server* provides the layers of SRM with the necessary knowledge to generate a multimedia document. Note the difference with the document engineering transformation chain, which is independent of external (knowledge) resources. The knowledge server consists of *experts* (i.e. *agents*) that cover a particular aspect of the presentation such as *application*, *user*, *context* and *design*.

User Expert The user expert maintains a model of a specific user currently using the system. This knowledge may include physical and mental abilities, preferences, knowledge and beliefs.

Application Expert The application expert provides the system with application specific knowledge. This includes the current state of the application (e.g. the user has paused the presentation). It also makes (generic) knowledge bases accessible to the specific needs of the application (e.g. conversion of data format).

Context Expert The context expert keeps track of the context of the presentation. This includes knowledge about what has been presented to the user before, and in which way the user has interacted with the system. This type of information allows the system to be personalized. For example, it can prevent the system from presenting the same information twice to the same user.

Design Expert The design expert contains information needed for an IMMPS to present certain types of information. This includes knowledge about layout, colors, modalities, design constraints and design characteristics of the output device.

The SRM-IMMPS and the systems it was based on were developed by research groups with a strong background in Artificial Intelligence (AI). As result, a number of AI technologies were suggested for implementing parts of the SRM-IMMPS. For example, a planning system could be used for organizing the steps involved in creating a presentation. This includes plans for generating media as well as plans for combining them together in a presentation. A knowledge server could be used for providing the knowledge about the domain, the design and the user, required by the planner for making decisions concerning the adaptation of the presentation. Finally, experts (i.e. agents) could negotiate trade-offs in the desirability of certain features. For example, a large font increases readability, however, it might require too much space in a particular context.

The reference architecture provided by SRM-IMMPS gives a good overview of the processes, data-flow and required knowledge resources in an intelligent multimedia presentation generation system, which is still mostly valid today. From a practical perspective, however, the knowledge and computational resource requirements are too demanding to successfully build such systems on a large scale. More specifically, the investment necessary for describing the required knowledge makes the approach only suitable for small focused domains. Furthermore, as the SRM-IMMPS assumes a closed architecture, porting the system to other domains is difficult.

2.3.3 Intelligent multimedia systems on the web

The automatic generation of media items is considered part of the reason why systems based on SRM-IMMPS are only viable in specific domains. When, at the beginning of last century

media formats, such as MPEG4 [86] were standardized, multimedia data became more readily available. Furthermore, the web started to emerge, which inspired the reuse of media.

Worring [164] proposed a system design which (re)uses existing multimedia data stored in a multimedia database, such as the Monet multimedia database. The Monet system by Windhouwer et al. [163] uses an automatic webcrawler, which retrieves media items in bulk from the web. The media items are analyzed and the results are stored as metadata in a relational database. In addition to advanced querying of web based resources, the available metadata also allowed reuse of media items in a web context. Up until then, media resources were stored and used locally.

From the hypermedia community, Hardman et al. [77] proposed a model which integrated SRM-IMMPS with the Amsterdam Hypermedia Model, which explicitly supports reuse of web based media items.

Euzenat et al. [51] argue for generic adaptation techniques for multimedia documents. They propose a framework of formally described constraints that describe the document in a qualitative way. They compare a qualitative description of a multimedia document to writing a textual document in \LaTeX instead of PostScript. Since a qualitative description is closer to the semantic intention of the author it is better suited for applying adaptations without changing the intention of the document. For example, a constraint solver can decide to present two images in sequence rather than in parallel if the resources of the delivery context (e.g. small screen) are insufficient.

In the remainder of this section we introduce the web based information systems (WIS), Hera, Artequakt, SampLe and Vox Populi that deal with multimedia content in a web environment.

Hera

Hera [147] is a methodology that supports the design and engineering of a Web Information System. It distinguishes three layers:

Semantic Layer The semantic layer defines a meta model (*conceptual model*), which provides a homogeneous view of the (external) structured content resources used in the WIS.

Application Layer Based on the conceptual model the application layer defines an abstract hypermedia presentation structure. This presentation structure includes adaptation strategies based on a device or user model.

Presentation Layer The presentation layer combined the instance data provides by the semantic layer, and uses it to instantiate the presentation structure as defined by the application layer, while applying the required adaptations on the fly.

As in document engineering, Hera explicitly separates function (Semantic Layer and Application Layer) from form (Presentation Layer), consequently it can present a collection of data, without the need to author each document individually. In addition, the approach allows adaptation to the delivery context.

Hera primarily focuses on the function of a presentation (Semantic Layer and Application Layer). For the form (Presentation Layer) Hera uses a relatively simple XSL(T) transformation for presenting a document in various formats (e.g. HTML, SMIL, WML). If more sophisticated formatting is required, it refers to external formatters such as Cuypers [140].

Artequakt

Artequakt [89] generates artist biographies. The system uses a biography ontology that defines the data for an artist biography. Information is collected by parsing text found on the Web and is presented using templates. This allows, at least in theory, the reuse of textual information published on the Web about artists.

Artequakt generates artist biographies that are comparable with our own work (see DISC and SEMINF in chapter 5). It focuses primarily on automatic knowledge extraction from web pages and the narrative representation on these in a biography. As such, Artequakt is text-based and does not consider the issues of presenting biographies using multimedia data.

SampLe

SampLe [52] is a system that, rather than generating presentations fully automatically, provides intelligent support during the authoring phases of a presentation. It distinguishes the following authoring stages for which it provides support.

Topic The author of a presentation needs to select a topic of the presentation. The topic of a presentation can be known *a priori* in which case the *topic selection* is the starting point of the presentation authoring. Alternatively, an author can browse through the available material in order to select a topic of her liking. In addition to browsing facilities, SampLe portrays associated domain semantics to allow the author to become familiar with the domain.

Discourse Structure The structure of a presentation typically follows a genre, such as biography, essay or article. The selection of a discourse structure is dependent on the available media content and the message an author intends to convey. SampLe suggests one or more genres which a user can manually adapt to fit the message intended.

Media Content The content of the presentation is the actual media items used in the presentation. The selection of media items depends on both the topic of the presentation, as well as the genre and the discourse role the media item plays within the discourse structure. SampLe either suggests existing media items which are stored in a repository, such as images and text fragments which fit the requirements of the topic and genre, or allows an author to create or include new media material.

Style In addition to structure and content, an author also needs to choose appropriate layout, colors and typography. These are dependent on the topic of the presentation and the genre, as well as external factors such as the user group the presentation is authored for. SampLe suggests stylesheets for presenting the presentation in HTML as well as exporting a presentation structure which can potentially be interpreted by a presentation generation engine such as Cuypers [140].

SampLe makes the phases in an authoring task explicit and acknowledges the dependencies between them. Similar dependencies are present in a fully automated system (see 4). SampLe resolves these dependencies by presenting suggestions to a human author. A human author can decide to follow a suggestion if she feels it is appropriate, otherwise she can intervene by authoring the part manually. By including a human author in the loop, SampLe is potentially capable of generating more sophisticated presentations than a fully automatic system. However,

SampLe is an intelligent authoring system, designed to address a single document at a time, whereas our document engineering system addresses a class of documents.

Vox Populi

Vox Populi [26] is a system that allows users to automatically generate video documentaries supporting or attacking a point of view selected by the user. The existing video clips, interviews with people about the terrorist attack of 11th September 2001 in the United States, are annotated with statements denoting the argument conveyed in the clip. Based on these annotations, knowledge of constructing an argument and cinematographic knowledge, the system automatically assembles a sequence of video clips that support or attack the selected point of view.

Vox Populi shows the importance of contextual information for the interpretation of a media item. For example, Vox Populi sometimes uses the same clip in different contexts to express different point of views. For example, a claim conveyed by a particular clip can be made stronger by adding material that backs up the claim, or made weaker by adding material that opposes the claim. In our own work, similar contextual dependencies are shown by using the same image as a portrait in a biography of Rembrandt, or illustrating the painting technique chiaroscuro (see chapter 5).

2.3.4 Discussion

The *pipe and filters* software architecture fits well the sequential transformations steps (*transformation chain*) typically used in document engineering applications. Furthermore, the stateless property of the pipe and filter comply with the architecture of the web. However, as shown by SRM-IMMPS, the processing of multimedia documents sometimes needs to backtrack. This conflicts with the pipe and filters architecture which is unidirectional.

SRM-IMMPS is the synthesis of a number of projects concerning the automatic generation of multimedia documents. The reference model specifically aims at describing a vocabulary that identifies common understandings and allows comparison to specific systems, including Cuypers (see chapter 5).

However, the architecture of SRM-IMMPS is based on the assumption that the included media is automatically generated. Although this was a viable assumption at the time, since there existed no framework that provided incentive for a system to be optimized for reuse (the web was still relatively novel and only used on a small scale), this required significant computational resources. Furthermore, the authors of the SRM-IMMPS acknowledged the requirement of machine processable knowledge resources. As a solution they refer to a research initiative that was meant to support knowledge sharing (KQML).

The high requirements on computational resources and knowledge resources, prevented the development of automatic multimedia presentation systems on a large scale.

2.4 Summary

In section 2.1 this chapter we showed that current document engineering technology does not apply to multimedia documents. Furthermore, in section 2.2 we showed that in order for a machine to process multimedia data, a formal framework describing the media from different perspectives is required. Finally, in section 2.3 we showed that generating multimedia documents requires considerable investments in computational resources and knowledge encoding.

Based on these observations, in the next chapter we will derive requirements for an extended document engineering model that includes multimedia documents.

Chapter 3

Requirements

Based on the desirability of document engineering methods for multimedia documents (chapter 1) and the state of the art in related areas (chapter 2), we derive a first set of requirements for an extended document engineering model that supports multimedia documents. As described in section 2.3.1, a document engineering model is typically not independent of the environment in which it is applied. Therefore, in addition to requirements for the model, we also derive architectural requirements for the system that implements the model. These requirements have then been extended and refined based on the lessons learned during the development and evaluation of the software framework that is discussed later (chapters 5 and 6). The resulting set of refined requirements is presented in this chapter.

The first four sections focus on requirements for an extended document engineering model. The requirements in these sections are based on the traditional document engineering model, where essential differences compared to the traditional model are emphasized. In section 3.1 we focus on the requirements that define the relation between input, transformation and the produced output of a document engineering model. The subsequent sections focus on the vocabularies used to describe the individual components, transformation (section 3.2), input (section 3.3), and output (section 3.4).

In the fifth section we derive practical requirements for a system that implements the extended document engineering model and clarify the mutual dependency between the model and the architecture. Finally, in the last section we summarize our conclusions.

3.1 Document engineering principles

Recall that a form convention is a shared understanding between an independent author and reader about how a particular function can be communicated using a particular form. Form conventions are thus independent of a particular document, while they may be used to convey part of the document function. Document engineering exploits form conventions as a means to reuse authoring and design effort.

In this section, we first explicitly define the preliminary requirements that need to be satisfied to be able to apply a document engineering model. In the remainder of this thesis we assume the preliminary requirements are met.

3.1.1 Preliminary requirements

The document engineering model exploits form conventions as a document-independent abstraction to support adaptation to the delivery context and reuse of style (see section 1.1.1). We assume form conventions in multimedia documents can be exploited in a similar way. Thus, a preliminary requirement for a document engineering model that supports multimedia documents is that there exist sufficient form conventions to be exploited by the document engineering model.

Preliminary-Requirement a (FORM CONVENTION). *The author and reader of a document should independently agree on the function that may be represented by a form convention.*

If this requirement is satisfied, the document engineering model may be applied, which introduces the designer as a third independent stake holder. Since author, designer and reader fall outside the scope of the model, an additional preliminary requirement is necessary to ensure that three stakeholders, author, designer and reader share the objective to preserve document function as faithfully as possible.

Preliminary-Requirement b (STAKEHOLDER AGREEMENT). *Reader and author should both trust the designer to apply form conventions that convey to the reader the functions intended by the author.*

The author and reader of the document do not necessarily need to share the representation form conventions used to express function. For example, the presentation of date and time may be different between an American author and a European reader. This is, however, not a problem as a designer may, independently of the author, choose a representation that is most appropriate for the reader.

3.1.2 Reuse of authoring and design effort

The primary objective of the document engineering model is to reduce authoring and design effort (see section 1.1.1). Our first requirement therefore is that a document engineering model should support reuse of authoring and design effort.

Requirement 1 (SCHEMA DOCUMENT). *A document engineering model should support reuse of authoring and design effort.*

We have named this requirement “Schema document”, which may not be directly evident. However, in text-based document engineering this requirement can be (partly) formalized by a schema document. A schema document defines a class of structured documents. Likewise, a stylesheet defines the class of structured documents the stylesheet can be applied to. If both author and designer agree on the schema document, the stylesheet may be applied to transform the structured document and reuse of authoring and designing effort can be guaranteed (see section 2.1.3).

Document engineering reduces authoring and design effort by explicitly separating the two (see section 2.1.1). The results of a single authoring effort may be reused to produce multiple versions, adapted to a specific delivery context, of essentially the same document. Therefore, a document engineering model should be able to represent authoring effort explicitly, while abstracting from design effort. We refer to this representation as the structured document.

Requirement 2 (STRUCTURED DOCUMENT). *A document engineering model should have an explicit representation of document function that abstracts from the document form.*

The represented function in a structured document may be implicit or explicit. For example, in text-based documents, such as HTML and L^AT_EX, the formatting of a title is a form convention that is reusable (e.g. typically a title is positioned on top of the page and uses a relatively large font). The markup `<title>This is not a title</title>` indicates that the text “This is not a title” is, indeed, a title. In this example, the `title` tag makes part of the function explicit. In contrast, the function of the text in between the tags remains implicit, which is illustrated by the contradiction that is apparent to the reader, but not to the formatter.

Similar, but counter intuitive, are media items, which also represent function implicitly through form. For example, a figure may be part of the structured document, which is typically realized by including a reference to the file that contains the figure. However, the file is, typically, not accessed during the formatting, but only at the very last rendering stage when all the formatting decisions have been made (see section 2.1.3).

By explicitly separating design effort from authoring effort, the results of a single design effort may be reused to format multiple structured documents according to a consistent style. Therefore, a document engineering model should be able to represent design effort explicitly, while abstracting from authoring effort. We refer to this representation as the stylesheet.

Requirement 3 (STYLESHEET). *A document engineering model should explicitly represent the design effort necessary to transform an explicit representation of the document function to its corresponding document form.*

The class of structured documents defined by schema document and the class of documents that can be transformed by a stylesheet do not necessarily need to be identical. For example, the formatting of the title page of this thesis is formatted according to specific prescriptions of the university. The required form conventions for this page are not part of the more generic “book” stylesheet we use, and needed to be specified specifically for this thesis. The same holds for the associated structured document in which this thesis was authored, of which the original schema did not provide for functional elements to represent the content that needed to be conveyed on the title page. Although one can argue that the schema document should be extended to include the needs of a specific document, this would make the schema document and the associated stylesheet more detailed and therefore reduces the class of documents it can be applied to. Consequently, there is a trade-off between the specificity of a stylesheet’s transformation and the size of the class of structured documents that can be transformed by the stylesheet.

3.1.3 Implicit assumption: formatting satisfies constraints of delivery context

Although a textual document is typically spatially constrained by page or screen borders, the transformation of a structured document is commonly assumed to always succeed. The current document engineering model assumes an “overflow” strategy that guarantees the resolution of spatial constraints. Examples of such overflow strategies include the ability for paper-based documents to add an additional page if the remaining text does not fit the current page, whereas for electronic documents scroll-bars are often used as an overflow strategy. Due to such overflow strategies, adapting the document form to the spatial constraints imposed by the delivery context is for text-based document engineering considered a transformation detail that is to be resolved at the rendering stage, when all major decisions relevant to the document form are al-

ready made¹ (see section 2.1.3). Nevertheless a document engineering model should enforce that the constraints imposed by the delivery context are respected.

Requirement 4 (RESPECT CONSTRAINTS). *A document engineering model should enforce that the document form respects the constraints imposed by the delivery context.*

Some textual documents, such as newspaper articles or scientific publications, may also have page limits. Although a formatter can have strategies to limit the required space, for example, by adjusting the spacing, these strategies are not guaranteed to produce a document within the page constraints. In this case, the formatting fails (it produces a document that exceeds the page limit) and the key principle of document engineering, that adaptations to the form need not to involve changes to the function, breaks down. Manual intervention of the author or designer will be needed to resolve the conflicting constraint, for example, by rephrasing or deleting parts of the text in the structured document.

In order to respect the constraints imposed by the delivery context, they should be made explicitly available.

Requirement 5 (DELIVERY CONTEXT). *A document engineering model should have an explicit representation of the delivery context that constrains the document form.*

In addition to constraints on the spatio-temporal real estate, the delivery context might also constrain other properties of the document form, such as technical support for specific media types, network bandwidth or more user-oriented accessibility constraints.

3.2 Stylesheet vocabulary

A stylesheet represents the collection of form conventions that are used to transform a structured document to its corresponding document form. In this section we state the requirements on the vocabulary to represent form conventions in a stylesheet. This includes, besides requirements that mostly coincide with the traditional text-based model, additional requirements that are of specific importance for multimedia documents, notably, the transformation of media items and the resolving of constraint violations.

3.2.1 Representing form conventions

Reuse of authoring and design effort is based on the assumption that form conventions can be explicitly described. A first requirement is thus that the stylesheet vocabulary should be sufficiently expressive to explicitly represent form conventions. We refer to explicitly represented form conventions as *style rules*, the part of the document function that it conveys as a *functional construct*, and the part of document form that conveys the function as a *form construct* (cf. section 2.1.3). The part of the style rule that defines to which functional constructs it may be applied is referred to as the *selector* and the part of the rule that defines the resulting form construct is referred to as the *descriptor* of the rule.

¹Note that although all major design decisions have already been made, the remaining client-side transformation still includes detailed formatting issues such as pagination, hyphenation and kerning, which are important and involve relatively complex algorithms (see section 2.1.1).

Requirement 6 (STYLE RULE). *A stylesheet vocabulary should, for each of its style rules, be sufficiently expressive to (a) explicitly select the functional construct(s) the rule applies to, and to (b) explicitly describe the resulting form construct.*

The subsequent sections 3.3 and 3.4, elaborate on requirements for the vocabularies used to define functional constructs and form constructs.

A stylesheet, and thus a style rule, are, in principle, independent of a specific document. The function(s) to which a style rule could be applied are defined by a selector, which typically defines the conditions that should be met in order to apply the style rule. It thus abstracts from the details of a specific functional construct in a specific document. Similarly, the form represented by a style rule is a description of the form that conveys the selected function, but abstracts from the details of the specific form construction used in a specific presentation. For example, consider the CSS style rule `p {color: blue;}`. The selector of this style rule is sufficiently abstract to address all `p` elements (i.e. functional constructs) in a broad class of structured documents. The descriptor only specifies that they should be colored blue, abstracting from other details of the particular form construct used to convey the `p` elements.

For another example, consider the CSS style rule `p.quote {color: green;}`, which is more specific than the previous example as it addresses only the `p` elements that have a `class='quote'` attribute. This example illustrates that in CSS, style rules can have a different scope, selecting larger or smaller parts of the structured document. This is useful for a stylesheet designer because it allows a general rule to specify the bulk of the transformation, with exceptions that can be specified by more specific rules. A selector vocabulary should thus be sufficiently expressive to select any sub-part of the structured document. Because multiple style rules may match, a model needs to define how to unambiguously select which rule to apply and in what order.

Sub-Requirement 6.1 (EXPRESSIVE SELECTOR). *The selector vocabulary of a style rule should be sufficiently expressive to address any subset of functional constructs in a structured document.*

A CSS style rule basically adds formatting properties to the information described in the structured document, but does not alter the *structure* of the functional constructs as they are conveyed by the document form. The underlying assumption in CSS is that the structure of the structured document is also the most appropriate structure of the document form. This is, however, not always valid as form conventions sometimes influence the structure of a document. For example, dates in Northern America follow the pattern month-day-year, whereas in Western Europe they are structured day-month-year. In contrast to CSS, XSLT style rules allow adaptation of the structure of a form construct (section 2.1.3). For multimedia documents, quite often the structure of functional constructs does not correspond to the structure of the associated form construct. For example, an ordered sequence of images in the structured document can be presented using a temporal structure, showing one image after the other, or using a grid structure. In both cases, the structure of the form construct may differ from the structure of the underlying functional construct. Therefore, a style rule should be sufficiently expressive to define form constructs with structures that differ from the functional constructs they convey.

Sub-Requirement 6.2 (EXPRESSIVE DESCRIPTOR). *The descriptor vocabulary of a style rule should be sufficiently expressive to adapt the structure of a form construct.*

3.2.2 Implicit assumption: default style rule adapts form while preserving function

Typically, a stylesheet is understood as the collection of explicit style rules that formalizes form conventions, which may be adapted by a designer. However, it is important to note that in addition to these explicit style rules, there are also implicit style rules. These transform the parts of a structured document of which the function is not made explicit, such as media items and plain text. For example, the text in an HTML document that does not contain mark-up, represents function implicitly. A generic, default style rule that is not part of an explicit stylesheet, will typically transform the text in the structured document to the text in the document form.

Since these style rules are independent of the function they represent, they are generic and can also be applied independently of the stylesheet that is used. Therefore, they are typically built into the formatter and therefore hidden to author and designer. Nevertheless, a document engineering model should, besides style rules that transform the explicitly represented document function in a structured document, have support for style rules that transform the implicit document function in a structured document.

Requirement 7 (DEFAULT STYLE RULES). *The function that is implicitly represented in a structured document (i.e. media items) should be generically transformed to form while preserving the function.*

The requirement allows the form of a media item to be adapted during the transformation. A typical textual example of this is the typesetting of plain text which may apply hyphenation, line and page-break algorithms to adapt the resulting form. While these algorithms may be complex, they should not, and typically do not, change the function of the text. Indeed, in text, cases where the formatting alters the meaning of the text are quite rare. Consider the sentence: “The merchant liked the agora”. This sentence is ambiguous since agora can either mean “a small coin” or “a marketplace”. However, the hyphenation of both word senses is different: ago-ra and ag-o-ra, respectively. If the word agora needs to be hyphenated, the formatter needs to know which sense of agora the author intended in order to apply the correct hyphenation. This information is, in general, not available and therefore the formatter can make a mistake in preserving the function.

In multimedia documents, the risk that similar changes to the form unintentionally change the underlying function is much larger. Even superficial operations, such as the scaling of figures, cannot be considered a generally applicable transformation strategy. For example, the correct perception of the scaled image may depend on the portrayed content (e.g. holiday photographs might require less detail while scaled down x-ray scans may become useless).

3.2.3 Implicit assumption: formatting always succeeds

Implicitly, the traditional document engineering model assumes that documents have a generic overflow strategy, which ensures the transformation does not fail (see section 3.1.3).

Formatting of multimedia documents cannot be based on such simple and generic overflow strategies. The positioning and alignment of a media item, in relation to other media items, may carry significant semantics. Furthermore, the combination of media items often conveys a particular relation that would not be conveyed if the media items would be presented separately [48, 49, 93, 110]. For example, the Russian filmmaker Lev Kuleshov demonstrated that an audience may interpret a shot differently if the shot showed prior to it is different².

²The experiment showed an audience a neutral face of a man. The shot prior to the face showed either a

Consequently, the flexibility to adapt the form for multimedia documents is reduced, which increases the chances of a formatting failure.

Requirement RESPECT CONSTRAINTS (#4/p.60) states that the document form should respect the constraints imposed by the delivery context. Since violation of these cannot be solved by simple overflow strategies without changing the function being conveyed, the formatting of a multimedia document may fail. An extended document engineering model should, therefore, detect such formatting failures and allow the specification of resolution strategies to resolve them.

Requirement 8 (ALTERNATIVE FORMATTING). *The transformation from function to form should detect failure and be able to apply an alternative transformation that preserves function while violating no constraints.*

The traditional text-based document engineering model does not support the possibility to detect formatting failures and specify an alternative resolution strategy. Therefore, currently available document engineering technology is not suited to implement the document engineering paradigm for multimedia documents (see section 2.1.4).

3.3 Structured document vocabulary

In this section we derive requirements for the vocabulary used to explicitly represent document function. In addition to requirements that largely coincide with the text-based model, these also include additional requirements as a consequence of the ALTERNATIVE FORMATTING (#8/p.63) requirement, which is absent in the text-based model.

A structured document represents part of the document function explicitly, with the intent to reuse form conventions that are described in a stylesheet. The vocabulary used to specify the structured document should therefore be sufficiently expressive to explicitly represent the function represented by a form convention.

Requirement 9 (FUNCTIONAL VOCABULARY). *The vocabulary used to describe a structured document should be sufficiently expressive to explicitly denote the function represented by a form convention.*

Note that requirement FORM CONVENTION (#a/p.58) refers to the conceptual agreement between author and designer on the function represented by a form convention, whereas this requirement (FUNCTIONAL VOCABULARY (#9/p.63)) refers to the vocabulary used to represent the function of form convention.

In multimedia documents and text-based documents grouping is typically used to convey the hierarchical/rhetorical structure of the document. While the layout and style of these groups is determined by the style sheet, the functional constructs should provide the author with the vocabulary to define the underlying grouping relations.

Sub-Requirement 9.1 (EXPRESS GROUPING). *The vocabulary used to specify the structured document should be able to express grouping of functional constructs.*

girl, a bowl of soup or a coffin. Depending on the shot showed prior to the face the audience interpreted the emotional expression of the man as, respectively, desire, hunger and grief. This effect is known as the *Kuleshov effect*.

For example, in text-based documents, paragraphs group units of text that semantically belong together. Likewise, paragraphs may be grouped into sections and sections into chapters. Similarly, in multimedia documents a group of functional constructs can be semantically related to another group of functional constructs. In figure 3.1 on page 66, for example, the groups of the painting with caption (5,6) and the group formed by the explanatory text with title (2,3) are related because the painting illustrates the concept of genre painting discussed in the text. Finally, the title above the multimedia document (1) relates to all media items in the document. Again, the layout details that determine how these relations are conveyed to the user are to be made explicit in the style sheet. On the document level, functional constructs are needed to express the hierarchical relations described above.

Authors of both text-based documents, as well as multimedia documents, structure the discourse of a document according to the premises that a reader “understands” the part of the document that she read so far, and is unaware of the function of the remaining part³. Text-based documents as well as multimedia documents typically have linear or at least partially ordered discourse structures. Subsequently, the vocabulary used to express the structured document should be able to express the order of functional constructs.

Sub-Requirement 9.2 (EXPRESS ORDER). *The vocabulary used to denote the structured document should be able to express order between functional constructs.*

The ALTERNATIVE FORMATTING (#8/p.63) requirement states that a formatter should try alternative formatting in the case of a formatting failure. Compared to the formatting algorithms available for text, the ability to adapt the form of a media item while preserving the intended function is relatively limited. To compensate for formatting failures, it is therefore sometimes necessary to suppress part of the structured document. The represented function will thus be altered by omitting part of the structured document, and the distortion compared to the original represented function should be minimized. Therefore the relative importance of a functional construct to the document function should be made explicit.

Sub-Requirement 9.3 (EXPRESS PRIORITY). *The vocabulary used to denote functional constructs should be able to express the relative importance of the functional construct with regard to the document function.*

Priorities are less apparent in text-based documents. However, for some constrained documents, for example newspaper articles, writing styles exist that consider priorities. Newspaper articles are typically written in “inverted pyramid” style. This ensures that the most important part of the function is presented first. In this way an article can be cut at the end of any paragraph while ensuring that the most important part of the article has been conveyed. Note that priority is not the same as order. For example, in a document presenting the paintings created by an artist, the paintings can be ordered chronologically by date of creation. This order most likely does not correspond to the order of the most significant works of the artist.

3.4 Form vocabulary

A document engineering model generates the document form based on a specification in the stylesheet (and a structured document). The form vocabulary should therefore be sufficiently expressive to specify the form that conveys the function associated with a form convention.

³Hyperlinks provide multiple ways to traverse a document. It is, however, the responsibility of the author that the discourse can be interpreted as a coherent structure.

In addition, a form vocabulary allows a designer to represent the form of a document while abstracting from the irrelevant procedural knowledge necessary for constructing the document form. This may include command codes that are sent to a printer, or the rendering instructions that activate the pixels of a screen.

A form vocabulary should thus be sufficiently expressive to allow a designer to represent the document function through form. However, it should abstract from irrelevant procedural details to produce the document form.

Requirement 10 (FORM VOCABULARY). *The vocabulary used to describe form should, on the one hand, be sufficiently expressive to describe form constructs that convey the intended function. On the other hand, the vocabulary should abstract over irrelevant details of the production of the document form.*

Examples of form vocabularies for text-based documents include XSL-FO and T_EX (see section 2.1.3). Similarly, the Amsterdam Hypermedia Model (AHM) is a form vocabulary for multimedia documents (see section 2.1.2).

The author of a text-based structured document is typically unaware of the form vocabulary, which is hidden in a stylesheet, and focuses on the authoring of the structured document. In contrast, the author of a multimedia document typically uses the form vocabulary as the authoring vocabulary (e.g. SMIL). We consider this situation as illustrative for the absence of document engineering technology for multimedia documents. The more detailed requirements on form vocabulary in the remainder of this section are derived from the AHM. However, since we exclude support for authoring convenience, the requirements presented here are simpler than those specified by the AHM. Furthermore, the AHM is not concerned with potential formatting failures. In order to satisfy requirement RESPECT CONSTRAINTS (#4/p.60) we therefore specify additional requirements on the form vocabulary that are absent in the AHM.

3.4.1 Representing form

Figure 3.1 presents a screen shot of a multimedia document about the concept “genre painting” in the work of the painter “Vermeer”. The image of “The Kitchen Maid” (6), is used by the multimedia author as one of multiple illustrative examples of this concept, which are presented in sequence. We use this figure to illustrate the required expressive ability to represent function through form.

Recall that part of the document function is represented by media items in a structured document. For example, the author of the Vermeer presentation portrayed in figure 3.1 selected an image of the painting “The Kitchen Maid” to represent part of the function of the multimedia document. The image (6) in figure 3.1 conveys the function represented by the originally selected image. Although the media items may look identical, the image in the document form is actually the result of a transformation that scaled down the image.

A media item in the structured document is represented by a, possibly transformed, media item in the document form. The form vocabulary should therefore be sufficiently expressive to represent a transformed media item.

Sub-Requirement 10.1 (MEDIA ITEM). *The vocabulary used to describe document form should be sufficiently expressive to represent media items.*

The caption text, (5) in figure 3.1, is associated with the image (6). This is conveyed by the position of the caption in relation to the position of the image. In addition to the positioning of



Figure 3.1: Screen shot of a multimedia document about genre paintings and Johannes Vermeer. (Numbers in brackets are used for future reference.)

media items, layout is also used to specify composite regions in a multimedia document. For example, the image (6), its caption (5) and the navigational arrows (7 and 8) belong together, which is expressed by defining a region that “contains” these media items. These examples illustrate that the layout of a multimedia document can convey part of the document function. Therefore, the form vocabulary should be able to specify the layout in sufficient detail that all layout decisions that may impact the form have been made during the transformation phase, and need not be deferred to the rendering phase.

Sub-Requirement 10.2 (LAYOUT). *The vocabulary used to describe form should be sufficiently expressive to describe the layout of a multimedia document.*

The title (2) and the paragraph (3) are both textual media items. They have a different function though, which is indicated by the use of a larger font for the title. This is an example of how the style of a multimedia document may convey part of the document function. Therefore, the form vocabulary should be able to represent styling properties.

Sub-Requirement 10.3 (STYLE). *The vocabulary used to describe document form should be sufficiently expressive to denote styling properties.*

Hyperlinks may also influence the reader’s perception of a document, and may even be used as a means to convey document function. For example, the hyperlinks attached to the media items (7) and (8) in figure 3.1 may be used to proceed to the next example painting, or go back

to the previous one, which conveys the chronological ordering between the example paintings. Therefore, the form vocabulary should be sufficiently expressive to represent hyperlinks.

Sub-Requirement 10.4 (HYPERLINKS). *The vocabulary used to describe document form should be sufficiently expressive to denote hyperlinks.*

3.4.2 Form properties to detect constraint violations

As stated by requirement RESPECT CONSTRAINTS (#4/p.60) the form of a multimedia document should respect the constraints imposed by the delivery context. In order to detect constraint violations the formatter needs to be aware of the properties in the document form that may cause constraint violations. Therefore these properties should be made explicit.

Requirement 11 (FORM CONSTRUCT PROPERTIES). *The properties in the document form that may cause a constraint violation should be explicitly described.*

Typically the properties that may cause constraint violations refer to the spatio-temporal resources used in the document form. However, the delivery context may constrain other properties, such as available bandwidth or color capabilities. In order to respect such constraints, the corresponding properties in the document form should be made explicit.

3.5 Practical requirements

The requirements for an extended document engineering model we derived in the previous sections abstract from the formatter that will implement the model. The document engineering model that fulfills these requirements is, therefore, independent of the technology used to implement the formatter, which is an important property of a document engineering model. However, there are practical concerns that we should take into account for a successful document engineering system. These include requirements to optimize resource reuse and the ability to use the formatter in a web context.

3.5.1 Optimize for reuse

In the previous chapter we discussed SRM-IMMPS, which is a reference model for systems that automatically generate multimedia documents. As mentioned, a practical concern of these systems was the extensive amount of explicit knowledge necessary to automatically generate a multimedia document. Typically this requires significant investment in human resources, which is only viable in very specific contexts. The SRM-IMMPS acknowledges this and proposes the use of an open architecture to address the extensive knowledge requirements. Although the objectives of the systems in SRM-IMMPS are more ambitious, the knowledge requirements are typically higher compared to traditional document engineering systems (see section 1.1.2). In order to make document engineering viable for general use cases, the required knowledge investment should be decreased by reusing and preserving the available knowledge and media items. Note that reuse cannot be enforced as a requirement as it is dependent on the commitment of the user. Instead, we make a requirement that optimizes the model to support reuse.

Requirement 12 (OPTIMIZED FOR REUSE). *Metadata made available for the transformation should be optimized in such a way that it may be reused for future transformations (and potentially other purposes). In addition, the model should be sufficiently flexible to reuse existing metadata that was originally created for other purposes.*

Although this is a practical requirement, it still affects the way the model needs to deal with metadata in the associated vocabularies. For example, if only the resulting document form is sent to the client, this document form needs to include the metadata in order to allow other applications to make use of it. This is in contrast to the traditional approach, where multimedia presentation formats contain little or no metadata.

3.5.2 Web compliant

The web is a medium many people have access to. Since their delivery contexts are highly heterogeneous, document engineering is especially desirable in a web context. Therefore, a document engineering system should be applicable on the web and comply to the (REST) architecture of the web (see section 2.3.1).

Requirement 13 (WORLD WIDE WEB). *The architecture of an extended document engineering system should comply with the Representational State Transfer (REST) architecture used for the World Wide Web.*

Note that this is a requirement on the architecture of an extended document engineering system, whereas the previous requirements were on the model of a document engineering system.

From a document engineering perspective, the web physically separates the author's (i.e. server) structured document from the document form the reader (i.e. client) perceives. The transformation from a structured document to its corresponding document form is therefore necessarily split between an independent server and client. Therefore, the communicated data structures between client and server should conform to a common standard that is shared by both server and client.

Typically, for text-based documents on the web the structured document (e.g. HTML) and the stylesheet (e.g. CSS) are standardized. A client-browser requests the structured document and stylesheet from a server and generates the corresponding document form in an internal representation format. This architecture has the advantage that the transformation, which is a computationally intensive step, is performed client-side. Consequently, the load on the server is reduced, which is desirable in a centralized server architecture, such as the web.

However, for multimedia documents this architecture is practically infeasible because the existing standardized vocabularies for structured documents and stylesheets do not account for potential formatting failures. In order to adopt a similar client-side formatting architecture for multimedia documents, standardized vocabularies for structured document and stylesheet should be available. Since we do not expect this to be a realistic assumption in the near future, we assume that multimedia documents are formatted server-side.

However, this assumption has consequences for the architecture of a document engineering system that includes multimedia documents on the web. Since the stylesheet is processed server-side it does not need to be standardized in a server-side transformation architecture. However, in this case, the delivery context, which defines the constraints of the presentation environment, should be sent to the server. Since client and server are independent the delivery context should be communicated to the server in a commonly accepted format.

Sub-Requirement 13.1 (DELIVERY CONTEXT FORMAT). *The delivery context should be communicated to the server in a commonly accepted format.*

Although few client applications currently sent the properties of the delivery context to the server, there are working groups in W3C, notably the Ubiquitous Web Applications (UWA) [72], that actively develop recommendations to describe the delivery context, such as Delivery Context Client Interfaces (DCCI) [158] and CC/PP [90].

Besides the delivery context, the document form should also be represented in commonly accepted presentation format.

Sub-Requirement 13.2 (PRESENTATION FORMAT). *The representation of the document form produced by the server should conform to a commonly accepted presentation format.*

There are currently a number of standardized presentation formats available to describe a document on the web, including (X)HTML, SMIL and SVG⁴. Besides the “official standards” there are also proprietary standards such as Flash [100] that are commonly used.

3.6 Conclusion

In this chapter we derived requirements for an extended document engineering model. These include requirements derived from the traditional document engineering model. However, the traditional model assumes generally applicable overflow strategies, which is not the case for multimedia documents. Therefore, the formatting of multimedia documents may, in contrast to text-based documents, fail. An extended document engineering model should thus detect constraint violations and propose alternative formatting when necessary.

The knowledge necessary for detecting and resolving constraint violations should be made explicit in an extended document engineering model. This includes explicit knowledge on the properties of the delivery context and form constructs that are relevant for detecting constraint violations. As a result, the knowledge requirements in an extended document engineering model are significantly larger compared to traditional document engineering. To reduce the associated costs, an extended document engineering model should support reuse and preserve existing knowledge where possible.

Since document engineering is especially desirable in a web environment, we derived practical requirements for a system that implements the extended document engineering model in a web environment.

In the next chapter, we will discuss a document engineering model that is based on the requirements presented in this chapter. As in this chapter, we will use the traditional model as the basis, and emphasize the additions and changes that are needed to support multimedia documents, such as the handling of media items, the explicit representation of the constraints imposed by the delivery context, and the explicit representation of alternative style rules to handle situations where the primary formatting rules would result in a violation of these constraints.

⁴The World Wide Web Consortium (W3C) refers to the standards developed within the W3C as “recommendations”. This is to emphasize the open character of the web.

Chapter 4

A document engineering model for multimedia documents

The document engineering paradigm separates authoring effort from design effort with the intent to reuse parts of the authoring and design process and consequently reduce the cost of producing documents. Based on the requirements derived in the previous chapter, this chapter presents a model that extends the traditional document engineering model to make it applicable to multimedia documents.

The model presented here abstracts from the technical details of the software architecture implementing the model. This allows the technology used to implement the model to be optimized for a particular architecture. Furthermore, abstracting from technical details allows comparison to related document engineering models, which we use to draw parallels and clarify significant differences. In the following chapters we use the model as a reference to illustrate and clarify the implementation choices we made to implement and evaluate our document engineering framework.

We first model the document engineering paradigm, which is described in section 4.1. The subsequent three sections model the vocabularies used to represent design effort (section 4.2), the authoring effort (section 4.3), and document form as perceived by the reader (section 4.4). We conclude in section 4.5 by discussing the pros and cons of our model.

4.1 Modeling the document engineering paradigm

The requirements we derived in the previous chapter largely correspond to the requirements for a traditional document engineering model (see section 3.1). Therefore we base our extended model on the traditional document engineering model.

The main top-level concepts of our model are shown in figure 4.1¹.

¹The diagram presented in figure 4.1 is the first of a number of diagrams we use in this chapter to illustrate our model. The diagrams are expressed using UML class diagrams notation [116]. The boxes (classes in UML), such as document form, represent concepts in the real world. The name of a concept is presented on top, using a bold font (italics means that the concept is abstract and has no instantiations). Within the concept, attributes and operations are expressed that define the concept. Relationships between concepts are expressed by drawing a line between them. We use four types of lines in our diagrams. A plain line denoting a relationship

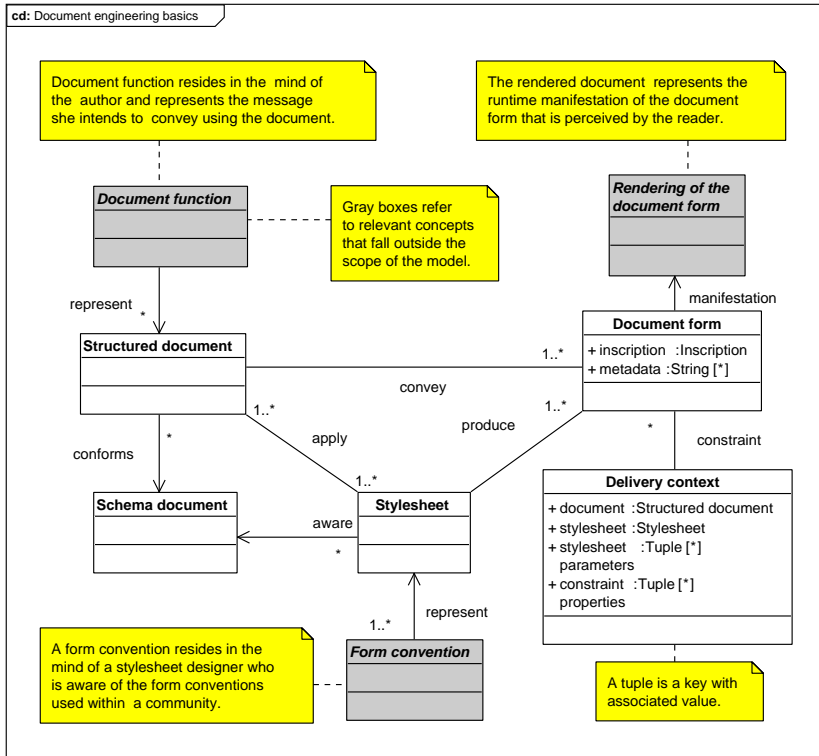


Figure 4.1: The document engineering paradigm separates authoring effort, represented by a *structured document*, from design effort, represented by a *stylesheet*. Based on a structured document and a stylesheet, the *document form*, which represents the document perceived by the reader, is automatically adapted to the *delivery context* of the reader.

Analogously to the traditional document engineering model, we represent authoring effort (i.e. document function) in a *structured document* (requirement STRUCTURED DOCUMENT (#2/p.58)) and design effort (i.e. form conventions) in a *stylesheet* (requirement STYLESHEET (#3/p.59)). A stylesheet is used to transform a structured document to its corresponding *document form*, which, after rendering, is presented to the reader. Note that the relationships, *convey*, *apply* and *produce* in figure 4.1 are relationships that do not need to be made explicit *a priori*. This allows multiple stylesheets to transform a structured document, and multiple structured documents can be transformed by a single stylesheet (indicated in figure 4.1 by UML multiplicity

between two concepts (association in UML). A line with a diamond on one side denotes a part-of relationship. If the diamond is closed it means one concept only exists if the other concept exists (composition in UML). If the diamond is open it means both concepts can live independently (aggregation in UML). All relationships, except is-a, have associated multiplicity. The default is 1, * denotes zero or more, 1..* denotes one or more. The dog-eared boxes denote informal comments.

constraints). However, for reusing design effort the prerequisite is that the authored structured document is a member of the class of structured documents that may be transformed by a particular stylesheet. Similarly, for reusing authoring effort the stylesheet should be a member of the class of stylesheets that may be used to transform a particular structured document. We represent this agreement between author and designer in our model by a *schema document* (requirement SCHEMA DOCUMENT (#1/p.58)). However, similarly to the traditional model, the schema document does not necessarily need to be made explicit.

While there are similarities between the traditional model and the extended document engineering model, there are also important differences. These differences we describe in more detail in the remainder of this section. Notably, in contrast to the traditional model, we explicitly represent the constraints imposed by the delivery context, whereas the traditional model abstracts from constraint resolution. Additionally, the traditional document engineering model ignores the role of metadata associated with function and form, whereas we model metadata explicitly.

4.1.1 Scope of the model

The gray areas in figure 4.1 specify relevant concepts that intentionally fall outside the scope of the model, but are nevertheless important to define the borders of the model. At the top left, the *document function* represents the message an author intends to convey as it resides in the mind of the author. To the right, the *rendering of the document form* represents the manifestation of the document form as the reader perceives it. In the lower middle part of the figure, the *form conventions* represent the conventions (shared within a community of people) that enable communication. Although these concepts fall outside the scope of the model they denote the extreme states in the process of communicating a message from author to reader. Intuitively, the document engineering process “happens” in between those extreme states.

Both our and the traditional model describe an abstract, single transformation, while in practice document formatting typically involves multiple transformations, implemented by a transformation chain (see chapter 2, page 29). We will, however, discuss transformation chains extensively in the following chapters.

4.1.2 Explicit modeling of delivery context

The *document form* in figure 4.1 models the entire artifact that is perceived by the reader. We follow the distinction between *medium* and *inscription* discussed in chapter 2 (see page 30). In our model, the inscription will represent the result of the document transformation, and will be discussed in more detail later.

The medium has an important influence on how the document function needs to be conveyed by the document form. The document form needs to be able to be perceived on the specific hardware the reader uses, and typically needs to match various constraints related to screen size, color capabilities of the device and the bandwidth of the connection. Further constraints may be related to the specific requirements of the reader, and may address her language preferences, accessibility constraints or level of expertise.

We model this set of potential constraints as the *delivery context*². In the traditional document engineering model, the delivery context is not made explicit. There, the delivery context is considered to only become relevant in the last transformation, from *document form* to *rendering*

²Our use of the term delivery context is in line with the use of the same term by the W3C Device Independence Working Group [153].

of the document form. During this transformation, text-flow algorithms, such as hyphenation and kerning, calculate the position of each glyph with respect to the delivery context (i.e. medium). However, these algorithms do not significantly alter the document form and therefore they are considered to be outside the scope of the traditional model. In contrast, our requirement DELIVERY CONTEXT (#5/p.60) states that the delivery context in an extended document engineering model should be made explicit as a formatter for multimedia documents needs to check whether the document form satisfies the constraints imposed by the delivery context. We thus model the delivery context explicitly, as shown in figure 4.1. The delivery context's *constraint properties* refer to a list of properties that are directly relevant to determine potential constraint violations. This list may include properties that describe the capabilities and limitations of a device, or properties that describe user preferences that are relevant for accessibility constraints.

4.1.3 Explicit parametrization of the style sheet

In contrast to the structured document and the stylesheet, the delivery context models the reader's influence on the document transformation. The relationships *apply*, *convey* and *produce*, of which further details are intentionally undefined, are only instantiated once the delivery context is known. We model the instantiations of these relationships by the properties *structured document* and *stylesheet* in the delivery context in figure 4.1. One can argue that the designers should develop a stylesheet for every potential delivery context, and the reader then only needs to specify the structured document and the stylesheet that fit her delivery context. This is indeed the approach taken by HTML and CSS. The document form is adapted on the basis of a stylesheet that the reader selects. Typically these stylesheets are provided by the author, although a reader can provide her own. However, there are serious disadvantages with this approach when it is applied to multimedia documents. Since many properties of the delivery context may impact the document form, each valid combination of these properties could require a different stylesheet. The number of required stylesheets could easily grow exponentially with the relevant properties of the delivery context the form should adapt to.

Therefore, more advanced text-based document engineering technology provides mechanisms to parametrize the style sheet, where the values of these parameters represent important information from the delivery context. Based on these parameters the stylesheet then adapts the transformation process dynamically. Similarly, we model the parametrization of the stylesheet by including *parameters* in the delivery context. In contrast to constraint properties, which will eventually need to be standardized, the set of supported parameters are determined during authoring time by the author and designer, and the reader and her delivery context may influence the values of these parameters in various ways at run time.

4.1.4 Explicit modeling of metadata

In order to reduce the extensive knowledge requirement inherent to multimedia data and to make the document engineering paradigm viable to multimedia documents, reuse of knowledge is essential, as stated by requirement OPTIMIZED FOR REUSE (#12/p.67). The extended document engineering model therefore explicitly models metadata in the document form (See figure 4.1). Because metadata may be dynamically and incrementally added during the transformation process, it is not only present in the resulting document form, but occurs in multiple places in the model.

Note that although our model explicitly includes metadata, in practical implementations the propagation of metadata still depends on the properties of the final format document.

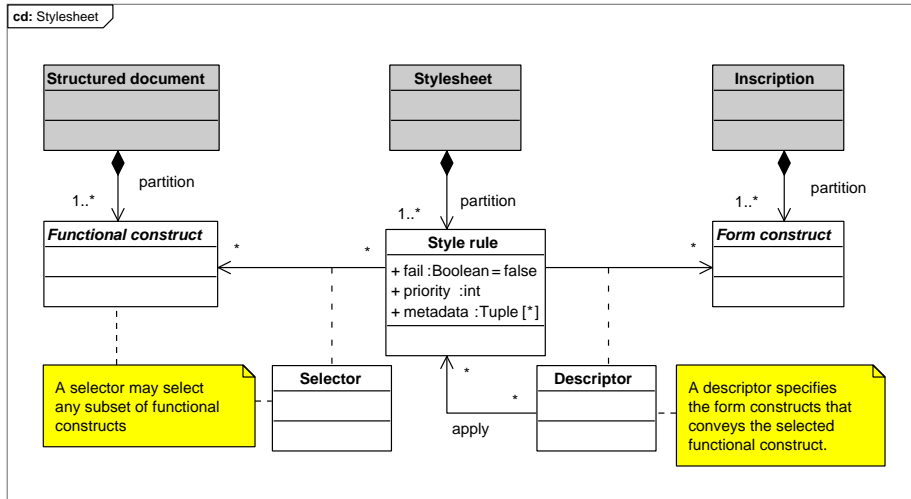


Figure 4.2: A stylesheet is a collection of *style rules* that describe form conventions. A style rule specifies a *selector*, which selects *functional constructs*, and a *descriptor*, which specifies the *form construct* used to represent the selected functional construct. Greyed class boxes denote classes that were previously introduced.

4.2 Modeling the stylesheet

Analogously to the traditional document engineering model, the result of the design effort is formalized by a stylesheet. In this section we model the vocabulary used to represent the stylesheet.

Requirement STYLE RULE (#6/p.61) states that a stylesheet vocabulary should be sufficiently expressive to represent form conventions. Similarly to the traditional model we represent form conventions by *style rules*. A style rule selects zero or more functional constructs from a structured document and produces the corresponding form construct, which is part of the *inscription* of the document form. This is illustrated in figure 4.2 by a *selector*, which is associated with the relationship between a style rule and a functional construct³.

Since a style rule may be independent of a structured document, its selector should be sufficiently expressive to select any subset of functional constructs from a structured document as stated by requirement EXPRESSIVE SELECTOR (#6.1/p.61). An expressive selector vocabulary is especially important for the reuse of style rules as it allows a designer to formulate the bulk of the transformation in a style rule with a broad scope, while the formatting exceptions may be

³In UML, properties of a relation may be represented by an associated class, indicated by a dashed line between the associated class and the relation. Note that the arrow is directed from 'Style rule' to 'Functional construct'. This denotes that, provided that a relationship exists, an instance of a 'Style rule' has access to the public properties of an instance of a 'Functional construct', whereas the reverse is not possible. As a result, the 'Selector' is typically implemented as part of the 'Style rule'.

expressed by a style rule with a specific scope⁴. In our model we abstract from the specifics of a selector vocabulary and the resolution strategy that selects the style rule that is applied. This allows the model to be used for stylesheet vocabularies with different expressive properties, such as CSS and XSL(T) for text-based documents.

A style rule produces zero or more form constructs that represent the selected functional constructs. The specification of the produced form constructs is specified by a *descriptor* as represented in figure 4.2. Similar to the selector of a style rule, the descriptor may be independent of a structured document. Again, we abstract from the details of the descriptor vocabulary, and only state it should be sufficiently expressive to describe the inscription to be generated (per requirement EXPRESSIVE DESCRIPTOR (#6.2/p.61)). Finally, a designer may associate metadata with a style rule. As in the traditional model, the metadata may be used to control the application of style rules (e.g. “named templates” in XSL(T)), but the metadata associated with a style rule may also be used to annotate the inscription for further processing.

In contrast to the traditional model, a transformation in the extended document engineering model may fail, as indicated by the Boolean *fail* property, which indicates whether a style rule failed to successfully complete its transformation. In the remainder of this section we further elaborate on the differences compared to the traditional model. These include multiple default styles to transform multiple media types and the detection and resolution of potential formatting failures. Furthermore, we discuss the pros and cons of modeling soft constraints in addition to hard constraints.

4.2.1 Multiple default style rules

Recall that a structured document only partially abstracts from the document form, but not completely (see section 4.1). Consequently, the part of the function that remains explicit in a structured document is expressed only through form. For example, in the traditional document engineering model, the function of a figure in a structured document often remains implicit and is expressed only through its form. Note that the corresponding figure in the document form need not be identical to the figure in the structured document, as it may have been transformed (e.g. through scaling or rotation, color mapping etc.).

For our extended model, part of the document function is expressed through media items. Typically, the function of these media items is at least partially implicit to the formatter. As a result, a media item can be included as functional construct although it represents form. As with figures, a media item representing a form construct may appear differently compared to its corresponding media item representing a functional construct because of the application of style attributes, such as scaling or cropping.

In figure 4.2 we define that a stylesheet has at least one style rule. This corresponds to requirement DEFAULT STYLE RULES (#7/p.62), which states that there should be a default style rule that defines the transformation if no other style rule applies. For text-based documents this is typically the generic style rule that applies text formatting algorithms, such as pagination, hyphenation and kerning. However, since media items in a multimedia document may use inherently different modalities (e.g. image, video, audio), a single transformation rule for multimedia documents is insufficient. Therefore, a formatter that implements the extended document engineering model requires more than one default style rule. In contrast to the traditional text-based model, the modality of the media item thus needs to be made explicit (see section 4.3).

⁴UML lacks a means of formally representing such constraints. Instead they may be informally represented within the class diagram by comment notes, as we do in figure 4.2.

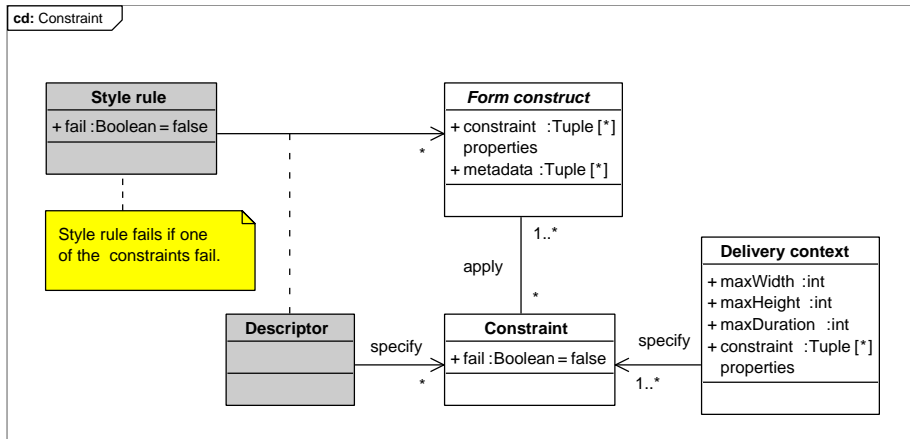


Figure 4.3: Style rules impose *constraints* on *form constructs* to convey document function whereas the *delivery context* imposes constraints to ensure a document can be played within the environment of the reader.

4.2.2 Detecting constraint violations

The RESPECT CONSTRAINTS (#4/p.60) requirement explicitly states that an extended document engineering model should respect the constraints imposed by the delivery context. Although this requirement is often implicitly assumed in the traditional model, there is no generic resolution strategy in the case of a constraint violation. Therefore, in the extended model it should be possible to detect constraint violations and provide alternative formatting as stated by requirement ALTERNATIVE FORMATTING (#8/p.63). Figure 4.3 illustrates the explicit specification of constraints on the formatting of a form construct. A *constraint* specifies conditions that the formatter needs to respect in order to successfully format the document form. Constraints are specified by the *delivery context* and the *descriptor* of a style rule.

The constraints imposed by the delivery context ensure that the document can be presented in the environment of the reader. These minimally include the three spatio-temporal resource constraints, as indicated by *maxWidth*, *maxHeight* and *maxDuration*, but may also include other types of constraints such as user or device constraints. Note that these resources are maximum values, typically imposed by the constraints of the hardware, the actual dimensions of the form constructs may be smaller.

Since the type of constraints used may vary between applications, we abstract from exhaustively defining the properties that may be constrained. Instead, we refer to them as general *constraint properties* that may be associated with the delivery context and a form construct, as indicated in figure 4.3. Note that the constraints imposed by the delivery context are known before the structured document is transformed. Therefore, every form construct is constrained *a priori* by the delivery context.

The constraints imposed by the descriptor of the style rule ensure that the function repre-

sented by a functional construct is conveyed by the associated form construct. If a constraint fails, which is indicated by the *fail* property, this implies that the style rule that imposed the constraint also fails and an alternative style rule should be invoked.

4.2.3 Selecting alternative style rules

Requirement ALTERNATIVE FORMATTING (#8/p.63) states that if a style rule fails to transform a functional construct, then the constraint resolution strategy of the formatter should invoke an alternative style rule. While this is not supported by the traditional document engineering model, most implementations (including CSS and XSLT) support the formulation of multiple style rules where one can override the other. We represent alternative style rules in the extended model in a similar fashion. However, the strategy that selects the style rule to execute is more elaborate in the extended document engineering model. Recall that the scope of a style rule may vary. As a result, there may be multiple candidates to transform a form construct. In most style sheet applications, the rule with the most specific selector is executed. In our model, the strategy that selects the style rule to execute also includes the selection of an alternative style rule if a prior one failed. Note that the exact strategy to select alternative style rules may depend on the application, and is typically influenced by multiple factors, including, aesthetics and performance. In our model we abstract from defining such a strategy. Nevertheless, the order in which style rules are applied should be known to the formatter implementing the model. Therefore, we explicitly denote the order in which style rules are applied by the *priority* property as indicated in figure 4.2. Based on the priority order, the formatter selects one of the potential style rules. If this rule happens to fail, the next potential rule is invoked until a transformation succeeds. Note that the successful execution of a style rule cannot be guaranteed, as a potential formatting failure cannot be completely avoided. However, in practice such a failure can often be prevented by generating a low key presentation reporting the inability to transform the document for the particular delivery context.

4.2.4 Discussion: soft constraints

A constraint in our extended document engineering model either succeeds, or fails. This suffices for spatio-temporal resource constraints, such as width, height and duration. However, a designer may additionally want to express constraints that should be respected if possible, but may fail if they compromise a successful transformation. We refer to them as *soft constraints*. Typically, soft constraints address the aesthetic quality of the document form. For example, in text-based document engineering, the distribution of white-space influences the perceived aesthetic quality and clarity of a document. Likewise, the first sentence of a paragraph presented as the last sentence on a page (i.e. an orphan) is considered bad typography and should be prevented, if possible. Most text-based formatters attempt to format the document in such a way that the soft constraints are respected whenever possible. Note that soft constraints may be (partially) contradictory. In such cases the formatter attempts to find a compromise. In \TeX , for example, the compromise is expressed through a “badness” value, which indicates the distance from a theoretical optimally formatted document.

For multimedia documents, similar soft constraints that optimize the aesthetic quality of the document apply. Consider, for example, constraints that enforce an aesthetic balance of the form construct on a page. Or, a constraint that enforces a harmonized color scheme for the form constructs used in the multimedia document. However, due to the absence of a generic overflow

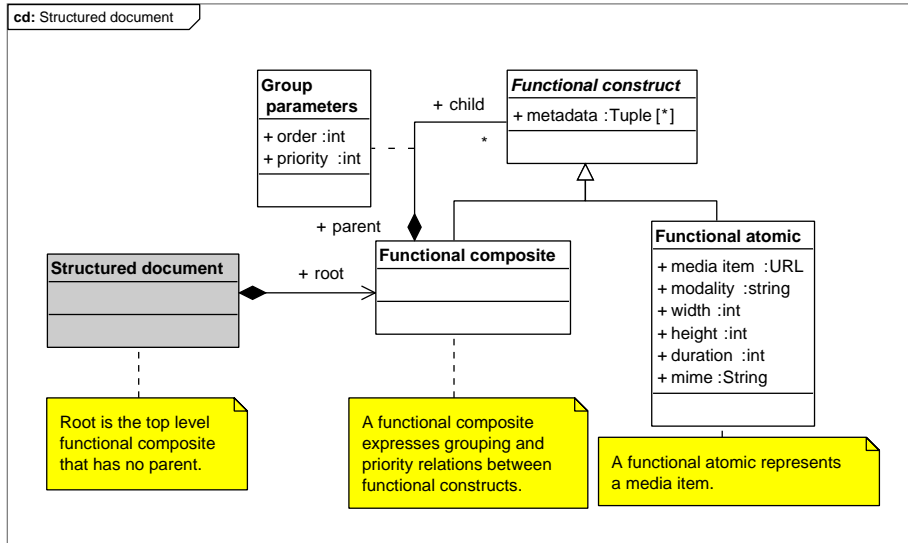


Figure 4.4: In a structured document function is either explicitly described in a *functional composite* or implicitly represented by a *functional atomic*.

strategy, soft constraints for multimedia documents are to be expressed by the designer as part of the stylesheet. This includes defining priorities between soft constraints so that the formatter optimizes the multimedia document in accordance with the intention of the designer⁵.

Soft constraints thus complicate significantly the authoring of a stylesheet, as a designer needs to deal with both the authoring of alternative formatting rules and the optimization of soft constraints. In addition, soft constraints significantly increase the computational complexity for the formatter implementing the model. For these reasons we exclude soft constraints from the model. As a result, the means to optimize the formatting of multimedia document for aesthetic quality are limited.

4.3 Modeling the structured document

Similar to the traditional document engineering model we represent authoring effort in a structured document. As stated by requirement FUNCTIONAL VOCABULARY (#9/p.63), the vocabulary used to represent the structured document should, on the one hand, abstract from the document form, and on the other hand be sufficiently expressive to allow an author to represent the document function she intends to convey.

⁵Designers typically need to balance multiple interests, including those of the author, content provider, designer and user.

In this section, we describe the structured document, illustrated by a UML diagram in figure 4.4. We first describe the explicit representation of media items in a structured document, which clarifies the classes *functional composite* and *functional atomic*. Subsequently we elaborate on the functional vocabulary, which in addition to grouping and ordering in the traditional model, also models priorities, which may be used to resolve potential formatting failures.

4.3.1 Explicit representation of media items

Recall that document function may be represented either explicitly, or implicitly through a media item. In figure 4.4, we represent *functional constructs* that represent function explicitly by *functional composites* whereas functional constructs that represent function implicitly through media items are represented by *functional atomics*. Note that both functional atomics and functional composites are modeled as subclass of *functional constructs*⁶, which means that they are a special type of functional construct.

To avoid the complexities of modeling media content, we assume atomics refer to their media content by the *media item* URI. Modeling direct embedding of media items in the structured document may be achieved by using data URLs [104].

In contrast to the traditional textual model, functional constructs that represent function implicitly also require metadata. For example, the generic style rules for multimedia documents require at least the associated *modality* of the functional construct in order to invoke the appropriate default style rule. Therefore, independently of whether document function is represented implicitly or explicitly, in our model we associate *metadata* with a functional construct, as illustrated in figure 4.4. Here, *metadata* refers to the complete set of metadata associated with a functional construct.

In this context, it is important to clarify that in our model, explicitly described document function refers to metadata associated with a functional construct that allows a style rule to format it appropriately. However, metadata does not necessarily need to be embedded in the structured document, as is typically the case in the traditional textual model, but may be accessed from external resources. This facilitates reuse of existing metadata as specified by requirement OPTIMIZED FOR REUSE (#12/p.67).

Requirement FORM CONSTRUCT PROPERTIES (#11/p.67) states that the form properties that may cause constraint violation should be explicitly described. Therefore, the *width*, *height* and *duration* of the media item are made explicit. Note that these are the intrinsic width, height and duration associated with the media item. These may be changed by styling effects such as scaling and cropping and therefore are not necessarily identical to the width, height and duration of the resulting form construct. Furthermore, the *MIME* type is necessary to detect whether the format of a particular media is supported by the delivery context. Note that the *MIME* type of a media item does not necessarily specify the modality as it is possible that a media item with the *image/jpeg* MIME type portrays text.

4.3.2 Representing grouping, ordering and priorities

Requirement EXPRESS GROUPING (#9.1/p.63) states that the vocabulary used to represent a structured document should be able to denote groupings of functional constructs. Similar to the traditional document engineering model, we model grouping by a containment hierarchy. As a result, a *functional composite* may be the parent of multiple functional constructs (functional

⁶Subclass relationships are represented in UML by an open arrow.

atomics and/or functional composites). The common ancestor of all functional constructs in the document is the functional construct at the top of the containment hierarchy, denoted as the *root* functional construct. In figure 4.4, the *root* attribute represented in the structured document, refers to the root functional construct and (implicitly) to all its descendants. A containment hierarchy has certain computational advantages since a branch can be processed independently from the rest of the tree. However, a tree structure does not necessarily result from the derived requirements and is in some cases too restrictive. For example, consider a two dimensional table. Every cell in a table is part of its row and of its a column. This cannot be expressed in a tree structure since it would require a cell to have two parents, which is not possible. Text-based document engineering technology circumvents this limitation by associating metadata with a functional construct to denote its membership to a group (e.g. in CSS `class='column1'`). In our extended document engineering model we adopt a similar approach.

Requirement EXPRESS ORDER (#9.2/p.64) states that the vocabulary used to represent a structured document should be able to represent ordering between functional constructs. This corresponds to the text-based model where ordering is implicitly represented by the order of the text within the structured document. We adopt a similar approach for our extended document engineering model and assume the specified order of the children of a composite to be the main ordering of the content. If necessary, alternative orderings may be represented by associated metadata.

Finally, requirement EXPRESS PRIORITY (#9.3/p.64) states that the vocabulary used to represent a structured document should be able to represent priorities. Prioritizing functional constructs is necessary in case the formatting of a multimedia document fails and a functional construct needs to be omitted. The traditional model does not require functional constructs to have associated priorities since the formatting is guaranteed to succeed. We model priorities as part of the parent-child relationship (i.e. every parent-child relation has an associated order and an associated priority). As a result, priorities are defined within the scope of their parent. This has the computational advantage that a functional construct may be omitted without affecting other functional constructs outside the scope of the parent. For this reason we disallow removing functional constructs outside the scope of the parent, although an author may consider them less important than the least important functional construct within the scope of the parent.

4.4 Modeling the document form

Loosely speaking, the document form is automatically generated based on the application of a stylesheet to a structured document. Recall that the document form consists of a medium, defined by the delivery context and the inscription, which represents the document function as represented by the structured document. Since the medium is beyond the control of the author and designer it is more correct to state that the *inscription* is automatically generated, and our form vocabulary is used to represent the inscription. In this section we present our form vocabulary that is based on the Amsterdam Hypermedia Model (AHM), which is an authoring vocabulary for multimedia documents. Figure 4.5 presents a graphical representation of the vocabulary's form constructs in UML.

According to requirement FORM VOCABULARY (#10/p.65), the vocabulary to describe document form should be sufficiently expressive to represent form conventions, and abstract from insignificant formatting details.

More specifically, requirement STYLE (#10.3/p.66) states that the vocabulary should be sufficiently expressive to describe the styling of a multimedia document. We model this by as-

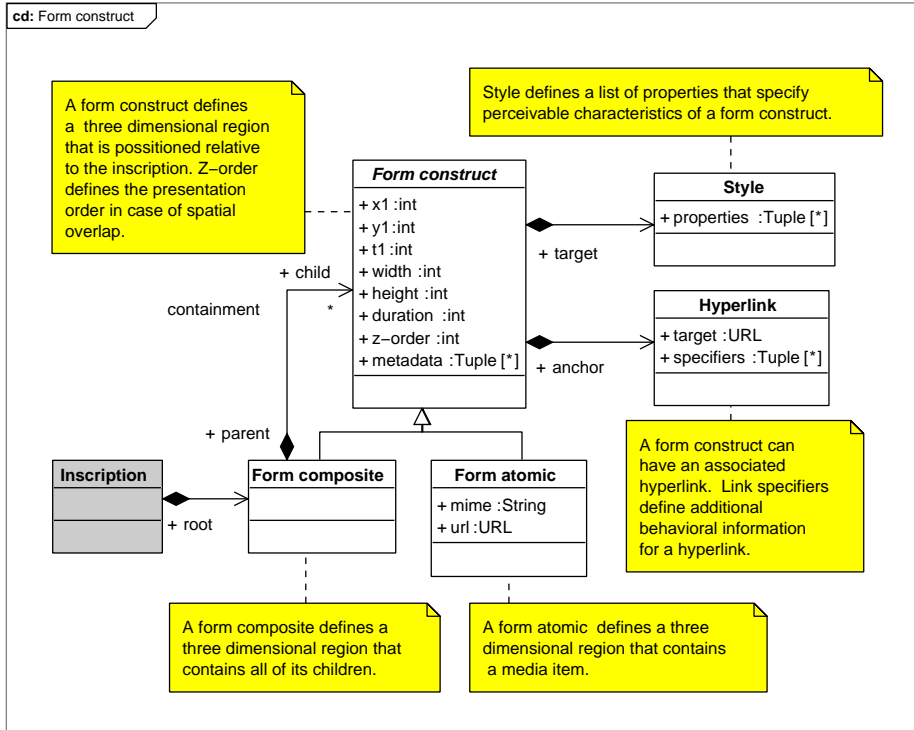


Figure 4.5: A *form construct* defines a three dimensional region that is used as a building block for describing the inscription of a multimedia document.

sociating a list of style properties with a form construct. Note that, similar to the AHM, we abstract from specifying style attributes because the specifics of style attributes may depend on the specific presentation format used to communicate the document form. However, typical style attributes used in multimedia documents would include color, font and transition effects.

Furthermore, the HYPERLINKS (#10.4/p.67) requirement states that the form vocabulary should be sufficiently expressive to denote hyperlinks in a multimedia document. We model hyperlinks by associating hyperlink properties with a form construct. The region defined by the form construct is then interpreted as the anchor of the link. A hyperlink minimally includes a *target*, which refers to the referred resource. Furthermore, the *link specifiers* specify the behavior of a link. This includes directives whether the link should be opened within the current presentation or using an external application.

In addition, requirement MEDIA ITEM (#10.1/p.65) states that the form vocabulary should be sufficiently expressive to denote media items. In our model we represent a media item by a *form atomic*, which is a special type of form construct as indicated by the subclass of relation in figure 4.5. A form atomic contains a *URL* attribute that refers to the URL of the media item and

the *MIME* type, which indicates the type of the referred media item. The latter is needed for the renderer of the document to correctly present the media item. Note that a form atomic, like its corresponding functional atomic, has a MIME type. Typically, the MIME type of the functional atomic and the resulting form atomic are identical. However, this is not necessarily the case as the transformation rule may transform the media item. For example, the generic transformation rule may convert a media item with a MIME type that could not be played in a particular delivery context to a media item with a compatible MIME type (e.g. text to speech).

In contrast to authoring vocabularies such as the AHM, our form vocabulary abstracts from the authoring phase. Therefore, we can simplify the form vocabulary compared to an authoring vocabulary by omitting support for authoring conveniences. In the remainder of this section we elaborate on our representation of three dimensional bounding boxes, whereas the AHM describes the temporal and spatial dimensions separately. Furthermore, we organize form constructs in a containment hierarchy, which reduces design effort significantly. However, a containment hierarchy imposes limitations that in some case may be too restrictive. We discuss these in the last section.

4.4.1 Three dimensional bounding box

Requirement FORM CONSTRUCT PROPERTIES (#11/p.67) states that the properties of a form construct that may cause a constraint violation should be explicitly described. This minimally includes required spatio-temporal resources. Therefore, based on the AHM, we model a *form construct* by a three dimensional box-shaped region of which the extents are denoted by *width*, *height* and *duration*. Additionally, requirement LAYOUT (#10.2/p.66) states that the form vocabulary should be sufficiently expressive to represent the layout of a multimedia document. Therefore, the position of a form construct with respect to the inscription is denoted by the coordinates x , y and t . Furthermore, the *z-order* defines the stacking order of form constructs in case of overlap.

One may argue that a three dimensional, box-shaped region is too restrictive, as regions in a multimedia document are not necessarily rectangular. For example, vector graphics may be used to specify graphics that have a different shape. However, since most spatial media items are rectangular and the computational complexity of detecting spatio-temporal constraint violations is significantly lower for 3 dimensional bounding boxes, we model regions as three dimensional bounding boxes. For similar reasons, the model does not support rotation of three dimensional regions.

In our extended model, the spatial and temporal layout are combined in a form construct, whereas the AHM (and SMIL) explicitly separate the spatial layout from the temporal layout. The advantage for an author of separating temporal layout from spatial layout is that the model allows an author to express that multiple media items are presented using the same spatial region. As a result, the style attributes associated with a region may be inherited and do not need to be specified for each individual media item. The disadvantage, however, is that the formatter needs to keep track of a separate temporal and spatial hierarchy, which complicates the processing model. Since we automatically generate the document form and we therefore do not need the manual authoring conveniences provided by the AHM, we use a simplified model that combines the temporal and spatial layout.

Note that in the perceived form construct the function is represented implicitly, while often it has been made explicit in the corresponding functional construct and its metadata. This metadata, such as author and title, that may be used to apply the appropriate style rule, is typically lost in

the traditional document engineering model. However, requirement OPTIMIZED FOR REUSE (#12/p.67) states that the extended document engineering model should be optimized for reuse. Therefore, the extended document model supports the preservation of metadata by explicitly including *metadata* within a form construct as indicated in figure 4.5.

4.4.2 Discussion: the containment hierarchy

A *form composite* is a special type of form construct that contains zero or more form constructs (which may be form atomics or form composites). Similar to formatting vocabularies such as \TeX and XSL-FO, composites are used to model a strict, single containment hierarchy. This does not necessarily follow from the requirements derived in chapter 3. To illustrate this, document formats such as the first version of SMIL [149] do not (fully) use a containment hierarchy to represent the document form⁷. However, having a single containment hierarchy significantly reduces the complexity for a stylesheet designer, for example the ability to specify relative spatio-temporal constraints that are automatically inherited by its children. In this model, the spatio-temporal constraints of the delivery context can be imposed on the *root* form construct, which represents the form construct that is the predecessor of all other form constructs in the document form. Since all form constructs are contained within the root, the spatio-temporal resource constraints automatically apply to all form constructs in the document form. Likewise, to position a group of form constructs relative to another group of form constructs, it suffices to constrain the two bounding boxes that contain the respective groups. Without a strict containment hierarchy, an explicit constraint specification between all involved form constructs would be necessary.

In addition, a containment hierarchy allows descendants of a form construct to inherit designated style attributes⁸. For example, the specification of a background color may be inherited by the descendants of a form construct, which prevents having to explicitly state the background color for each individual form construct. Note that CSS lacks explicit form constructs, and style properties are inherited over the functional construct hierarchy, not over the form construct hierarchy.

In a strict containment hierarchy, however, a form construct cannot have more than one parent. As a result, a functional grouping that exists between two form constructs that do not share a parent is not explicitly representable in the document form. This limitation also applies to a text-based model and becomes apparent in, for example, the formatting of tables. Typically, every cell in a table belongs to both a row and column. Based on the contents of the cells, the width of a column can be determined once the width of all the cells in the column are known. Similarly, based on the contents of a cell, the height of a row can be determined once the height of all the cells is known. Clearly, there is a dependency between the width and height of a cell. However, this dependency cannot be expressed in a containment hierarchy since rows and columns are orthogonal groups of cells. Instead, most text-based document engineering tools, such as \LaTeX , require the author to either fix the width or height of a cell *a priori*⁹.

Unfortunately, orthogonal grouping occurs more often for multimedia documents since spatio-temporal relations between form constructs need to be explicitly expressed. To illustrate this,

⁷SMIL1 represents only the temporal dimension by a containment hierarchy. Versions after SMIL1, notably SMIL2 [155], however also include support for a containment hierarchy for the spatial dimensions, although this is not mandatory.

⁸Not all style attributes are suited to be inherited. In chapter 5 we elaborate on this.

⁹Some HTML formatters allow a table to resize already rendered cells while processing the remaining cells. The assumption, however, is that there are no spatial constraints that require an overflow strategy, such as for documents that are constrained by the size of a piece of paper.

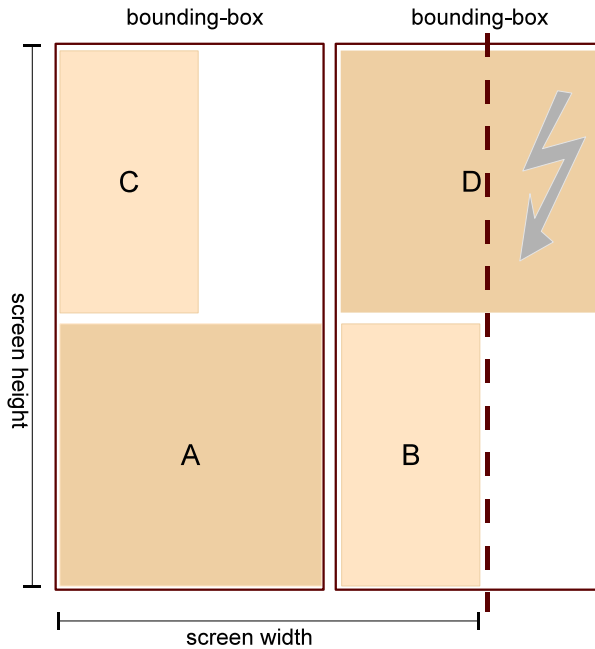


Figure 4.6: Bounding-box issue

consider figure 4.6, which shows 4 spatial media items (we omit the temporal dimension), two large ones (A and D) and two smaller ones (B and C). All images can be presented on the screen together by putting a large image next to a small image. However, this solution will not be found if the images A and C, as well as the images B and D are together in a bounding box. Since the bounding box is as large as its largest child, this model will find no way to present the images on a single screen.

Although orthogonal groupings can be considered a necessity in a document engineering model for multimedia documents, the conceptual implications for the model are significant as it introduces dependencies between transformation rules. For example, the transformation rule that aligns captions depends on the transformation rule that formats a caption and image. If the first transformation rule fails it is not clear if the second rule should be executed or not. Similarly, the consequences if the second rule fails while the first rule succeeds are not clear. Therefore, we decided to exclude orthogonal groupings in favor of a single containment hierarchy as used in the text-based model.

4.5 Summary and Conclusion

In this chapter we presented our extended document engineering model. We model a single transformation from document function to document form using a style sheet. Compared to the

traditional model, our model extends the notions of function, form and style to meet the specific requirements of multimedia documents. We include an explicit and parametrized delivery context that represents the constraints of the environment the document is played in, and the specification of alternative style rules that are automatically invoked by the formatter if the resulting document form does not comply to the hard constraints imposed by the delivery context. For simplicity, the model does not support soft constraints, as a consequence, aesthetic preferences are harder to make explicit. The model accommodates the preservation and reuse of metadata by explicitly modeling it in the structured document, stylesheet and document form. Media items and their intrinsic spatio-temporal dimensions are recorded explicitly in the structured document, whereas the media content itself is referred to by means of a URI. All functional constructs have an explicit priority. Both the structured document and the document form are modeled as an ordered tree. More flexible graph structures may be desired in some cases, but are not supported for reasons of simplicity. Similarly, the formatting model is based on a simple, three dimensional bounding box paradigm.

We discuss the main pros and cons of our extended document engineering model below.

Author abstracts from document form (pro) The principle of document engineering is abstraction from the document form. This leads to certain advantages for the author of the document, such as the automatic adaptation of the document form and the reuse of style. Although the traditional document engineering model does not apply to multimedia documents, our model makes these advantages available to the authors of multimedia documents.

Author abstracts from delivery context (pro) An advantage of our model is that the author of a document can also abstract from the properties of a particular delivery context. Subsequently, a formatter that implements the model can potentially generate a presentation for an obscure delivery context the author is *a priori* unaware of (provided that the properties of the delivery context can be represented). In contrast, the author of a SMIL or HTML document needs to be aware of the potential delivery contexts the document is played in and specify *a priori* potential adaptations for the various delivery contexts.

Formatter needs to have more knowledge (con) In contrast to the text-based model there is no generic overflow strategy for multimedia documents. Therefore, the formatting of a multimedia document requires the specification of more explicit knowledge to compensate for potential formatting failures.

Less abstraction allowed in document form (con) In contrast to the text-based model, the representation of document form for multimedia documents is more detailed than for text-based documents. For example, the formatter of a text-based document abstracts over the length of a document: depending on the context, the document form can be rendered using pages or by using scroll-bars. The additional detail for multimedia documents is necessary because the formatting of a multimedia document is not guaranteed to succeed. Therefore, the extended document engineering model needs to make more detailed formatting decisions concerning the document form. Consequently, the adaptability of the document form of a multimedia document is limited compared to text-based documents.

Chapter 5

Cuypers document engineering framework

The transformation from a structured document to its corresponding document form is performed by a formatter, which is independent of a particular document. In this chapter we present Cuypers¹, our document engineering framework that is implemented based on the extended document engineering model, described in the previous chapter.

The framework demonstrates that a formatter based on our extended model can be implemented using commonly available software tools and commodity hardware. The framework will be evaluated based on three usage scenarios in the next chapter.

In section 5.1 we present the high-level architecture of the Cuypers framework and how it can be integrated into a more general Web architecture. Section 5.2 describes the document engineering vocabularies that describe the structured document, the stylesheet, the document form and the delivery context. Section 5.3 describes the Cuypers formatter, which implements the core of the extended document engineering model. In section 5.4, we conclude and highlight the most important lessons learned during the implementation of Cuypers.

This chapter is partly based on material that has previously been published in [65, 66, 98, 140, 141].

5.1 Overview of the Cuypers framework architecture

Recall that document engineering is especially desirable in a web context due to the heterogeneous nature of the available delivery context. Therefore, requirement WORLD WIDE WEB (#13/p.68) states that the formatter that implements the extended document engineering model should comply to the architectural requirements of the web. Although the document engineering model abstracts from such architectural issues, it does affect the implementation of the formatter. The main goal of this section is to provide an overview of the architecture of the Cuypers

¹The Cuypers framework is the result of a continuous line of research that spans multiple funding projects, including the Dutch national projects NASH, ToKeN2000 and Dynamo; and the European ITEA/RTIPA and FP6/Question How projects.

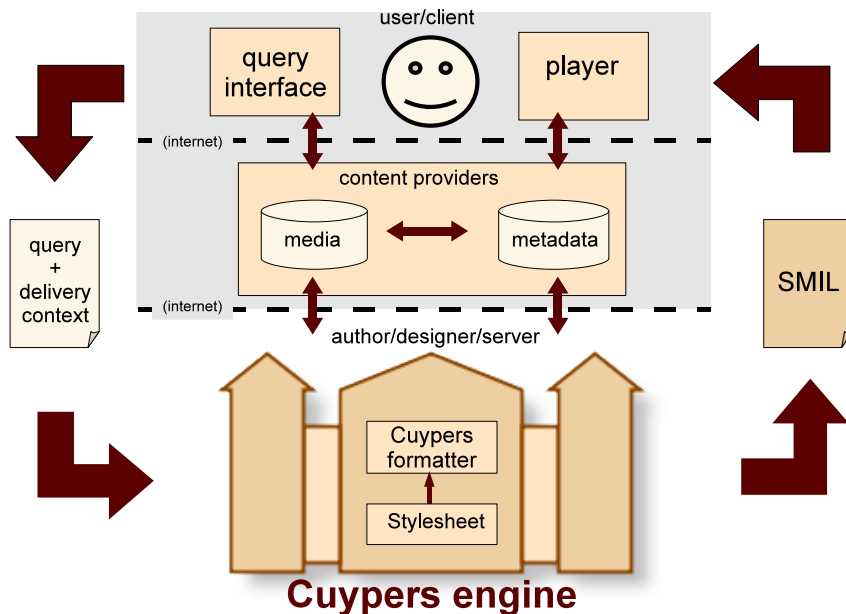


Figure 5.1: High-level overview of the Cuypers document engineering framework.

document engineering framework and to show the implications of a common web architecture on the implementation of the formatter.

In a web environment, server and client are typically independently developed entities that communicate through standardized, typically stateless protocols and data structures (see section 2.3.1). Since the document engineering model abstracts from the client-server issues imposed by the web, a one-to-one correspondence between the model and the implementation is not possible. Similar to text-based document engineering on the web, the transformation from structured document to document form as described by our model could be implemented by multiple sequential transformations. We refer to these transformations, which may be distributed across a network, as the *transformation chain* (see section 2.1.3). In this respect it is important to clarify the relation between the commonly used terms reader, user and client. The *reader* denotes the person who perceives the document form as described in our extended model (see chapter 4). The *user* is the person who uses an interactive web application to access a document made available by an *author*. In this context *user* corresponds to the *reader* described in the model. Note that the term “document” may refer to the structured document from the authors perspective, and the document form from the readers perspective. The user is also associated with the *client*, which denotes the web-application that the user uses to request a document. However, depending on the context, it may also refer to the more general notion of a software component that receives information from a server. Similarly, *server* is typically associated with the web-server application that sends the requested document to the client. With respect to the extended document engineering model, the author of the structured document and the designer

of the stylesheet are typically associated with the server. Furthermore, in a web environment, the media items and associated metadata necessary to produce the document form may be distributed on multiple servers on the web. We refer to these as *content providers*.

A high-level overview of the Cuypers framework is given in Figure 5.1. The user specifies a request for a document through a query interface, which typically corresponds to a web-application that generates the URL that identifies the document on the server. In addition to the specification of the document, the web-application also sends the relevant properties of the delivery context that are used to adapt the document form. Requirement DELIVERY CONTEXT (#5/p.60) states that the format of the delivery context should be standardized; we therefore base the specification of the delivery context on CC/PP [90]. However, since few client applications implement this standard, the properties of the delivery context currently need to be explicitly specified by the user².

The server, indicated in figure 5.1 by the Cuypers engine, produces the document form based on the user's query, the delivery context, the stylesheet and the media and metadata provided by the content providers. Part of the Cuypers engine is the Cuypers formatter that we describe in section 5.3. The produced document form is sent back to the client. In a web context the client is independent of the server. Therefore, the format of the document form that is sent to the client needs to be standardized. In figure 5.1 we represent the document form as a SMIL document, although other formats are possible. The use of SMIL allowed us to use the Cuypers server with a wide range of Web clients, including RealPlayer [124], Ambulant [33], GRiNS [118], QuickTime [9] and Internet Explorer [108]).

Based on the high-level overview of figure 5.1, the remainder of this section will zoom in on the five-step transformation chain that is the core of the Cuypers engine (section 5.1.1 and the way it has been embedded in a Web server (section 5.1.2). We conclude this overview by discussing the differences between Cuypers and more traditional implementations of document transformation chains (section 5.1.3).

5.1.1 The five steps of the Cuypers transformation chain

The document engineering model describes a structured document that is transformed by a stylesheet to its corresponding document form. Although this is often modelled as a single transformation, in practice there are often software engineering reasons to implement the transformation as a sequence of transformations (i.e. see section 2.1.3). For example, in text-based document engineering, L^AT_EX and XSL-FO first transform to intermediate formats that can then be relatively easily transformed to various delivery formats, such as PostScript, PDF and HTML. Furthermore, structured documents (e.g. HTML) in a modern web environment are often dynamically generated from existing (database) content.

Similar to text-based document engineering, Cuypers implements the transformation from structured document to document form by a sequence of transformation steps. As a result, document engineering concepts described in the model, such as *structured document*, *delivery context*, *document form* and *stylesheet* are implemented in the framework in a format that is optimized for a particular transformation step. Figure 5.2 illustrates the transformation chain in Cuypers.

²Although most traditional document engineering models abstract from the delivery context, there are exceptions. Especially for mobile devices and hardware used by people with disabilities, the properties of the delivery context are highly heterogeneous. This requires formatting strategies that are adapted to the specific device or user. The W3C activity on Ubiquitous Web Applications [72] aims to standardize device and user profiles to make this possible.

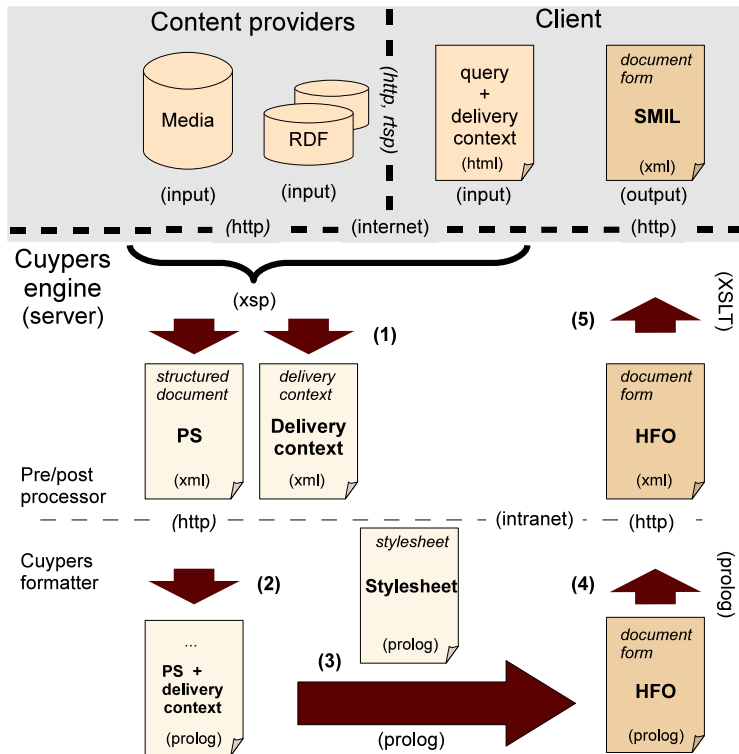


Figure 5.2: Cuypers server engine architecture

Concepts that refer to the model are represented using an italic font (e.g. *structured document*, *delivery context*, *stylesheet* and *document form*), whereas their corresponding representations in the framework are represented using a bold font (e.g. **PS**, **delivery context**, **stylesheet** and **HFO**).

As illustrated in figure 5.2 the structured document is represented in Cuypers by *Presentation Structures* (PS). This term may be counter intuitive as it refers to the functional structure of the multimedia document, which abstracts from presentation related issues. However, “Presentation” is in this context used as a synonym for multimedia documents), comparable to “book” or “rapport” in text-based document engineering. In section 5.2.2 we discuss the PSs and how they implement the functional constructs as described by the model. The document form in Cuypers is represented by a *Hypermedia Formatting Object* (HFO), and is discussed in section 5.2.3. The *delivery context* (section 5.2.1) and *stylesheet* (section 5.2.4) are implemented by data structures with the same name.

The number of transformation steps, in general, depends on the complexity of the transformation. In Cuypers we identify the following five steps:

Aggregation (1) The first transformation, represented in figure 5.2, aggregates the different inputs, including the user query and properties of the delivery context and dynamically

generates the PS representing the structured document from external repositories.

Note that the dynamic generation of a structured document falls outside the scope of our model. However, since pre-authored PSs are in general not readily available, we need to include this processing step in the framework. In the next chapter, when we describe specific evaluation scenarios, we elaborate on the methods used to generate the structured document (see section 6.2, section 6.3 and section 6.4).

Normalization (2) The second transformation concerns a relatively straightforward transformation, which merges PS and delivery context into a format that can be interpreted by the formatter, which we implemented in Prolog.

Formatting (3) The third transformation generates the document form, and is the conceptually the main step in the transformation chain. It involves the application of style rules and the detection and resolution of formatting failures. As we implemented the formatter in Prolog, the result of this transformation is an HFO represented as a Prolog data structure (e.g. a Prolog term). In section 5.3 we describe the implementation of the formatting step in more detail.

Serialization (4) The fourth transformation serializes the Prolog HFOs to an XML format. Although we could transform an HFO directly to a specific delivery format, such as SMIL or HTML+Time, we intentionally serialize to a more abstract format. This way, it is relatively easy to support multiple delivery formats.

Standardization (5) The final, fifth transformation is a relatively straightforward XSL(T) transformation that transforms the XML representation of an HFO in the delivery format that is processable by the client application. Currently we have implemented support for SMIL, SMIL2 and HTML+TIME, but other suitable formats may be added by simply implementing a XSL(T) transformation that transforms the HFO XML serialization to the desired format.

5.1.2 Embedding the Cuypers chain into a Web server

To manage the server-side transformation chain we use the Cocoon [133] Web development framework, which is based on a flexible pipe and filters software architecture (section 2.3.1). In an earlier version of the Cuypers engine all pre and post processing steps were hosted by Cocoon through servlet, embedding and auxiliary communicator software. For example, the Cuypers formatter, at that time implemented in the constraint logic programming system ECLⁱPS^e, was embedded in a Java auxiliary application hosted by Cocoon. Although this approach successfully implemented the Cuypers engine we found that maintaining the system was too complicated. The software components had been independently developed, and communication between software components was only successful for particular versions of the software, with incompatibilities occurring when upgrading one of the software components. Because of these dependencies, maintaining an up to date version of the system as a whole was non trivial. In addition, debugging the system was overly complex as the cause of a reported error was often hard to infer when multiple layers of embedded technology are in use (e.g. the formatter in Prolog, the SWI-Prolog interpreter, the Java communication layer, the Cocoon transformation layer, the Java Virtual Machine, etc.).

Therefore, in the current version we have reimplemented the Cuypers formatter as a standalone HTTP server. Cocoon sends the query, represented as a URL, to the Prolog formatter,

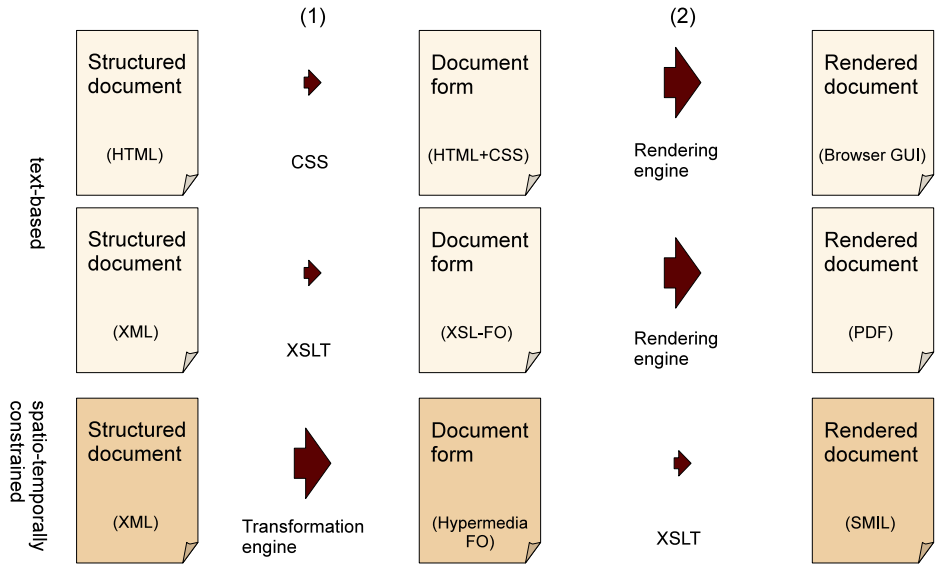


Figure 5.3: Formatting differences between text-based documents and multimedia documents. The larger arrows represent computationally more expensive transformations.

which performs the transformation and returns the result directly in XML. The advantage of this approach is that the transformation does not depend on third party libraries and can be tested independently of the Cocoon transformation chain. In addition, it is also faster because the Prolog engine used to implement the Cuypers formatter needs to be started only once, whereas the embedded solution started a new Prolog session for each request.

5.1.3 Discussion: The Cuypers versus the traditional transformation chain

Figure 5.3 illustrates the differences between transformation processing of typical text-based documents versus multimedia documents. The upper two transformations refer to text-based transformations, such as HTML or PDF. The first transformation (1) is typically processed on the server, which sends the result to the client. The client performs a second transformation (2), which results in the rendering of the document form. The larger arrow for the second transformation represents a computationally intensive step in the chain. For text-based formatting, typesetting the text typically involves complex algorithms for kerning, hyphenation and pagination. The smaller arrow represents a relatively simple syntactic transformation, typically performed by a server, while the computationally heavy step is performed by the client. As a result the load on the server is reduced. This is possible because the first transformation is independent of the delivery context, while the second is independent of the original structured document. The only constraint is that server and client should agree on the format of the communicated document and stylesheet.

In contrast to text-based document engineering, the formatting of a multimedia document

can fail, and every formatting step may depend on the delivery context. Therefore, most of the formatting decisions for multimedia documents are made during the first transformation step. In figure 5.3 this is illustrated in the lower transformation by a larger arrow denoting the transformation from a structured document to document form. Cuypers implements the computationally expensive step server-side. However, the big advantage of this approach is that it allows the use of standard web clients, including Real Player, Ambulant and GRiNS.

One can imagine an alternative model that complements the structured document with additional metadata and alternative style rules that allows adaptation of the document form client-side. This would, however, require standardization of the metadata necessary for adapting the document to its delivery context, and standardization of a transformation language with alternative style rules. It is unlikely such a standard will emerge in the near future, therefore we perform the transformation server-side.

5.1.4 Summary

The Cuypers framework implements a document engineering framework that complies to the client-server architecture of the web. Main advantage of this approach is that the formatter is applicable in a larger, heterogeneous environment where document engineering technology is especially desirable. Additional advantage is that off-the-shelve software components may be used, which reduces the implementation and maintenance effort of the framework. Disadvantage is that, in contrast to text-based document engineering, the computational expensive formatting step is necessarily performed server-side. As a result, the server may more easily be overloaded when multiple multimedia documents residing on the server are requested in parallel.

5.2 Cuypers vocabularies

The document engineering model described in the previous chapter, models the required properties of the vocabularies used to describe the structured document, stylesheet and document form. In addition, it models the properties of the delivery context, that, in contrast to the traditional text-based model, should be explicitly described.

In this section we present the implementation of these vocabularies within the Cuypers document engineering framework. We describe the representation of the *delivery context*, which describes the environment the document form is adapted for. The vocabulary that implements a structured document we have named *presentation structures* (PS). Note that the term “presentation” is used here as a synonym for multimedia document. The vocabulary that implements the document form we have named *hypermedia formatting objects* (HFO).

5.2.1 Delivery context

The generated document form should satisfy the constraints imposed by the delivery context. Section 4.1.2 modeled the required properties of the *delivery context*. In this section we describe the implementation of the delivery context. Figure 5.4 presents a UML diagram, that represents the delivery context as implemented in Cuypers (c.f. Figure 4.3 on page 77).

Analogously to the model, the delivery context in Cuypers includes a reference to a particular instance of a *structured document* the reader wishes to access, and a reference to the *stylesheet* that should be used to transform the selected structured document to its corresponding document form.

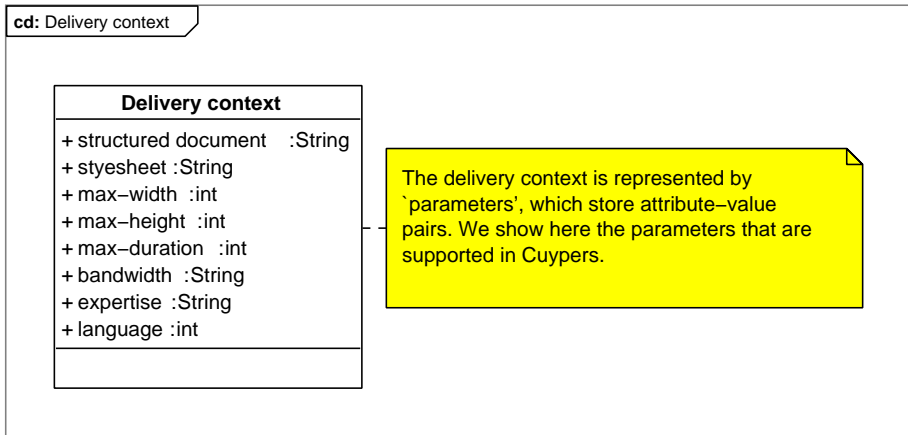


Figure 5.4: The *delivery context* is a data structure that represents the constraints imposed on the generated document form.

In a traditional text-based document engineering system, the transformation may be executed once the structured document and stylesheet are known to the formatter. However, as indicated in the model, for multimedia documents the properties of the delivery context that constrain the document form should be made explicit. This includes the attributes *maximum-width*, *maximum-height*, *maximum-duration*, which denote the maximum available spatio-temporal resources for the document form.

Besides spatio-temporal constraints that should be respected, the delivery context may constrain other properties of the document form. In Cuypers we have implemented support for *bandwidth*, which can have one of the values fast, medium or slow. We use this value to optimize the document form to the available bandwidth.

The *expertise* attribute represents the expertise level of the reader, which we use to tailor the document form to the interest of the reader. We represent expertise by an integer value from 1 (novice) to 5 (expert), which is simple but sufficient to illustrate the principle of user adaptation.

Finally, the *language* attribute is used to adapt the document form to the preferred language of the reader. We represent language by an HTML language code³.

5.2.2 Presentation Structures

Our presentation structures implement a functional vocabulary used to represent the structured document⁴. Section 4.3 modeled the required properties of a functional vocabulary for multimedia documents. In this section we present the implementation of the vocabulary within

³<http://www.ietf.org/rfc/rfc1766.txt>

⁴The term “presentation” in “presentation structure” is used as a synonym for a multimedia document and does not refer to the action of presenting.

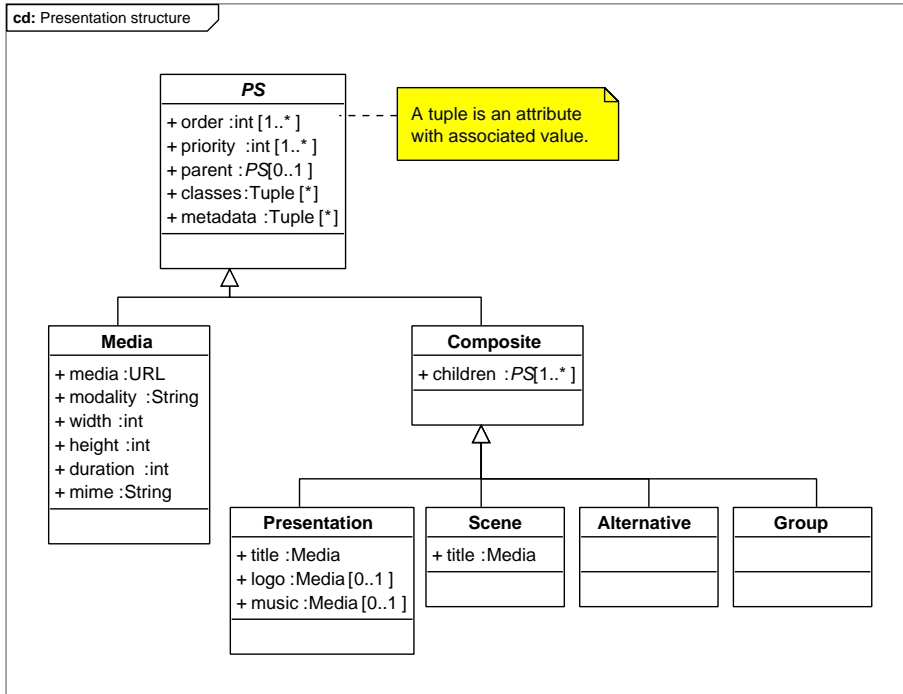


Figure 5.5: *Presentation structures* (PS) implement the structured document.

the Cuypers framework. Figure 5.5 shows a UML diagram representing the presentation structure vocabulary. Furthermore, table 5.1 presents the mapping between functional constructs described by our extended document engineering model and the corresponding implementation in the framework (c.f. Figure 4.4).

As illustrated in table 5.1, the presentation structures Presentation PS, Scene PS, Alternative PS and Group PS are different variants of the composite functional constructs in our model. They implement specific form conventions that are relatively generic (and shared by the scenarios presented in the next chapter), so we included them in our presentation structure vocabulary. We discuss this decision after the description of the implemented presentation structures.

PS

A PS defines the common properties that are shared by all presentation structures. In figure 5.5 this is represented by a UML inheritance relation. Analogously to the model, a PS is associated with an order and priority attribute. *Order* represents the linear order between presentation structures. It is composed of one or more integer values, which correspond to the typical section numbering also found in textual documents. For example, the value ‘1, 1, 4’ denotes that the

Model		Framework
Functional construct (4.3)	→	PS
Functional atomic (4.3.1)	→	Media PS
Functional composite (4.3.2)	→	Composite PS
„	→	Presentation PS
„	→	Scene PS
„	→	Alternative PS
„	→	Group PS

Table 5.1: A mapping between the functional constructs of the model and the presentation structures (PS) used in the implementation.

presentation structure is ordered fourth relative to its siblings, whereas its parent and parent’s parent are both ordered first relative to their siblings. *Priority* is represented in a similar way and denotes the order in which presentation structures should be suppressed from the structured document in case there are insufficient resources available for a successful transformation. The *parent* attribute denotes the parent of the PS. All PSs, except for the root have exactly one parent. This information may be used to contextualize the application of a style rule. A PS may have associated *class* attributes that denote the specific function of the PS (e.g. “title”, “quote” or “introduction”). Similar to CSS class attributes these are used to select a specific style rule to transform the PS. Finally, additional *metadata* an author wishes to preserve may be represented by zero or more attribute-value tuples.

All properties defined by the model for a functional construct are implemented by a PS. However, *classes* are explicitly described in a PS, whereas they are implicitly represented as *metadata* in the model (see section 4.3).

Media PS

A Media PS represents a media item in the structured document. Analogously to the model, a Media PS has a *media* reference, which is a URL that refers to the media item, and an associated *MIME* type. Furthermore, a Media PS denotes the *width*, *height* and *duration* of a media item, which are necessary to detect constraint violations of spatio-temporal resources. If one of these is not applicable, the respective value is set to 0. Note that although a Media PS contains a media item, which has associated form properties that are necessary for future processing, it abstracts from presentational issues. For example, the dimensions associated with a Media PS are not necessarily the dimensions that are used in the document form. The formatter is free to adapt a Media PS as long as the function that it represents is preserved. Finally, a Media PS includes *modality* information, which is used to determine the default applicable style rule. Since true modality information is typically not available, we use the MIME type associated with a media item to guess the modality. For example, an media item with MIME type `image/jpeg` typically has the modality “image”. However, this is not necessarily the case since a `jpeg` image may portray a (scanned) text. In this case an inappropriate default style rule may be applied resulting in incorrect behavior.

All properties defined by the model for a functional atomic are implemented by a Media PS (see section 4.3.1).

Composite PS

A Composite PS is used to define groups of presentation structures within a structured document. Analogously to the model a Composite PS groups one or more children, which are denoted by the *children* attribute.

All presentation structures (except for the root) have exactly one parent. Consequently, a PS is, together with its siblings, member of exactly one group, resulting in a strict containment hierarchy with one PS as its root. The main advantage of this approach is that the transformation of a PS is independent of other orthogonal branches and therefore reduces the computational complexity of the transformation.

However, from an authoring perspective it may sometimes be desirable to assign a PS to multiple groups. For example, suppose there are multiple images with associated captions. An author may wish to group an image with its corresponding caption to express the relation between the two, but also a second group that contains just the captions. This, for example, to ensure that all captions in the group are aligned consistently. Similarly, a third group containing the images may be wished for, to express that all images within the group use a consistent color scheme. The consequence of this approach is that a PS could become dependent on any other PS in the document. This would increase the computational complexity of the transformation significantly, because it becomes harder to check for failures, and it becomes harder to find alternatives.

Therefore, in Cuypers we do not support explicit orthogonal grouping, but instead support some orthogonal grouping implicitly through class attributes (similar to HTML and CSS). Although class attributes may be used to select a particular style rule, explicit constraints on the members of such a group are not possible.

All properties defined by the model for a functional composite are implemented by a Composite PS (see section 4.3.2).

Presentation PS

A Presentation PS is used to implement the form convention that helps us to recognize the genre of a multimedia document. In Cuypers all generated multimedia documents include a *title*, *background music* and potentially one or more graphical *logos* representing the company or institute associated with the multimedia document. Since this form convention is relatively generic we include it in our functional vocabulary. Recall that, like all presentation structures, a Presentation PS abstracts from the document form, therefore the layout and styling of title, background music and logos are unspecified.

Scene PS

Typically, a multimedia document consists of multiple scenes, which help an author to rhetorically structure a multimedia document. A Scene PS implements the form convention that helps a reader to recognize a scene within a multimedia document and the media items that belong to it. For this purpose, a Scene PS may have an associated *title* denoting the scope of the scene, but this is not obligatory. Alternatively, a designer may choose to distinguishably format a scene by particular style elements, such as background color, background music, or borders. However, these are formatting decisions a Scene PS abstracts from.

Alternative PS

An Alternative PS allows an author to specify alternative presentation structures. This is, for example, used when there are multiple equivalent Media PSs available with different presentation properties, such as dimension, media quality or media type. Currently, the formatter simply selects the first available alternative and backtracks if this choice turns out not to be appropriate.

Group PS

A Group PS is a presentation structure that specifies that its children should be conveyed as a group. However, it should prevent that the user becomes cognitively overloaded, which happens if two or more media items played in parallel require the attention of the user. For example, when two videos or music fragments are played in parallel. This is especially useful when the structured document is automatically generated and the types of the media items are not known *a priori*.

To prevent cognitive overload, the style rule that implements the transformation from Group PS to its corresponding HFO should be aware of the modalities that are associated with a media item (see section 2.2.2). Unfortunately, the modalities used by a media item are, in general, not available. However, by using the MIME type one can often make a relatively good guess.

Discussion

Functional vocabularies often relate to the genre of a particular document. For example the functional vocabulary for the text-based genre “book” may include volume, part, chapter, section, sub-section and paragraph. Note that the functional vocabulary for the more specific genre “thesis” partly overlaps with the vocabulary for “book”. In contrast, the presentation structures vocabulary we defined for multimedia documents is relatively shallow compared to the vocabularies for text-based documents. One may argue this is the result of the relatively short history of multimedia documents compared to the long history of authoring text-based documents. The presentation structure vocabulary we defined here allows for a bare minimum of generic adaptation strategies. This includes alternative positioning of logos and title (Presentation PS, Scene PS), formatting of an arbitrary number of media items with arbitrary MIME types (Group PS) and the ability for an author to specify alternative media items (Alternative PS). This presentation structures vocabulary is used to implement the three scenarios described in the next chapter (see chapter 6). Although the vocabulary is relatively generic, other genre multimedia documents may need to extend, or specify a different set of presentation structures.

5.2.3 Hypermedia Formatting Objects

Hypermedia Formatting Objects (HFO) are a form vocabulary used to describe the document form of a multimedia document. Section 4.4 modeled the required properties of a form vocabulary for multimedia documents. Based on this model, we describe the form vocabulary that we implemented in the Cuypers framework. Figure 5.6 presents a UML diagram that illustrates the different types of implemented HFOs. Furthermore, table 5.2 presents the mapping between form constructs described in the model and its corresponding implementation in the HFO vocabulary.

The Root HFO, Box HFO and Text HFO implement specific composites with example formatting behavior that proved useful during the implementation of the scenarios described in the

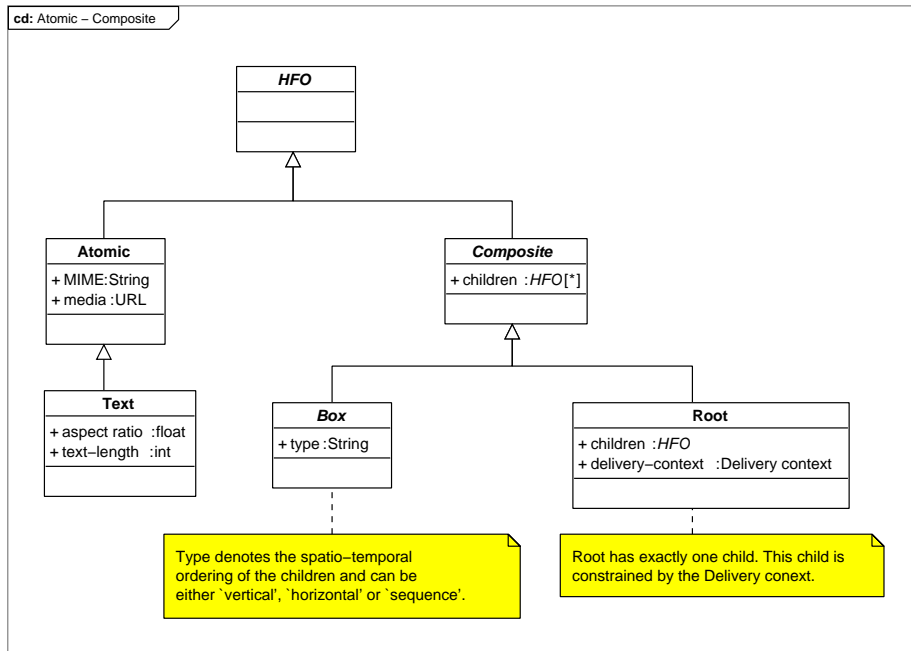


Figure 5.6: Hypermedia formatting objects implement the document form using a nested, 3D box model. All media content is in the Atomic HFOs, and can be grouped by using Composite HFOs. Text, Box and Root HFOs are examples of HFO implementations with more specific formatting behavior.

next chapter. We do not claim completeness, and the set of composite HFOs may need to be adapted or extended to achieve other formatting results.

In the remainder of this section we elaborate on the implementation of a functional construct by a HFO, which following the model, includes layout, styling and hyperlinks. Subsequently, we describe the different types of HFOs implemented in Cuypers. Finally, to serialize the document form we use the standard SMIL authoring vocabulary. We conclude by comparing the HFO and SMIL vocabularies.

HFO

An HFO defines the common properties that are shared by all HFOs. Figure 5.7 presents these properties in a UML diagram (c.f. figure 4.5 on page 82). On the left-hand side the HFO class is presented. Based on the CSS box formatting model (see section 2.2.2), it defines four three-dimensional bounding boxes to represent respectively the *margin*, *border*, *padding* and *content* box. The *content-box* of an HFO defines the region reserved for the content of the HFO. This can be a media item, or zero or more HFOs. Figure 5.8 presents a graphical illustration of the

Model		Framework
Form construct (4.4)	→	HFO
Form atomic (4.4.1)	→	Atomic HFO
Form composite (4.4.2)	→	Composite HFO
„	→	Root HFO
„	→	Box HFO
„	→	Text HFO

Table 5.2: Mapping form constructs to hypermedia formatting objects (HFO)

four bounding boxes.

The properties that define a *bounding box* are illustrated in figure 5.7. These include 6 absolute coordinates that represent the corners of the bounding box. The *z-order* associated with an HFO defines the stacking order in case of spatial overlap between HFOs. HFOs with a higher *z-order* value are presented on top of HFOs with lower *z-order* values. The *metadata* attribute represents a attribute-value list, that denotes the metadata that may be associated with an HFO.

Finally, an HFO includes the attributes *style* and *hyperlink*, which refer respectively to the Style class and Hyperlink class, presented in figure 5.7. These are discussed in more detail below.

Style

The Style class implements a set of styling properties that are associated with the corresponding HFO. The Style class in Figure 5.7 includes a list of example style properties that we have implemented in Cuypers. This list is not intended to be complete (for a complete list of the style-attributes implemented in Cuypers see appendix A). The style properties that relate to the spatial layout, such as *color*, *font* and *border* are derived from CSS and have comparable semantics [28]. Temporal style properties, such as *transition* and *delay* are based on SMIL properties and have identical semantics [155].

For design convenience we define, similar to CSS, an inheritance strategy for style attributes of a HFO⁵. Note that inheritance of style attributes, just like CSS, reduces the design effort for a stylesheet designer, but is not strictly necessary as each HFO can be styled individually.

Hyperlink

The Hyperlink class implements the ability to associate an HFO with a hyperlink. We base our implementation of hyperlinks on the specification of hyperlinks in SMIL⁶. The *target* refers to the resource the HFO is referring to. *Properties* specify the effect the activation of a hyperlink has to the presentation. Although SMIL defines a collection of relevant properties, such as *show*, *external* and *actuate*, Cuypers only implements the *external* property, which denotes whether or not a link should be opened in an external application.

⁵Like CSS, not all style attributes are inheritable, see appendix A for an overview of inheritable style attributes in Cuypers.

⁶See <http://www.w3.org/TR/REC-smil/#hyperlinking>

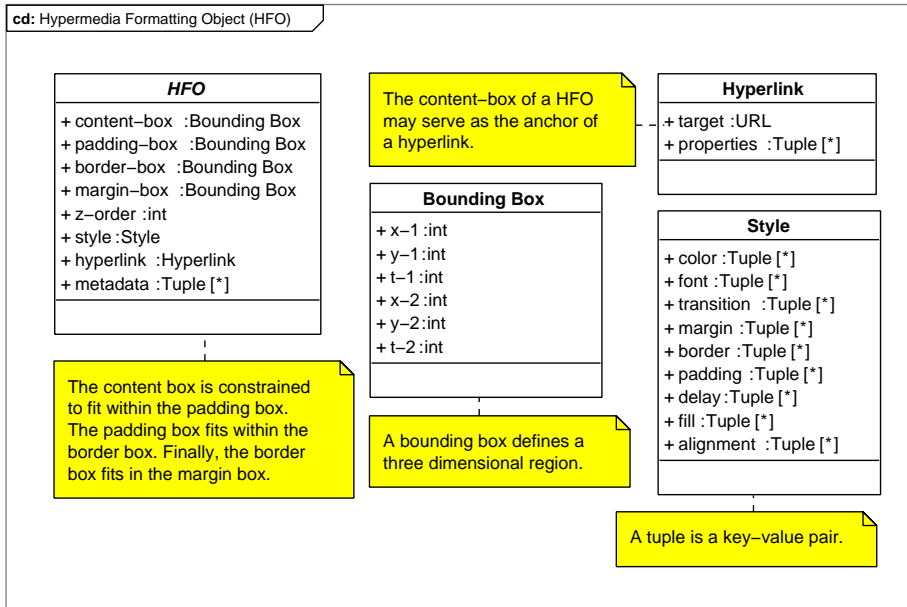


Figure 5.7: More detailed view of the 3D layout boxes, hyperlink and style properties of Hypermedia Formatting Objects.

Atomic HFO

An Atomic HFO is used to represent a media item in the document form. This includes the MIME type and URL of the *media item*. The content-box of an Atomic HFO is constrained to reserve sufficient resources for presenting the media item. Typically, the required resources are based on the metadata associated with the media item. Images, video and audio can be represented by a generic atomic HFO since the width, height and duration are known *a priori*⁷. and require no specific, media dependent calculations. The width and height of a textual media item are typically determined only at run-time, and require text-specific calculations. This is handled by the Text HFO.

Text HFO

The Text HFO calculates the width and height of a text media item based on the length of the text, the aspect-ratio and font properties, such as font-size and font-family. The *aspect-ratio* denotes the ratio between width and height of the content box. The *text-length* denotes the length of the text. The font-properties are specified as style properties.

⁷We assume media items represent pre-recorded assets, and do not support live streams with unknown duration.

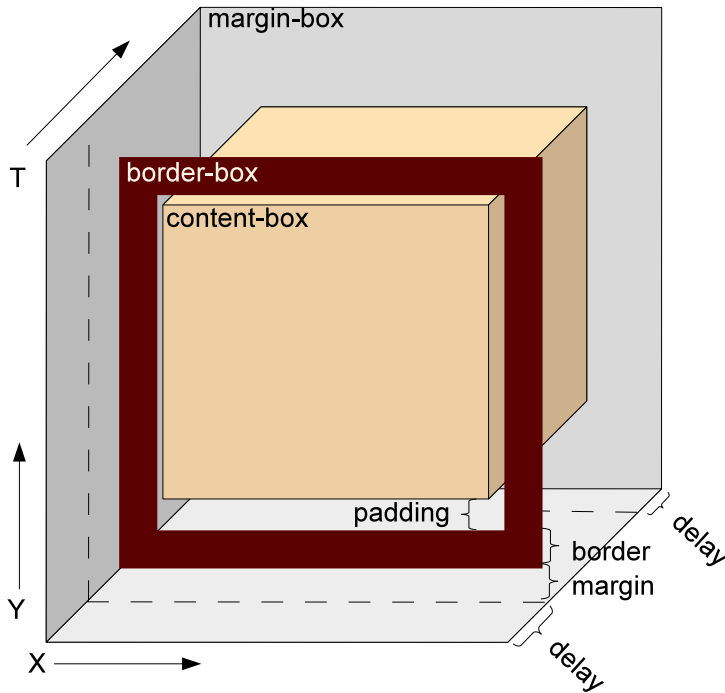


Figure 5.8: 3D view of the margin-box, border-box and content-box of a Hypermedia Formatting Object (the padding-box is not explicitly indicated).

Composite HFO

A Composite HFO is used to group zero or more HFOs, which are represented by the *children* attribute. All children have to be contained within the Composite HFO, resulting in a containment hierarchy (see section 4.4.2). As a result, the dimensions of a Composite HFO can be calculated based on the dimensions and positions of its children.

Furthermore, the inheritance strategy of the style attributes is based on the containment hierarchy imposed by a Composite HFO. This is in contrast to CSS, which is based on the tree structure of the structured document. Basing the inheritance hierarchy on the structured document may, however, lead to odd behavior. For example, consider a structured document that contains a paragraph and within this paragraph a footnote. Since the footnote is a child of the paragraph, the style attributes of the paragraph are inherited by the footnote. This is typically not the desired behavior because the footnote and the paragraph are presented in spatially different regions of the document form. In contrast, if the inheritance is based on the document form, as we do here, the footnote will inherit the expected style attributes.

Box HFO

Although a Composite HFO contains all of its children, it does not specify additional spatial-temporal relations between them. Based on our experience while implementing the scenarios, we found that there are typical spatio-temporal relations, such as sequential ordering of children, which often occur. Therefore, we extended the Composite HFO with a Box HFO, which specifies additional spatial or temporal constraints on its children. The *type* attribute associated with a Box HFO denotes the type of spatio-temporal constraints, which can be ‘horizontal’, ‘vertical’, or ‘sequence’. A horizontal-box constrains its children to be ordered from left to right, whereas a vertical-box orders its children from top to bottom, and a sequence-box orders its children one after the other in time.

Root HFO

A Root HFO implements the HFO that contains all other HFOs in the document form. The Root HFO has a reference to the *delivery context* as illustrated in figure 5.6. The width, height and duration of the Root HFO are constrained by the max-width, max-height and max-duration that are imposed by the *delivery context*. Since all other HFOs are part of this HFO, these constraints are automatically applied to all descendant HFOs.

Discussion

Our HFO vocabulary is based on the document form of our extended document engineering model, which is a simplified version of the AHM (see section 4.4). In contrast, the SMIL authoring vocabulary is an extension of the AHM. As a result there are SMIL constructs that are not fully supported by our HFO vocabulary:

par/seq Temporal synchronization as modeled in SMIL is partly represented in Cuypers by the temporal containment hierarchy of hypermedia formatting objects. As a result, synchronization issues due to, for example network latency, can be automatically resolved in most cases. However, synchronization relationships between two or more orthogonal HFOs cannot be explicitly represented in our hypermedia formatting vocabulary. Consequently, if an unexpected delay occurs that disrupts this relationship, the synchronization cannot be preserved.

switch The SMIL `switch` element allows a user or player application to select one alternative out of a number of provided alternatives. For example, the `switch` element may be used to include subtitles in multiple languages. The user may then dynamically switch between subtitle languages. Cuypers generates the document form tailored for a specific delivery context, which includes the language preference of the user. As a result, the document form is generated directly including the preferred subtitle language. However, if a user wishes to dynamically switch between languages this will reformatting the entire document. The reason for this is that media items may use spatio-temporal resources differently. Since the document form is generated server-side, all required resources should be known *a priori*.

region SMIL allows an author to reuse a specified spatial region for multiple media items. Since in our model the spatio-temporal hierarchies are integrated, every media item requires a separate region (section 4.4). This is because we generate document form auto-

matically and therefore do not need the authoring convenience of reusing regions offered by SMIL.

5.2.4 Style rules

```
% (1) instance of a Presentation PS
ps([type:presentation, order:1, title:ps(...), ...])

% (2) instance of a vertical-box HFO
hfo([type:vertical-box, x1:100, x2:500, y1:0, y2:50, ...])
```

Figure 5.9: Example Prolog representations of a PS and an HFO. Attributes are denoted in a Prolog list (e.g. [...]) by key:value pairs. A key is an atomic label, whereas a value can be a composite structure. % indicates a comment and “...” indicates omissions.

The transformation from functional constructs to their corresponding form constructs is described by style rules as described in our model (see section 4.2). Existing stylesheet vocabularies, such as CSS and XSL(T), are insufficiently expressive as they do not provide support for the detection of failing style rules and compensation strategies.

In Cuypers, we use Prolog (see section 2.3.1) as the language to implement the *style rule* as it supports the required properties for a stylesheet vocabulary. Prolog, like CSS and XSL(T), is a declarative language, which allows the designer to specify the transformation while abstracting from the procedural processing steps to execute the transformation. Furthermore, the requirement that a document engineering system should detect formatting failures and automatically invoke an alternative style rule, is supported natively by Prolog’s support for backtracking during evaluation. By using Prolog we also avoid the need to invent another style and transformation language. However, on the down side, the expressive power of the Prolog language exceeds the specified requirements necessary to represent style rules. As a result the language used is more complicated than a special-purpose style language would have been.

In the remainder of this section we use a simplified Prolog notation to denote PSs, HFOs and style rules. In contrast to the previous section, which described the PS and HFO vocabularies conceptually, here we discuss the stylesheet vocabulary implemented by using example instances of PSs, HFOs and style rules. Note that these are just examples to illustrate the stylesheet vocabulary and are not meant to represent a meaningful transformation.

Figure 5.9 shows an example of a PS and an HFO represented as a Prolog term. The PS (indicated by (1) in the figure) denotes a Presentation PS, which is indicated by the type attribute. It has a title attribute with a Media PS as value that represents the title. The HFO (indicated by (2) in the figure) denotes a hypermedia formatting object that formats its children organized in a vertical-box HFO.

Figure 5.10 presents an example of two style rules represented by Prolog predicates⁸, where

⁸A Prolog *predicate* is a declarative, fallible rule that consists of a *head* and a *body*, separated by a “:-”. The input and output variables of a predicate are represented as part of the head. Variables in Prolog start with a capital. Due to its declarative nature, Prolog does not make an explicit difference between input and output

```

% style rule for Presentation PS (1)

% stylerule(+PS, -HFO)
stylerule(PS,HFO) :-
    presentation(PS),
    attribute(PS,title:TitlePS),
    \+ attribute(PS,logo:_),

    stylerule(TitlePS,TitleHFO),
    ...
    Spec = hfo(
        [
            type:vertical-box,
            children:[TitleHFO,ScenesHFO],
            ...
        ]),
    createHFO(Spec,HFO).

% alternative rule for Presentation PS (2)

stylerule(PS,HFO) :-
    presentation(PS),
    ...
    Spec = hfo(
        [
            type:temporal-box,
            children:[TitleHFO,ScenesHFO],
            ...
        ]),
    createHFO(Spec,HFO).

```

Figure 5.10: Prolog representation of fallible style rules that transforms a PS to a HFO.

PS and HFO represent the input and output variables of the `stylerule` predicate.

Analogously to the style sheet model discussed in section 4.2, we need to support a sufficiently expressive selector vocabulary to select PSs from the structured document, a sufficiently expressive descriptor vocabulary that specifies the corresponding HFOs in the document form. Furthermore, the system should be able to detect failing style rules and subsequently invoke alternatives. The implementation of these ingredients is discussed below. We conclude by discussing the depth-first selection of alternative style rules imposed by Prolog backtracking.

arguments. However, it is custom to indicate by a “+” if a variable is supposed to be used as input, and a “-” if a variable is supposed to be used as output. If a variable can be used as both input and output this is indicated by a “?”

Selector vocabulary

The selector of a style rule should be sufficiently expressive to select any subset of functional constructs from a structured document (see section 4.2). Similar to XPath in XSL(T), a style rule is applied if a given PS from the structured document satisfies the conditions as described by the selector. For example, the first style rule in figure 5.10 is applied if the given PS is a Presentation PS with an associated title but without a logo. If the PS is not a Presentation PS, in which case the `presentation(PS)` predicate fails, or the PS does have a logo, in which case the `\+ attribute(PS, logo:-)` predicate fails⁹, then the selector does not match and the style rule is not applied. Although the selector in this example is relatively simple, the full expressive power of Prolog may be used to represent more complex selectors (see section 2.3.1).

Descriptor vocabulary

Once a PS satisfies the conditions imposed by the selector, the descriptor specifies the corresponding HFO that conveys the selected PS in the document form. Recall that the descriptor vocabulary should be sufficiently expressive to create a document form with a different (tree) structure than the (tree) structure of the structured document. Analogously to the extended document engineering model, and similar to XSL(T), this is achieved by allowing the designer to specify the order in which style rules are applied, and by allowing style rules to explicitly apply style rules to the descendants of the selected functional construct.

For example, suppose a PS passes the selector of style rule (1) in figure 5.10. The next predicate `attribute(PS, title:TitlePS)` retrieves the presentation structure that represents the title of the multimedia document. The subsequent predicate `stylerule(TitlePS, TitleHFO)` transforms the presentation structure to a hypermedia formatting object. The resulting `TitleHFO` is included as the *first* child in the specified `vertical-box` HFO, independent of the position of the title in the structured document.

Style rule failure

In contrast to traditional document engineering, the execution of a style rule may fail. In Cuypers, this is implemented by the native failure and backtracking of Prolog's evaluation strategy. For example, the predicate `createHFO(Spec, HFO)` in the descriptor of the first style rule in figure 5.10 may fail due to a constraint violation. As a result, the first style rule may completely fail, in which case the Prolog interpreter will try the second, alternative style rule automatically.

Alternative style rule

When a style rule fails, the extended document engineering model specifies that the formatter should invoke the next alternative transformation rule, which is denoted by the next highest *priority* value. In Cuypers the priority of a style rule is represented by its position in the Prolog stylesheet¹⁰. For example, suppose that the first style rule, which is designed to present the title

⁹"\+" denotes negation in Prolog.

¹⁰Technically, the execution order of the style rule predicates is determined by their position in the Prolog database of loaded predicates, and this position could, in principle, be manipulated by the style rules themselves. However, in Cuypers we do not use this higher order feature. Consequently, the order in which style rules are applied corresponds in practice to their order in the stylesheet.

above the scenes, has failed due to insufficient vertical screen estate. The formatter then automatically invokes the next style rule (2), which is designed to present the title *before* the scenes. If this style rule succeeds, the transformation of the structured document is continued. Otherwise, the formatter exhaustively tries all other style rules. If all the alternatives fail, the parent style rule will also fail. As a result, the next alternative parent style rule will be invoked. This process continues until either a solution is found, or no solution is possible given the current set of style rules and the delivery context. In the latter case, Cuypers generates a minimal multimedia document informing the user of its inability to generate the requested document.

Discussion

The key design decision for the implementation of the style rules is the choice for Prolog as the language to implement the style rules. As mentioned before, the declarative nature and built-in support for backtracking make Prolog well-suited to implement our (alternative) style rules. In addition, have a general purpose programming language is convenient during prototyping, when the precise requirements for the style language are not yet known. As we will see in the next section, Prolog also has good support for constraint programming, which we will need to check if an HFO created by a style rule actually meets the constraints imposed by the delivery context. Finally, the SWI-Prolog distribution also includes libraries to deal effectively with metadata in RDF, which we will need to implement the usage scenarios discussed in the next chapter. Drawback of Prolog is the fact that few media designers will be familiar with this language, which will make it hard to advocate this approach outside the current research prototype context.

On a lower level, a disadvantage of this approach is the efficiency of the depth-first approach of backtracking over the tree of style rule alternatives. This is not always the most efficient way to resolve a failing style rule since depth-first backtracking does not consider the context in which the style rule failed. For example, consider a style rule that formats an image with a caption that is positioned below the image and an alternative style rule that formats the image without a caption. Suppose that the style rule which formats a number of images with captions has just failed because there is insufficient real-estate to present the images ordered from left-to-right. Although the removal of the captions cannot lead to a solution since it does not influence the available horizontal real-estate, the formatter will nevertheless try every possible permutation of images with and without captions to see whether it leads to a solution.

5.2.5 Summary

The vocabularies described in this section are used to represent instances of the delivery context, structured document, document form and stylesheet, which are based on the model described in the previous chapter. However, in contrast to the model, which abstracts from domain-specific and formatting-specific issues, we implemented several objects providing such specific behavior. The functionality provided by these objects is relatively generic and is used in and sufficient for the implementation of all use-cases presented in the next chapter. These objects are, however, mainly provided here as an example, and are not intended to provide a complete functional or form vocabulary implementation.

Striking differences compared to traditional text-based vocabularies are that as a consequence of the server-side formatting approach in Cuypers, the delivery context should be communicated to the server. Therefore, the vocabulary to represent the delivery context is, in contrast to the text-based model, made explicit in the Cuypers framework. In addition, the structured doc-

ument and stylesheet vocabularies are designed to cope with potential formatting failures, which are absent in traditional text-based vocabularies.

5.3 The Cuypers formatter

The Cuypers formatter transforms a structured document (i.e. PSs) to its corresponding document form (i.e. HFOs) that is adapted to a particular delivery context by executing style rules.

Recall that the style rules are represented using the Prolog language (see section 5.2.4). To facilitate the execution of style rules, the Cuypers formatter is implemented in SWI-Prolog, which is a general purpose Prolog implementation. To reduce the complexity of manipulating Prolog data structures and improve the maintainability of the software, the Cuypers formatter uses an object-oriented representation of PSs and HFOs [103]. For this we use Logtalk [109], which is an object-oriented extension of Prolog (see section 2.3.1).

In contrast to traditional document engineering, the Cuypers formatter should detect formatting failures and invoke an alternative style rule to resolve the failure. We implemented this behavior by basing the formatter on the constraint logic programming paradigm (CLP), which extends the efficient detection of constraint violations, with the automatic invocation of alternative rules (i.e. backtracking) to prove a goal used in logic programming (see section 2.3.1)¹¹.

In the remainder of this section we first provide an overview of the core formatting transformation step. Then we will focus on the implementation of constraints in Cuypers. In particular, the generation and resolution of constraints and the distribution of excess resources that may remain after satisfying the constraints. Finally, we discuss the advantages and limitations of our approach.

5.3.1 Formatting process

Figure 5.11 presents a UML state diagram, that illustrates the procedural behavior of the formatter. The start state, which is indicated by a filled black circle, represents the start of the Cuypers formatter. The termination of the formatter is indicated by the end state, which is represented by an open circle with a filled black circle in the center. The processing steps are indicated by simple and composite states, which are both represented by oval boxes with a descriptive title. Simple states have an empty body, whereas composite states contain a state diagram. A composite state terminates when its internal state diagram reaches an end state. The transition from one state to the next is indicated by an arrow. If there are multiple transitions possible, the condition to pursue a transition is indicated in square brackets.

The first (pre-processing) and last (serialize) are just syntax transformation required to get the input and output in a convenient format. The formatting decisions are taken in two steps. In the transform step an HFO tree is generated that satisfies all constraints. In the labeling step all remaining layout decisions are made by binding all constraint variables to a specific value (see upper third of figure 5.11).

¹¹In a prior version of the Cuypers formatter, we experimented with ECL/PS^c, which is software particularly dedicated to constraint logic programming. However, although the resolution of constraints is, in general, more efficient in ECL/PS^c we decided to continue development using SWI-Prolog because of its extensive libraries that facilitated integration of the software in a web environment, notably libraries dedicated to HTTP communication, XML processing and support for Semantic Web technology.

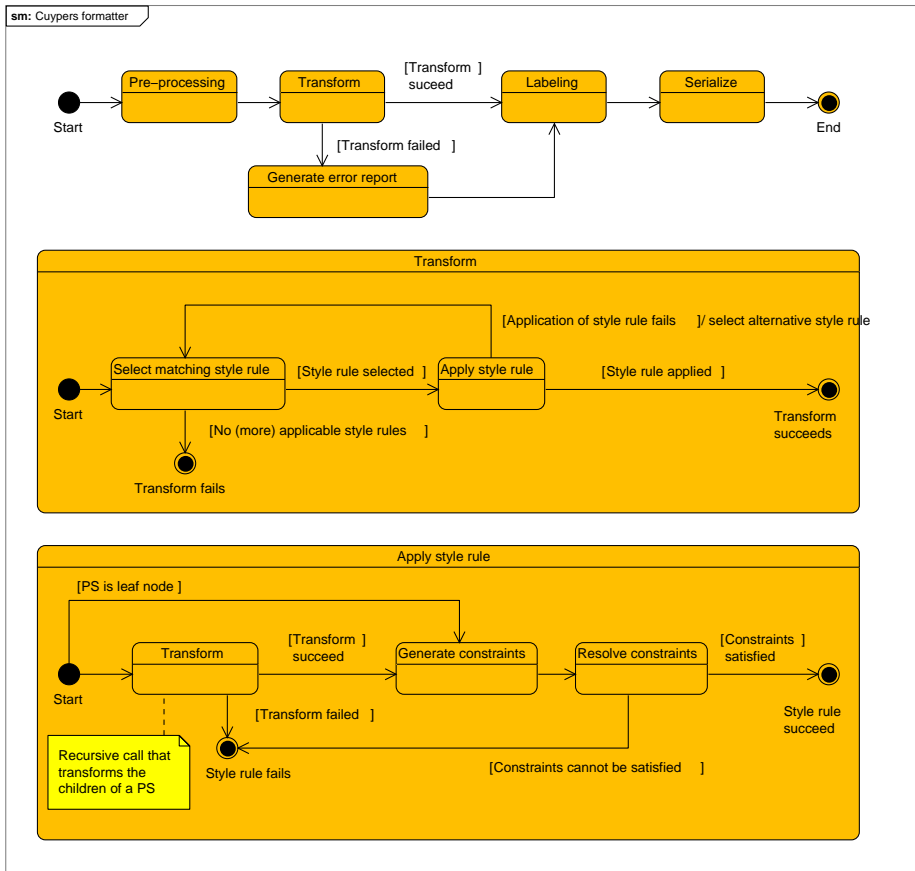


Figure 5.11: Overview of the formatting process, expressed as a UML state diagram.

pre-processing includes the transformation from the structured document and delivery context into an object-oriented representation using Logtalk. This step is a direct consequence of using Logtalk.

transform attempts to transform the structured document to its corresponding document form. It is the main step that generates all constraints, checks them and invokes alternatives constraints are violated. Note that this is a common step in every a CLP program.

Typically, this transformation produces the Root HFO and its descendants, together representing the document form. During this process, each HFO asserts its own constraints, and constraint violations cause alternative style rules to be executed. However, it is possible that, given the current structured document, stylesheet and delivery context the formatter is unable to successfully generate the document form. In this case the formatter gener-

ates an error report that informs the reader about its inability to generate the document form. Note that the error report is also represented by an HFO, therefore the result of the *transform* state is always an HFO.

The *transform* step consists of selecting and apply stylesheets (see middle of figure 5.11). *select matching style* selects one of the applicable style rule. As indicated by the style-sheet vocabulary in section 5.2.4, the Cuypers formatter selects the first applicable style rule. If there is no (more) applicable style rule, the *select matching style rules* state, and, consequently, the *transform* state, fails.

The *apply style rule* step attempts to transform a PS to a HFO by applying the previously selected style rule. Like the *transform* state, the *apply style rule* may fail. This typically is because of insufficient spatio-temporal resources but it may also fail for other reasons, such as incompatible network characteristics or conflicting user preferences. If the application of a style rule fails, the formatter backtracks to the previous *select matching style rule* state to select the next applicable style rule (if there is one). Otherwise, the *apply style rule* state and consequently the *transform* state succeeds.

The *apply style rule* process attempts to apply a give style rule to a given PS and produce its corresponding HFO (bottom third of figure 5.11).

transform attempts to transform the children of the PS to their corresponding HFO. Note that this is recursive process that ends when a PS does not have children as indicated in figure 5.11.

generate constraints imposes constraints on the resulting HFO. This includes constraints that are imposed by the delivery context and constraints that should be satisfied in order to convey the represented function of the PS through form.

resolve constraints attempts to satisfy the imposed constraints. If all constraints imposed by the style rule can be satisfied the *apply style rule* succeeds. However, if there is conflict, which prevents satisfying all constraints, the *resolve constraints* state and consequently the *apply style rule* state fail.

labeling distributes excess resources that typically remain after all constraints are satisfied (see section 2.3.1). Note that this is also a common step in every a CLP program. For spatio-temporal resources in multimedia documents, labeling corresponds to applying style attributes that specify the use of available space, such as the `fill-width` attribute that, similar to CSS, specifies that the width of an HFO should be minimized or maximized.

serialize transforms the object-oriented representation of the HFOs representing the document form into an XML format that may be further processed by the Cuypers engine (see section 5.1).

5.3.2 Resolving constraints

An HFO is represented by four three-dimensional bounding boxes: the content-box, the border-box, the padding-box and the margin-box (see section 5.2.3). The coordinates of these boxes are represented as constraint variables, which are variables with an associated domain. In contrast to Prolog variables, which do not have an explicit associated domain, the domain reduces when constraints that involve the variable are added.

Allen's qualitative relations	Quantitative constraints
A before B	$A.Z_2 < B.Z_1$
A during B	$A.Z_1 > B.Z_1, A.Z_2 < B.Z_2$
A overlaps B	$A.Z_1 < B.Z_1, A.Z_2 > B.Z_1, A.Z_2 < B.Z_2$
A meets B	$A.Z_2 = B.Z_1$
A starts B	$A.Z_1 = B.Z_1, A.Z_2 - A.Z_1 < B.Z_2 - B.Z_1$
A finishes B	$A.Z_2 = B.Z_2, A.Z_2 - A.Z_1 < B.Z_2 - B.Z_1$
A equals B	$A.Z_1 = B.Z_1, A.Z_2 = B.Z_2$

Figure 5.12: Allen's relations expressed using quantitative constraints. "A" and "B" denote a particular hypermedia formatting object, whereas "Z" denotes a particular dimension (x, y or t)

In Cuypers, we represent the coordinates of an HFO by constraint variables with integer domains. For example, initially the domain of the $X1$ coordinate of a content-box, which we denote as $X1_{content}$, is $\{-9223372036854775807 \dots 9223372036854775807\}$ ¹². Suppose that the delivery context specifies that the maximum width of the screen used to play the document is 1024. This is expressed by the constraints $X1_{content} \geq 0$ and $X1_{content} \leq 1024$. As a result, the domain of the $X1_{content}$ coordinate can be significantly reduced to $\{0 \dots 1024\}$.

Some of the constraints on the coordinates of an HFO depend on style attributes, such as margin, border and padding. For example, the relation between the $X1_{content}$ coordinate and the $X1_{border}$ coordinate expressed as a constraint is $X1_{content} = X1_{border} + \textit{Padding}_{left}$. The domain of the constraint variable $\textit{Padding}_{left}$, however, is specified by the style attributes `min-padding-left` and `max-padding-left`. For an overview of the style attributes that are relevant for the creation of an HFO see appendix A.

Besides constraints based on integer values, the domain of a constraint variable may be dynamically reduced by specifying relations between constraint variables. For example, the fact that the $X1_{content}$ coordinate is always smaller than the $X2_{content}$ may be expressed by the constraint $X1_{content} \leq X2_{content}$. If the domain of $X1_{content}$ is reduced, the domain of $X2_{content}$ is automatically reduced so that the constraint $X1_{content} \leq X2_{content}$ still holds. Similarly, if the domain of $X2_{content}$ is reduced the domain of $X1_{content}$ is automatically reduced as well.

Spatio-temporal relations between two HFOs may be expressed by inequality constraints. For example, we may express that a certain HFO, which we indicate by A , is spatially positioned left-of another HFO, which we indicate by B , by the constraint $A.X2_{margin-box} < B.X1_{margin}$. Note that content-box and padding-box are contained within the margin-box, therefore a constraint on the margin-box suffices. In a similar way, other spatio-temporal relations may be expressed. Figure 5.12 represents the spatio-temporal relations and their corresponding inequality constraints that we have implemented in Cuypers. These quantitative relations are based on

¹²Theoretically, the domain should be $\{-\infty \dots \infty\}$ but in practice the domain is typically bound by the largest possible representation of an integer.

Allen's 13 temporal relationships to denote the qualitative relationship between two objects [5]. Although Allen's relations are designed for denoting temporal relationships we use them to represent spatial relationships as well¹³.

The domains of constraint variables are always reduced, never expanded. As a result, the search space of possible assignments for a variable gets smaller, resulting in a more efficient resolution of the constraints. If the domain of a variable gets reduced to {} (the empty domain) this means that there is no satisfactory value left in the domain and, consequently, the constraint that led to the empty domain fails. Recall that the failure of a constraint is interpreted similar to a formatting failure by the Prolog engine. Consequently, the style rule that imposed the constraint fails, and the Prolog backtracking mechanism automatically invokes an alternative style rule.

Discussion: Non-arithmetic constraints

Currently constraints within the Cuypers formatter are limited to spatio-temporal constraints. However, constraints on other properties of a form construct, such as style and media items, may be possible as well. One can imagine variable font sizes depending on the screen size of the device. For example, a constraint that states that the font size of a title should be larger than the font size used for a paragraph. Another example of a non-numeric constraint may be that the dominant color of an image should match the color-scheme as defined in the stylesheet. Although, CLP is not limited to arithmetic domains, the necessary domain reduction rules are typically not readily available for such exotic domains and need to be specifically defined. Besides the challenge of identifying constraints and domains for multimedia documents, defining efficient domain reduction rules is a relatively complex task, which requires knowledge of constraint programming theory. Therefore, dynamically extending the formatter for other types of constraints, by for example a stylesheet designer, is not realistic.

5.3.3 Labeling of constraint variables

If all style rules are successfully applied this ensures that the created HFOs satisfy the imposed constraints and the document form will be presentable within the specified delivery context. However, the domains of the constraint variables that model an HFO are typically not reduced to a single value (i.e. bound), but may contain multiple values that all participate in a solution to the imposed constraints. This is a typical situation, since it is unlikely that the constraints imposed by the HFO exactly meet the constraints imposed by the delivery context. In order to generate the document form, the formatter needs to select for each constraint variable a single value from its associated domain. In constraint programming, this process is called the labeling of constraint variables. However, since the constraint variables of HFOs represent spatio-temporal resources, the labeling of constraint variables actually implements a heuristic that specifies the usage of excess spatio-temporal resources.

The usage of excess spatio-temporal resources is specified by the designer through style attributes that are associated with an HFO (see appendix A). For example, `fill-width`, which can have the value `min`, `max`, or `center`, specifies whether the width of an HFO should be minimized, maximized or balanced, respectively. Similarly, the `fill-height` and `fill-duration` style attributes specify the usage of vertical resources and temporal resources.

¹³Denoting spatial relationships by using Allen's relations results in a relatively simple set of spatial relations that is, for example, insufficiently expressive to represent rotation. However, since we currently do not support rotation in Cuypers, this simple set of relations suffices. For an overview of qualitative spatial representation and reasoning see [37].

An HFO may also use excess space for positioning. This is specified by the designer through alignment style attributes. For example, the `align-horizontal` style attribute, which can have the values `left`, `right` or `center`, positions the HFO, within its parent, to the left, the right, or in the center, respectively. Similarly, the style attributes `fill-height` and `fill-duration` align the respective HFO vertically and temporally.

Most text-based formatting vocabularies, including XSL-FO and \TeX , process formatting objects bottom-up (e.g. the child is processed before the parent). For our hypermedia formatting objects we adopt a similar approach with the exception that we first determine the size of the Root HFO. This allows us to minimize the use of spatio-temporal resources. Especially, when the multimedia document is relatively small and the spatio-temporal resources as specified by the delivery context are large, this leads to more aesthetically pleasing results.

5.3.4 Summary

The Cuypers formatter uses style rules implemented in Prolog to transform Presentation Structures (PSs, representing a structured document) to Hypermedia Formatting Objects (HFOs, representing a corresponding document form). The need for detection of constraint violations and backtracking over alternative style rules has been the key motivation for choosing a Constraint Logic Programming approach. In Cuypers, each generated HFO asserts the resources it requires by added the associated constraints to the total set of constraints. If this leads to an insolvable set, creation of the HFO fails, which automatically leads to invocation of an alternative style rule. We focus on spatio-temporal resource constraints, but also consider constraints related to network characteristics and user preferences.

In most typical situations the formatter finds a combination of style rules that satisfies the imposed constraints. For extremely constrained delivery contexts, none of the alternative style rules may lead to a satisfactory solution. Consequently, in contrast to most text-based formatters, a successful transformation cannot be guaranteed in Cuypers. However, in most cases a solution is found. In these cases, the domain of most of the constraint variables in the HFO tree has been significantly reduced. Labeling, the process of picking concrete values from these reduced domains corresponds to the freedom a designer has to distribute small amounts of space over paddings, borders and margins of the various HFOs.

5.4 Conclusion

In this chapter we discussed the Cuypers document engineering framework, which provides an implementation of the model presented in the previous chapter. The framework is based on a client/server Web architecture.

In contrast to traditional text-based formatters on the web, the Cuypers formatter produces the document form server-side. Although this has the disadvantage that it increases the load on the server, it has the advantage that standard available technology may be used to present the document form. Note that in this architecture the delivery context of the reader should be made available in a standardized format to the formatter. Although client-side support is still limited, there is progress in this direction.

In addition to the implementation of formatter, in this chapter we have presented the vocabularies implementing the stylesheet, the structured document and the document form.

The stylesheet vocabulary is implemented in Prolog as currently available stylesheet vocabularies are insufficiently expressive to detect formatting failures and invoke alternative formatting.

Since these properties are available through Prolog backtracking behavior, we represent style rules as Prolog predicates.

The functional vocabulary is represented by our presentation structure vocabulary (PS), which is used to represent the structured document. The vocabulary implements the required properties as represented in the model. However, based on our experience with the implemented scenarios presented in the next chapter, we have extended the vocabulary with a number of form conventions that are typical for multimedia documents. These include Scene PS and Group PS.

Finally, the form vocabulary is represented by our hypermedia formatting objects vocabulary (HFO). Each HFO comes with a set of constraints that formalize the resources it requires. Although the HFO vocabulary fully specifies the document form, we intentionally abstract from using a specific presentation format. An additional XSL(T) transformation transforms HFOs to a specific presentation format. This transformation is relatively straightforward, and can be used to support multiple standardized presentation formats. In Cuypers we have implemented support for SMIL 1.0, SMIL 2.0 and HTML+Time.

In the next chapter we present an evaluation of our model based on the implementation of three document engineering scenarios that use the Cuypers framework described in this chapter.

Chapter 6

Evaluation scenarios

In chapter 4, we extended the traditional document engineering model to include the transformation from structured document to document form for multimedia documents. Chapter 5 presented the document engineering framework that implements this model. In this chapter we show, by using three implemented document engineering scenarios, the application of the document engineering paradigm for multimedia documents. In addition, we show that the Cuypers formatter is sufficiently efficient to apply the document engineering paradigm in practice.

Document engineering is particularly relevant when the amount of data is too large to author manually, or when the represented data is dynamically updated, such as for news and weather forecasts on the web. These types of documents are typically automatically generated from a database. The document engineering paradigm assures that the generated document form is automatically adapted for the heterogeneous delivery context on the web. Although multimedia documents, such as SMIL, may be automatically generated, the current state of the art requires that the delivery context is known at authoring time, which is typically not the case on the web. Based on a database the Rijksmuseum uses to populate part of their website, our first scenario, ScalAR¹ automatically generates multimedia documents that are adapted for a specific delivery context.

Although ScalAR successfully generates multimedia documents from a database, this approach requires *a priori* knowledge on the structure of the database to formulate the queries. Consequently, the structure of the generated structured document is relatively static. The second scenario, SEMINF, uses a relatively flat structured database containing media items that are annotated with Dublin Core metadata. Based on the annotations, SEMINF dynamically infers relationships between media items, which are represented in the structure of the document. As a result, the structure of the structured document is variable.

The inference rules used in SEMINF are based on the Dublin Core metadata scheme. Although this scheme is relatively simple and therefore commonly used, it lacks formalized semantics and is therefore hard to process by a machine. As a result, the inferred relationships in SEMINF are relatively simple and the correctness is not guaranteed. The third scenario, DISC, uses semantic web technology to generate more sophisticated structured documents.

The first section clarifies the method we use to evaluate our model, which is based on three

¹The name “ScalAR” stands for scalable ARIA, where ARIA (Amsterdam Rijksmuseum Interactive System) refers to the multimedia database of the Rijksmuseum.

document engineering scenarios. The subsequent sections ScalAR, SEMINF and DISC elaborate on the practical implementation of these scenarios using our Cuypers framework. Performance is important on the web as the document form is typically generated on the fly. Therefore, in the performance section we evaluate the performance of the Cuypers formatter. Finally, in the conclusion section we synthesize the results presented in this chapter.

The SEMINF use case is developed in collaboration with Suzanne Little. The rules to generate discourse used in the DISC use case is work from Stefano Bocconi [26]. Furthermore, this chapter includes previously published material from [65, 98, 140, 141].

6.1 Method

To validate our model and framework we demonstrate in three use cases that the document engineering paradigm may be successfully applied to a number of multimedia documents that are representative for each use case. Successful in this context means that: firstly, a single set of style rules may be used to transform multiple structured documents. This corresponds directly to meeting the reuse requirements discussed in section 3.1.2. Secondly, the intended output is automatically adapted to the delivery context without changing the function that is conveyed. This corresponds directly to constraint requirements discussed in sections 3.1.3, 3.2.2, 3.2.3 and 3.4.2. Thirdly, the intended function, form and the transformation between the two can be expressed by the provided vocabularies. This corresponds directly to expressivity requirements discussed in sections 3.2.1, 3.3 and 3.4.1. In addition, the system implementing the model should be able to communicate with other applications using Web data formats and protocols, and should be able to import existing metadata as much as possible and also make metadata available to other applications. This corresponds directly with the architectural requirements discussed in sections 3.5.1 and 3.5.2.

We claim that the use cases show that our approach allows reuse of a single stylesheet to automatically generate document forms of sufficient quality for multiple structured documents. We do not claim, however, to successfully generate a document form for all delivery contexts. For example, delivery contexts with extremely small screens, or screens with extreme aspect-ratios are not supported. What we do claim is that for documents that need to be published for heterogeneous delivery contexts, such as the web, adaptation to many realistic contexts can be automated using our approach. In addition, we do not claim that the quality of the resulting presentations is always comparable to the quality of those produced by professional author/designers. We claim that the quality of the resulting document form is sufficient for a reader to interpret the document function as represented in the structured document, and that our approach can be applied in cases where manual design is not feasible.

Limitations

A key motivation for this thesis is the fact that current text-based document engineering models and tools are not applicable to multimedia documents. A direct practical consequence of this lack of tools is, unfortunately, that there are currently also little or no structured multimedia documents which we can use to test our approach. Instead, the scenarios in this chapter use data sets that are sufficiently rich in media and rich in metadata to allow structured documents with media content to be generated automatically. However, such data sets are also rare, and the few that do exist are often not available due to copyright issues. As a result, all data sets deployed in our scenarios will come from a single domain: cultural heritage. This is a field with

a long tradition of carefully describing, and, more recently, digitizing cultural artifacts, often resulting in richly annotated image repositories. The field also has a long tradition in sharing their knowledge with the general public, and we thank the institutes that have made their data available and allowing us to use it for research purposes.

All three scenarios presented in this chapter automatically generate structured documents from a database, and all share a single domain. However, the intellectual content (i.e. the media items and discourse structure) of each document is variable.

The media content used mainly consists of text and images, due to the lack of audiovisual media assets in the original data sets. Despite this, all resulting presentations have sufficient temporal aspects to show that similar presentations could have been generated with audiovisual content as well.

Similar to the traditional text-based model, all resulting document forms need to be validated by the stylesheet designer: we do not aim to evaluate the correctness of the multimedia style rules themselves, but rather the consistency of the results of their application².

6.2 ScalAR

The first use case, ScalAR³ (<http://www.cwi.nl/~media/demo/aria/>), generates multimedia documents based on a fixed set of relationships that are represented in an RDF graph that is stored in a Sesame repository (see section 2.2.3). The relations are converted from a relational database provided by the Rijksmuseum, which the museum uses to populate a part of its website. The data set contains 369 artists, who are associated with 783 artworks with heterogeneous dimension. Furthermore, the data set contains 702 encyclopedia entries describing concepts in the art domain, which are associated with the artworks (3472 relations).

Figure 6.1 shows a screen shot of the interface used to formulate a query and to set the properties of the delivery context. The query is composed by filling in the variables in the sentence “Tell me about X in the works of Y”. X refers here to an encyclopedia entry and Y refers to an artist, which are both selected from a pull-down menu. The values in the pull-down menus are dynamically generated using Cocoon XSP [134] server pages. This technology allows a script embedded in a page to be executed by the server once it is accessed by a client. The embedded script sends a SeRQL [3] query to a Sesame RDF repository [31] retrieving all the artists and keywords in the repository. Although RDF is designed for the web, XML technologies based on XSLT and XPath do not support querying an RDF document/repository. We therefore used a dedicated extension of XSLT for the RDF processing [143]. Once a keyword or artist is selected the input-boxes are automatically updated to present only the relevant artists or keywords⁴.

We use a web interface to manually specify the properties of the delivery context as few client-applications submit the properties of the delivery context automatically. Figure 6.1 shows parameter settings for screen size, bandwidth, user expertise and user preferences. Note that in addition to the standard screen sizes, the “other” option allows a user to specify any screen size. In addition, the requested output format of the generated document form is explicitly specified. We have implemented support for SMIL1, SMIL2 and HTML+Time. A similar interface has been used for the SEMINF and DISC use cases.

²For an overview of research in this direction we refer to [6, 101, 146].

³Parts of the ScalAR use-case are previously published in [140, 141].

⁴There is an option to select “All” for either artist or keyword which will respectively retrieve all artist or keywords.

ScalAR query interface

Topic Selection
Tell me about in the works of

Delivery context
Max. display sizes (based on device profile)

width	height
<input type="radio"/> 640	<input type="radio"/> 480
<input type="radio"/> 800	<input type="radio"/> 600
<input checked="" type="radio"/> 1024	<input checked="" type="radio"/> 768
<input type="radio"/> 1152	<input type="radio"/> 864
<input type="radio"/> 1280	<input type="radio"/> 1024
<input type="radio"/> other	<input type="radio"/> other

Bandwidth (polled from network): slow medium broadband

Level of Expertise (taken from user profile): Beginner Art historian Museum curator

Modality preference (taken from user profile): Preference for text No preference specified

SMIL preferences

Output Format

Presentations	Debug "view source" options
<input type="radio"/> SMIL 1.0	<input type="radio"/> SMIL 1.0 served as text/xml
<input checked="" type="radio"/> SMIL 2.0 (requires RealOne or better)	<input type="radio"/> SMIL 2.0 served as text/xml
<input type="radio"/> HTML+TIME (requires IE 6.0 or better)	<input type="radio"/> HFO served as text/xml

Add RDF Metadata: None (select this when using RealOne) Dublin Core only Cuypers only All

Figure 6.1: Screen shot of the interface used in Cuypers to set the delivery context

The structure of the remainder of this section follows the five steps (aggregation, normalization, formatting, serialization and standardization) from the transformation chain as presented in figure 5.2 and section 5.1.1 of the previous chapter. In the last section we synthesize the lessons learned from the ScalAR demonstrator.

6.2.1 Aggregation

During the aggregation phase the structured document is produced from the available resources. In the remainder of this section we assume the selections have been made as in figure 6.1, that is “Chiaroscuro” as the topic, “Rembrandt” as the artist, and a delivery context with a 1024x768 screen, broadband connection, low expertise level, no media preferences and SMIL 2.0 as the preferred output format.

The artworks in the ARIA database have been annotated by the museum staff with relevant encyclopedia entries, for which a textual description is available. Based on this information we can construct a scenario where the relation between an artist and an encyclopedia entry is presented by a textual description, accompanied by example artworks created by the related artist.

Figure 6.2 presents this scenario for the artist “Rembrandt” and the encyclopedia term “Chiaro-

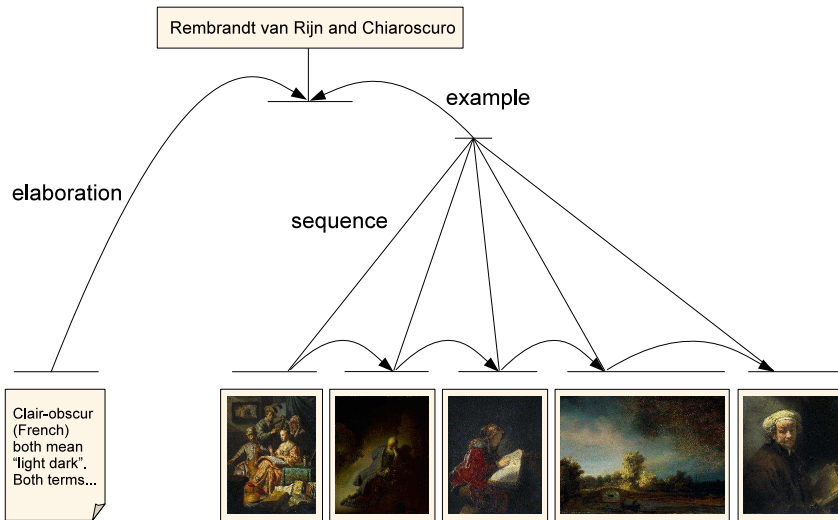


Figure 6.2: RST tree representation of a document about the artist “Rembrandt” and the encyclopedia entry “Chiaroscuro”. Media items (images and text) represent nucleus and satellites, the arrows represent rhetorical relationships between nucleus and satellites.

scuro”. The media items are represented by rectangles. The textual description of the encyclopedia term is shown on the left hand side, whereas the example artworks are shown on the right. In addition, we generate a title based on the name of the artist and the selected topic, which is shown at the top. The relations between media items are represented by rhetorical structure theory (RST) relationships (see section 2.3.2). Although RST is intended for linguistic documents, it can, to some extent, also be used to describe multimedia documents [34, 53, 97]. The media items represent the *nucleus* and *satellites*, which are RST terms to denote part of the document. In this presentation there is an *elaboration* relation between the title of the presentation and the encyclopedia description. Furthermore, there is an *example* relation between the title and the collection of images, that is a multi-nuclear *sequence* relation.

Similar to the approach used by DArt_{bio} (see section 2.3.2) we use the RST-structure as a template where the nucleus and satellites are filled in by media items relevant to the selected artist and encyclopedia entry. Note that the number of artwork images, and therefore the number of items in the multi-nuclear relationship “sequence”, is variable.

6.2.2 Normalization

During the normalization phase the aggregated structured document is transformed into a structure that may be processed by the Cuypers formatter. Typically, this concerns a relatively straightforward transformation. However, our initial implementation of ScalAR used RST relations as a structured document vocabulary, which was later replaced by our more generic Presentation Structures (PS) vocabulary (see section 6.2.6 for a discussion on this decision). Therefore, normalization in ScalAR includes a transformation from the generated RST structure

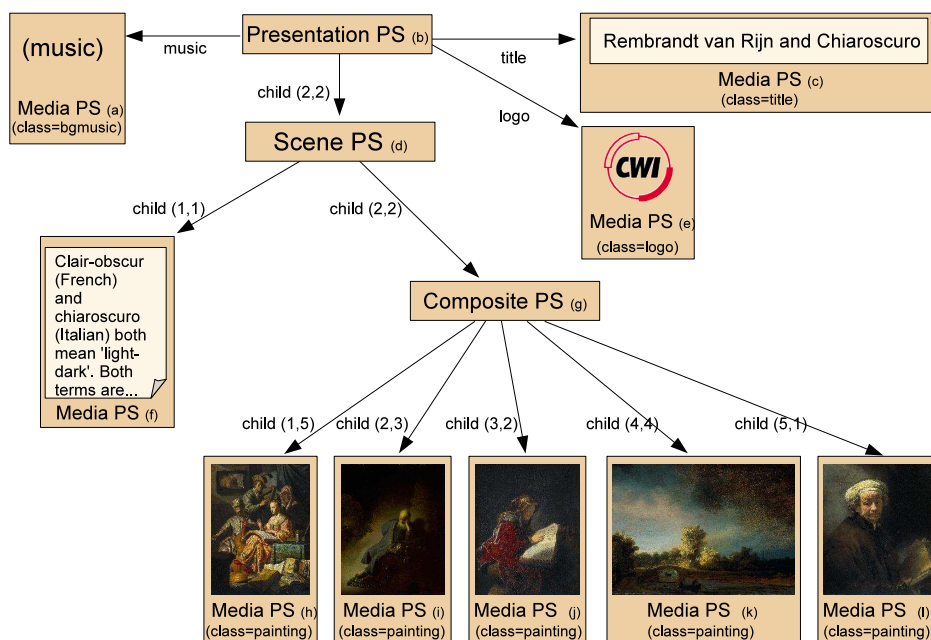


Figure 6.3: Simplified presentation structure tree for the “Rembrandt” and “Chiaroscuro” document. The boxes represent presentation structures (the letter in brackets is used for reference). The label associated with each arrow specifies the relationship between two PSs, whereas the tuple refers to the order and priority of the PS with respect to its siblings.

to the Presentation Structure (PS) format used by the formatter (see section 5.2.2).

Figure 6.3 represents the resulting structured document, where the boxes represent instances of the Presentation Structures classes defined in section 5.2.2.

If the PS is a Media PS we portray the referenced media item. Furthermore, the text in brackets refers to key-value annotations that are associated with the PS (e.g. `class=title`). The annotations we include in the figure are added at run-time when the PS is generated. Existing annotations, such as MIME type, are omitted from the figure for clarity. Similar to CSS class attributes, these annotations may be used to select the appropriate style rule from the stylesheet. The directed arrows between boxes represent relations between PSs (e.g. Media PS(f) is *child* of Scene PS(d), Presentation PS(b) has *title* Media PS(c)). The numbers in brackets associated with a relation denote, respectively, the order and priority of the PS with respect to its siblings (e.g. (3, 2) denotes the third child with second highest priority). The artworks are ordered by their date of creation, whereas the importance of the artwork is represented by the priority relation, which is based on the number of encyclopedia entries the artwork is annotated with.

In addition to the images of art objects and the textual description of the encyclopedia entry, the textual media items representing the title and captions for the artworks (not represented in figure 6.3) are generated on the fly. Furthermore, we include a media item representing the CWI logo and we generate a voice-over media item based on the encyclopedia entry (omitted from

figure 6.3 for clarity).

Recall that conceptually, the result of the normalization phase can be seen as an instantiation of a structured document in the model described in section 4.1. In addition to the structured document, the delivery context and stylesheet are the domain dependent inputs to the formatter. The delivery context is obtained through the web interface and the stylesheet is made available on server, which we describe in more detail in the next section.

6.2.3 Formatting

During the formatting phase the presentation structures (PS) are transformed to hypermedia formatting objects (HFO) by applying style rules that are made available in a stylesheet. The stylesheet, like the delivery context and structured document, may define the domain-dependent input for the formatter. Consequently, the formatter engine itself can remain domain independent.

Also recall that the same stylesheet can be applied to transform multiple structured documents. In ScalAR, the same stylesheet can thus be used to transform every structured document that can be generated based on an artist topic combination (as described in the previous section), even if the number and sized of the media items vary or the specification of the delivery context changes.

To illustrate the transformation process, figure 6.4 represents (a simplified version of) the transformation of the PS structure of figure 6.3 to the corresponding document form represented by HFOs.

As described in section 5.2.3 of the previous chapter, an HFO represents a three dimensional spatio-temporal box that is constrained by the delivery context. In contrast to text-based formatting, the style rules that implement the transformation may fail. If a style rule fails the formatter automatically invokes an alternative style rule. In the Cuypers formatter, described in section 5.3, a constraint failure corresponds to a failing predicate (i.e. style rule). Consequently, if a style rule fails, the Prolog backtracking engine automatically invokes an alternative style rule to generate alternative HFOs.

The properties of the delivery context are represented in the Root HFO, which defines the spatio-temporal constraints that should be respected by each HFO. In figure 6.4 the available real estate for an HFO is denoted by the width (w), height (h) and duration (t). Since all HFOs are contained within the Root HFO (indicated in figure 6.4 by the *child* relation) the spatio-temporal constraints imposed on the Root HFO also apply to the HFOs contained within it. In addition to width, height and duration, there are additional form properties, such as position and style, associated with an HFO. For clarity we omit them here and refer to figure 5.7 for a complete overview of the properties associated with each HFO.

With the exception of the Root HFO, which is generated by the system in a way that does not depend on the structured document or style sheet, all other HFOs are produced by applying a style rule to a PS. The bold text in figure 6.4 (e.g. Media PS (c)) denotes the *selector* of a style rule, whereas a box represents the *descriptor* of a style rule (see section 5.2.4). Since we are dealing here with instances of style rules, the selector refers to a specific PS from figure 6.3, and the descriptor refers to an instance of an HFO that conveys the selected PS.

In addition to the specification of a target HFO, the descriptor of a style rule typically invokes other style rules to recursively transform the children of the selected PS. For example, the style rule that transforms the Scene PS (d), represented in figure 6.5, specifies the application of the style rules to transform its children Media PS (f) and Composite PS (g). As with XSL(T), the designer of a style rule may choose to adapt the structure of the document form by specifying

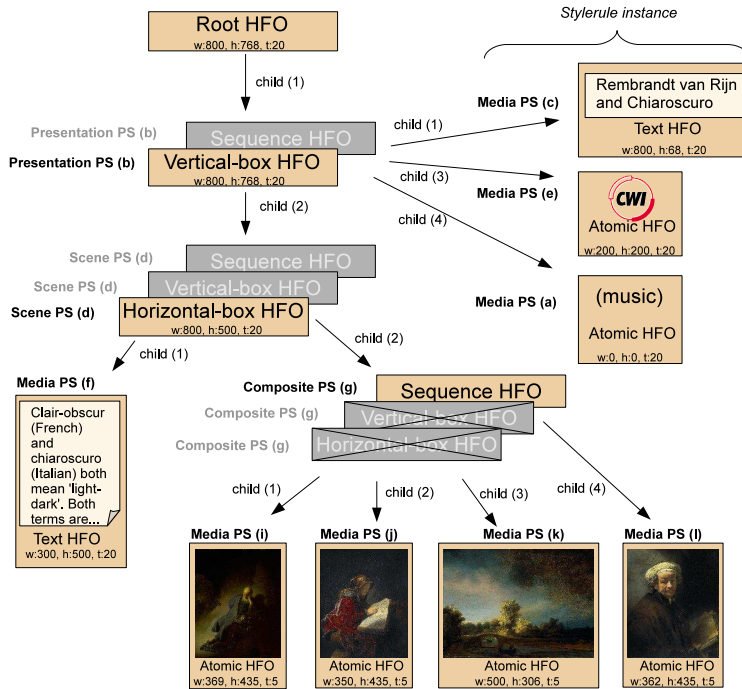


Figure 6.4: Representation of the transformation process from Presentation Structures (PS) to Hypermedia Formatting Objects (HFO). The text in bold refers to the originating presentation structure (PS) denoted in figure 6.3, whereas the boxes represent the resulting hypermedia formatting objects (HFO). Overlapping HFOs represent alternatives (the HFO selected for the document form is colored light).

the style rules that should be applied within the description of a style rule. However, in this scenario the structure is preserved. Therefore, the children of a PS map to the children of the corresponding HFOs.

Recall that the children of an HFO are contained within its parent HFO (see section 5.2.3). This is illustrated in figure 6.4 by the *child* relationship between HFOs. The number associated with this relation denotes the order of the child with respect to its siblings, which is obtained from the order of the corresponding PS.

Finally, overlapping boxes in figure 6.4 denote alternative style rules. For example, the Presentation PS (b) may be represented by a Vertical-box HFO or a Sequence HFO. If the transformation to Vertical-box HFO fails, the next alternative style rule (denoted by the stacking order in figure 6.4) is automatically invoked (through Prolog backtracking). In this scenario, there have been two failing style rules, indicated by crossed-out lines, in figure 6.4. Both concern the transformation from Composite PS (g). The first style rule attempts to format its children (i.e. the artworks) in a Horizontal-box HFO, which fails due to a lack of horizontal real estate. Similarly, the subsequent style rule fails due to a lack of vertical real estate. Finally, the third style rule

```

% style rule for Scene PS
stylerule(ScenePS,BoxHFO) :-
  scene(ScenePS), % SELECTOR: is PS a Scene PS?
  attribute(ScenePS,children:ChildrenPSs), % get children from Scene PS
  stylerule(ChildrenPSs,ChildrenHFOs), % transform children to HFO
  ...
  Spec = hfo( % DESCRIPTOR: specify HFO
    [ type:horizontal-box, % children presented left-to-right
      children:ChildrenHFOs,
      style:[x-align:min, ... ], % box is left-aligned (min=left)
      ...
    ],
    createHFO(Spec,BoxHFO). % attempt to create HFO
  ... % other/alternative stylerules

```

Figure 6.5: Example fragment of a stylesheet containing a style rule to transform a Scene PS to a horizontal-box HFO.

which presents its children in a (temporal) Sequence HFO succeeds. Note that Media PS(h) (see figure 6.3) is not included in Composite PS(g). This is because of a failing style rule due to the violation of a temporal constraint, which is resolved by omitting the least important Media PS (h) from the document form.

The designer needs to ensure that, when encoding a set of alternative style rules, that all possible alternatives still convey the intended message. For example, in Scene PS(d), the intention is that the “Clair-obscuré” text is presented to the user before the example paintings. All alternatives ensure this is the case, assuming a left to right order in the Horizontal-box HFO, and top to bottom in the Vertical-box HFO.

Figure 6.5, which represents the stylesheet, and figure 6.6, which represents the formatter, illustrate the formatting process in (pseudo) Prolog code. The `transform` predicate takes as input a structured document, which is represented by the `RootPS`, the current delivery context, represented by `DelContext` and returns the corresponding document form, which is represented by `RootHFO`. In general, the transformation recursively applies `stylerules`, which are defined in a stylesheet. However, the transformation of the `RootPS` is document independent and specified within the formatter, taking into account the constraints of the delivery context.

The children of the `RootPS` are transformed by style rules specified in a stylesheet. Figure 6.5 presents an example stylesheet containing a style rule that is used to transform a `ScenePS` to its corresponding `BoxHFO`. A similar rule may be used to transform a `RootPS`. Typically, a style rule recursively applies style rules to transform the children (`ChildrenPSs`) of the respective PS to their respective HFOs (`ChildrenHFOs`). Once the children are successfully transformed (this may fail), the `stylerule` transforms the parent into an HFO (which may also fail). In figure 6.5, `Spec` specifies a `horizontal-box` HFO that is left-aligned (`min` refers to the labeling strategy used, which we describe later). Finally, the `stylerule` attempts to create the specified HFO by invoking the `createHFO` predicate, which is defined by the formatter.

The `createHFO` predicate creates the specified HFO if there are sufficient spatio-temporal resources available. Figure 6.6) presents an example `createHFO` that is used to create a `horizontal-box` HFO. It first initializes the constraint variables representing the coordinates of the bounding box

```

% transform(+RootPS, +Delcontext -RootHFO)
% transforms a structured document to its corresponding document form
% adapted to a given delivery context
transform(RootPS, DelContext, RootHFO) :-
    stylerule(RootPS,DelContext, RootHFO), % recursively apply style rules
    labeling(extents, RootHFO),           % label extents recursively
    labeling(positions,RootHFO),         % label possitions recursively)
    serialize(xml,RootHFO).              % serialize HFO to XML

% creatHFO(+Spec, -HFO)                    % validate (or not) the constraints
                                           % imposed by a horizontal-box HFO

createHFO(Spec,HFO) :-
    attribute(type:horizontal-box,Spec), % Spec specifies a horizontal-box
    init(Spec, HFO),                    % initialize constraint variables
    ...                                  % get coordinates (cvars)
    attribute(x2:ChildA_X2,ChildA_HFO), % top-right coordinate of ChildA
    attribute(x1:ChildB_X2,ChildB_HFO), % top-left coordinate of ChildB
    ...
    ChildA_X2 #< ChildB_X1, ... .       % impose constraints (may fail)

...                                     % rules for other types of HFO

% labeling(+Type,-HFO)                    % establish position of the HFO
labeling(positions,HFO) :-
    attribute(x-1:X1,HFO),              % X1 is a constraint variable
    attribute(x-align:Strategy,HFO),    % Strategy = min|max
    label(X1,Strategy),                 % unifies X1 with a domain value
    ...                                  % according to Strategy
    attribute(children:ChildrenHFOs,HFO), % label other coordinates
    labeling(positions,ChildrenHFOs).   % label children
...                                     % similar rules for extents

```

Figure 6.6: Representation of the transformation process from PS to HFO. For simplicity of presentation we assume two children (ChildA, ChildB) in a horizontal-box. In reality the number of children is variable.

associated with the HFO. Then it invokes the constraints that are characteristic for the specified HFO. In this case the children of a `horizontal-box` HFO are presented ordered from left-to-right. Therefore, one of the constraints invoked states that the top-right coordinate of a child (e.g. ChildA) should be smaller as the top-left of its next sibling (e.g. ChildB). If this, or any other constraint specified by `createHFO`, fails, the `stylerule` that invoked `createHFO` fails as well. Consequently, the formatter backtracks and attempts to apply the next alternative style rule until, eventually, it finds a set of `stylerules` that satisfy all constraints. If the formatter fails to find a satisfactory set of style rules, which is exceptional, an error message is generated.

Once the *stylerule* that transforms the `RootPS` to its corresponding `RootHFO` succeeds, this ensures that the document form can be successfully generated. However, at this point, the document form is not yet fully established as its unlikely that the imposed constraints leave only one single solution for all constraint variables. Typically, many constraint variables still have a domain larger than one, which corresponds with excess spatio-temporal resources that allow an

HFO to “move” without violating a constraint. To fully establish the document form all constraint variables (i.e. coordinates) should be unified with exactly one value of their domain. This process is called labeling (see section 5.3.3).

In Cuypers, we label the constraint variables of an HFO by recursively traversing the HFO hierarchy two times. The first time (`labeling(extents, HFO)` in figure 6.6), the extents (i.e. width, height and duration) are established in a bottom-up fashion, meaning that the extents of a child are established before the extents of a parent. The one, debatable, exception to this rule is the Root HFO, whose extents are established first by unifying the respective constraint variables with the minimum value in their respective domains. Consequently, the document form uses available spatio-temporal resources conservatively, which we found, in general, more aesthetically pleasing. The second traversal (`labeling(positions, HFO)` in figure 6.6), establishes the position of the HFOs in a top-down fashion, meaning that the position (i.e. x, y, t coordinates) of a parent is established before the position of a child.

The lower level labeling strategy of constraint variables corresponds, on a higher level, to directives that describe how the formatter should use excess spatio-temporal resources. In Cuypers these directives are specified through style attributes that are associated with a HFO. For example, the `stylerule` in figure 6.5 that transforms a `ScenePS` specifies that the resulting `BoxHFO` should be left-aligned (e.g. `x-align:min`). The `labeling(position, HFO)` predicate illustrated in figure 6.6 applies the specified horizontal alignment, in this case using a labeling strategy that chooses minimal values for all X coordinates with domains larger than 1. See appendix A for an overview of all style attributes that affect the labeling. Once the labeling is finished the document form is completely established.

The result of the formatting phase is a completely instantiated (that is, with all layout decisions taken) document form represented by HFOs. The HFOs are represented in an object-oriented format (see section 5.3), which is passed to the serialization phase for further processing.

6.2.4 Serialization

During the serialization phase the document form is serialized from the Prolog representation used by the formatter to an XML format that can be further processed by the transformation chain. In principle, the document form could be serialized directly in a standardized presentation format, such as SMIL or HTML+TIME. However, by including an XML serialization of the HFOs, multiple presentation formats, including SMIL and HTML+TIME, can be supported by a relatively straightforward XSL(T) transformation. Figure 6.7 shows part of the serialization of the HFO tree that results from the query “Rembrandt and Chiaroscuro”.

6.2.5 Standardization

The last stage of the transformation thus concerns a transformation from the HFO representation in XML to a format that can be rendered by a third party player on the client. Figure 6.8 shows two screen shots of the RealOne Player presenting the document form that is generated by the Cuypers engine. The first shows the presentation that corresponds to the document requested by the reader in this scenario. The second shows the solution that would have been generated automatically in case the delivery context of the reader had specified a smaller screen size with a different aspect-ratio. In the second solution, alternative style rules have placed the elaborating text above the slide show of examples. As mentioned before, both solutions convey the intention

```

<hfo:root id="root-presentation-59044"
  hfo="hfo_root"
  type="root"
  hfo-subtype="root"
  title="Rembrandt Harmensz. van Rijn and Chiaroscuro"
  rights="CWI Amsterdam 2005"
  function="Root presentation with title and body"
  ...
>
  <hfo:deliveryContext
    styleSheet="cuypers"
    bandwidth="fast"
    modality="none"
    height="768"
    width="1024"
    ...
  />
  <hfo:properties
    fill-width="min"
    fill-height="min"
    fill-duration="min"
    ...
  />
  <hfo:y-box id="box-presentation-59044"
    hfo="hfo_box"
    type="composite"
    hfo-subtype="y-box"
  >
    ...
  </hfo:y-box>
  ...
</hfo:root>

```

Figure 6.7: XML serialization of the HFO tree representing “Rembrandt and Chiaroscuro” (“...” indicates omissions)

of the author to first make the concept chiaroscuro clear before presenting example artworks that illustrates it. If, however, the screen would also not have been sufficiently high to show both the elaborating text and the examples, another alternative style rule could have placed the text temporally before the images, leading to a third alternative solution (not shown in the figure).

6.2.6 Discussion

This section is based on the version of the Cuypers transformation engine at the time of writing. Before developing this version, we have developed several earlier versions, where we explored different aspects of multimedia document engineering in the context of the ScalAR use-case. All have been developed in an incremental way and each prototype can be regarded as a reaction to the lessons learned during the development of its predecessor. Below, we give a short account of these lessons learned.

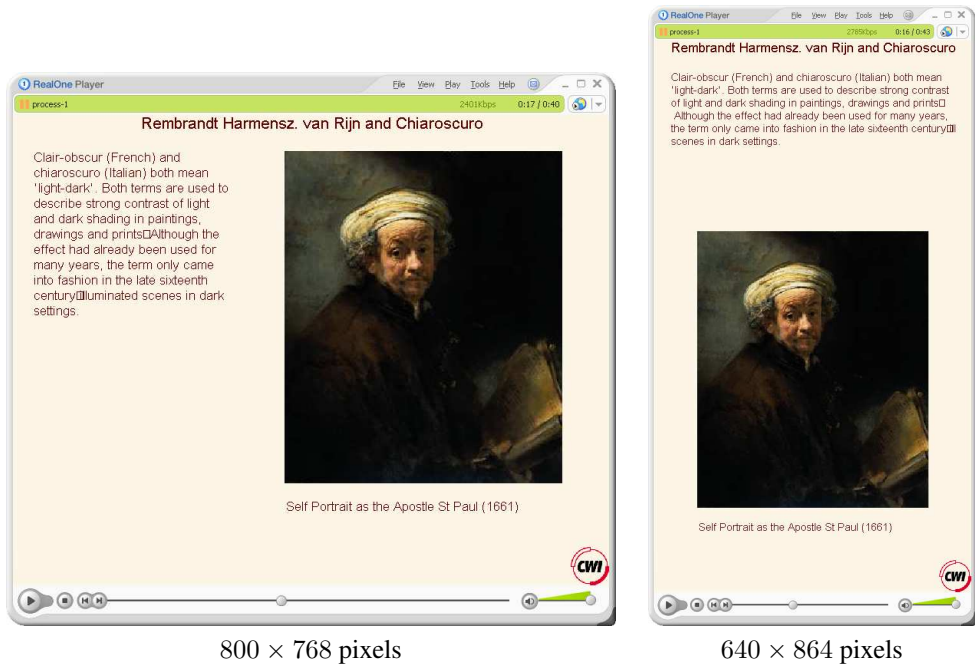


Figure 6.8: Two versions of “Rembrandt and Chiaroscuro”

Rhetorical Structure Theory insufficient as the functional vocabulary

Rhetorical Structure Theory (RST) defines a fixed set of domain independent relations that are used to describe the rhetorical structure of a document. Our initial hypothesis of using RST as functional vocabulary was based on the observation that the rhetorical structure of a text-based document is automatically preserved while the document form is adapted. Since this is currently not the case for multimedia documents, our assumption was that these relations should be made explicit in the structured document.

Although we successfully used RST in this scenario, we found that the properties of RST do not sufficiently coincide with our requirements on a functional vocabulary. Notably, requirement EXPRESS ORDER (#9.2/p.64), which states that a functional vocabulary should be sufficiently expressive to express order between functional constructs, is not met by RST. Although RST allows a nucleus to have multiple satellites, the order between sibling satellites is undefined. Furthermore, requirement EXPRESS PRIORITY (#9.3/p.64), which states that a functional vocabulary should be sufficiently expressive to denote priorities is also not satisfied by RST.

Need for a tightly integrated constraint solver

The first prototype, developed by Bailey *et al.*[127], aimed at the automatic transformation of the results of simple multimedia database queries into a multimedia presentation encoded in SMIL. The system consisted of a set of high-level transformation rules, specified in Prolog.

The detailed calculation of the exact spatial coordinates of the presentation's visual layout and the precise synchronization requirements were written in Java using an off-the-shelf constraint solver, Cassowary[16].

The problem with this approach, however, is that it is in general not possible to efficiently revise a constraint. Once the entire set of constraints has been determined, the system can either solve them or not. If it cannot, it is not known which constraint caused a failure and a complete new set of constraints needs to be generated, in the hope that the new set will resolve the previous problem. Additionally, the fact that the constraint generator (in Prolog) and the constraint solver (in Java) were two separate programs made it even harder to control the revision of constraints in a convenient way.

At the price of a slight performance penalty, we abandoned the dedicated constraint solver and successfully addressed this issue by using Constraint Logic Programming technology.

Quantitative versus qualitative constraints

A drawback we initially faced with CLP was inappropriate backtracking behavior. For example, when the domain of a specific coordinate has been reduced to, say [5..15], and the resulting layout with $x = 5$ fails for some other reason and causes the system to backtrack, the solver might try $x = 6$, $x = 7$, etc. This will, when the coordinates are expressed in pixel units, generate a number of similar layouts that differ by only one pixel value for a particular media item: clearly, in most cases this is not the desired backtrack behavior.

Instead of backtracking on the quantitative level, it is often more appropriate to backtrack on a more qualitative level, *e.g.* by backtracking over the decision that *A* should be left of *B*, by trying, for example, *A* above *B*. Typically CLP systems have no built-in solvers for qualitative constraints such as "left-of" or "above", however, they often support the definition of application-specific domains and constraints. The application needs to provide the rules which the system can use to reduce the domains of the associated variables. For example, we need a transitive rule which states that if image *A* is left of image *B* and *B* is left of image *C* then *A* is left of *C* as well. We also need a symmetric rule which states that if *A* is left of *B* then *B* is right of *A*, etc.

We experimented with qualitative constraints (implemented by CHR rules) [66] but abandoned the idea for various reasons. Firstly, the specification of such constraints is not trivial and is a programming task by itself. For example, to fully define the transitive behavior of the 13 Allen temporal relations[5], it turns out that almost 500 rules are needed (these can be generated automatically using rule generation algorithms [11]).

Secondly, since the availability of sufficient spatio-temporal resources should be ensured, a transformation to quantitative constraints is still necessary. It turned out that the performance improvement was marginal and did not outweigh the added complexity.

We found a more fruitful approach to address the inappropriate backtracking behavior by ensuring that the generated CSP conformed to a local consistency notion called arc-consistency (see section 2.3.1). This notion ensures that at labeling time all domain values of a constraint variable potentially participate in a solution. Consequently, during labeling no backtracking is needed.

On the down-side, ensuring that constraints are arc-consistent is not trivial for more domain-specific constraints. For example, the aspect-ratio constraint, which is not part of the standard constraint library, was initially implemented as an ordinary Prolog predicate. However, since the aspect-ratio could only be calculated once the respective Width and Height are known to the formatter, the aspect-ratio could fail during labeling. As a result, the formatter would enumerate (through backtracking) over all possible labellings of Width and Height until a satisfactory value

was found. To ensure arc-consistency we implemented the domain reduction algorithm for the aspect-ratio constraint ourselves.

6.2.7 Conclusion

The ScalAR scenario is successful to the extent that it applies the document engineering paradigm to multimedia documents, by automatically adapting the document form to a specific delivery context without altering the function conveyed, and by reusing the stylesheet for multiple structured documents. The PS and HFO vocabularies proved sufficiently expressive to describe the function and form of the documents in the scenario, and the input (RDF, XML) and output formats (XML, SMIL) used to communicate with the engine are all commonly accepted Web formats, with the exception of the style sheet (using Prolog).

However, the generated structured document is based on a fixed RST structure. As a result all the generated structured documents conform to this structure consisting of a single scene containing an elaborating text and a sequence of example images. In most practical applications, though, the structure of a structured document is not fixed, but might be different for each instance.

Secondly, the MIME types of the media items that are returned in response to a query are uniform and known *a priori*. In our case, the MIME type of the examples are all of MIME type `image/jpeg` and the elaborating text is of MIME type `text/plain`. However, in a typical case the MIME types of the media items used in the multimedia document are not known in advance. Therefore, it is not clear *a priori* what style rule the formatter needs to apply to transform the media items to HFOs. Furthermore, it is possible that a query returns media items of different MIME types, which is not the case for our ScalAR demonstrator.

As a result, the properties of the structured documents generated by ScalAR may not be considered representative for a generic document engineering scenario. Therefore, in the next use case, we address these issues. The structure of the structured document is dynamically generated based on the automatic inference of semantic relationships between media items, and the media repository contains media items of various MIME types.

6.3 SEMINF

The second use case, SEMINF⁵ (<http://www.cwi.nl/~media/demo/oai/>) was developed at the DSTC research institute in Brisbane Australia⁶.

The framework developed by the Open Archives Initiative (OAI) [94] facilitates interoperability between digital archives by defining a protocol that allows exchange of metadata records of the media assets in the archive. As a result, multiple archives can be queried simultaneously resulting in a cross archival search of which the results possibly reveal new information. However, when the query results are too numerous it is often difficult to manually detect the implicit relations between the results.

The SEMINF demonstrator automatically infers semantic relationships between the query results based on the Dublin Core metadata that is associated with the media items in the archive. The inferred relationships are then used to automatically generate a multimedia document that

⁵Parts of the SEMINF use-case are previously published in [98].

⁶The DSTC research institute closed in 2005.

conveys the relations to the user. Compared to the ScalAR demonstrator described in the previous section, the MIME type of the retrieved media items are unknown *a priori*. Furthermore, the structure of the structured document depends on the inferred semantic relationships and is therefore, in contrast to the ScalAR demonstrator, variable.

We use metadata from various archives, notably the Theodor Horydczak Collection (14,280 photographs)⁷, The Gottscho-Schleisner Collection (28,934 photographs)⁸ and the Library of Congress Early Motion Pictures Collection (520 videos)⁹. Furthermore, we use a subset of Wikipedia abstracts (5376) from DBpedia that are syntactically matched to OAI records¹⁰. The resulting metadata repository contains about 900,000 RDF triples.

The Dublin Core metadata associated with the media items turned out not to be suited for generic automatic processing as is. For example, most of the records reference, instead of the media item, an HTML page that contains copyright notifications in addition to the media item. Therefore, we post-processed the harvested metadata to refer directly to the respective media item. In addition, we also complement images with their spatial extents and MIME type. Finally, we transform the OAI records, which are represented in XML, to RDF. Furthermore, Dublin Core does not formally specify the values that are allowed for its metadata fields¹¹. As a result, the values found are highly heterogeneous, making it hard to define generic (string-based) matching rules to produce the relationships for our approach.

The structure of the remainder of this section follows the five steps aggregation, normalization, formatting, serialization and standardization from the transformation chain. In the last section we synthesize the lessons learned from the SEMINF demonstrator.

6.3.1 Aggregation

During the aggregation phase the structured document is generated from the available resources.

Figure 6.9 shows two media items with their associated Dublin Core metadata that are retrieved when querying for “Abraham Lincoln”. The image on the left portrays Abraham Lincoln, who is the creator of the emancipation proclamation, portrayed on the right. This relation is implicitly present in the metadata presented below the image, as Abraham Lincoln is denoted as the *dc.subject* of the left image and the *dc.creator* of the right image. The other rules that we use in SEMINF to infer simple semantic relationships are portrayed in figure 6.10.

Except for the query selection, the interface we present to the user closely resembles the interface for the ScalAR demonstrator. A user selects a query by selecting a keyword from a list of proposed subjects which are derived from the Dublin Core subject metadata field that is associated with each media item. Just like the ScalAR demonstrator the query is sent, together with the delivery context of the user, to the Cuypers engine. In the remainder of this section we assume the user has expressed interest in “Abraham Lincoln”. The system then selects media items from the repository for which either the *dc:subject*, the *dc:title* or *dc:description* fields match the query. To reduce the computational complexity we limit the size of matching media items to 100. We infer relationships according to the inference rules described in figure 6.10

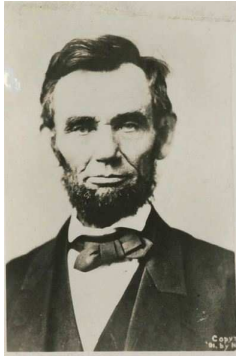
⁷<http://memory.loc.gov/ammem/thhtml/thhome.html>

⁸<http://memory.loc.gov/ammem/collections/gottscho/>

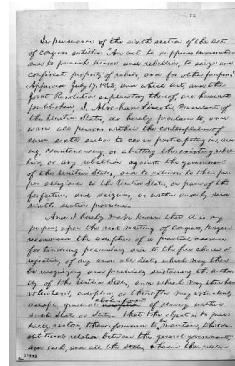
⁹http://lcweb2.loc.gov/ammem/oamh/motion_pictures.html

¹⁰<http://dbpedia.org/About>

¹¹at the time of writing, the Dublin Core specification, suggests reference values for some of the metadata fields, such as date, coverage and format. Our data sources, at the time, did not conform to these recommendations.



```
<title>Abraham Lincoln</title>
<creator>Gardner</creator>
<subject>Abraham Lincoln</subject>
<subject>President</subject>
<description>Photograph of Abraham
  Lincoln taken in Washington
</description>
<date>1862</date>
<type>photograph</type>
<type>image</type>
```



```
<title>Draft of the
  Emancipation Proclamation</title>
<creator>Abraham Lincoln</creator>
<subject>Emancipation Proclamation
</subject>
<description>Draft of the
  Emancipation Proclamation by
  President Abraham Lincoln,
  July 22, 1862
</description>
<date>22/7/1862</date>
<type>hand written document</type>
<type>image</type>
```

Figure 6.9: Example media items with associated Dublin Core metadata [66].

within the matched media items. Figure 6.11 represents a graph of matching media items and the inferred relationships between them.

The resulting set of inferred relations represents a graph structure, which does not map directly to the tree structure used in structured documents. In contrast to ScalAR, which uses a static RST tree, SEMINF does not have an inherent rhetorical structure. Consequently the generation of the structured document is not as straightforward.

To compensate for the lack of explicit higher level rhetoric, we consider in SEMINF a generated presentation as a sub-document within an interactive session with the user. A user scopes the session by selecting a media item of interest in the currently running presentation, which will be used as the *topic* of the subsequent presentation that is generated by SEMINF. This is in contrast to ScalAR, which considers a presentation as a stand-alone document. Except for the topic and the delivery context, which are passed to the subsequent presentation, from a technical perspective subsequent sub-presentations are generated independently.

To bootstrap this process, the *topic* for the first presentation is automatically inferred by selecting the media item that participates in most relations. The retrieved media items that have no direct relationship to the inferred topic are disregarded, but may (re)appear in one the subsequent

creates *Object 2 visually represents the creator of Object 1*
 IF (obj1[dc.creator] = obj2[dc.subject] AND (obj1[dc.creator] = obj2[dc.title] AND
 obj2[dc.type] = 'image') → created(obj2,obj1)

describes *Object 2 is a textual description of Object 1*
 IF (obj1[dc.title] = obj2[dc.subject] AND obj2[dc.type] = 'text')
 → describes(obj2, obj1)

depicts *Object 2 visually portrays Object 2 (e.g. a picture of Abraham Lincoln “depicts” an article about
 Abraham Lincoln)*
 IF (obj1[dc.title] = obj2[dc.subject] AND obj2[dc.type] = 'image')
 → depicts(obj2, obj1)

share context *Object 1 and Object 2 are related and have contextual information (e.g. date, coverage) in
 common*
 IF ((obj1[dc.subject] = obj2[dc.subject]) AND (obj1[dc.date] = obj2[dc.date] OR
 obj1[dc.coverage] = obj2[dc.coverage] OR obj1[dc.contributor] = obj2[dc.contributor]))
 → shareContext(obj1, obj2)

colleague of *Object 3 and Object 2 are visual representations of persons that are contributors to the same
 object*
 IF (obj1[dc.creator] = obj2[dc.subject] AND obj2[dc.type] = 'image' AND
 obj1[dc.contributor] = obj3[dc.subject] AND obj3[dc.type] = 'image')
 → colleagueOf(obj2,obj3) AND colleagueOf(obj3,obj2)¹²

Figure 6.10: The set of inference rules used in SEMINF

presentations if they correspond to the interest of the user. An alternative approach to determine the first topic is to present the user a generated HTML page listing all the resulting media items from which she selects the topic of interest. However, since the results are often numerous this approach requires additional cognitive effort from the user.

After the *topic* is determined and the relations inferred we group the relations in three groups reflecting the detail of the relation. Although this imposed grouping is relatively arbitrary, it serves to generate a variable structured document, which is the purpose of the SEMINF use case. The first group contains the `shareContext` and `shareSource` relations. Because we consider these relationships relatively generic we use them to generate the *introduction*. The second group contains the `depicts` and `describes` relations, which provide more detailed information. We use them to generate the *body* of the document. Finally, the third group contains the `colleagueOf` relationships. Since this might provide relevant related information to explore further we use it to generate the ending.

6.3.2 Normalization

During the normalization phase the aggregated structured document is transformed into presentation structures (PS) that may be processed by the Cuypers formatter.

Figure 6.12 illustrates the largest possible presentation structure that is generated in the case that all relationships have instances. However, in practice, due to the sparsity of the relations found, this is hardly ever the case. Therefore, the structure of the presentation structure is dynamically generated based on the available relationships.

As in the ScalAR demonstrator, the Presentation PS is the root of the presentation structure. The title associated with the Presentation PS is generated based on the Dublin Core title of the topic media item. Furthermore the Presentation PS consists of a maximum of three Scene PSs,

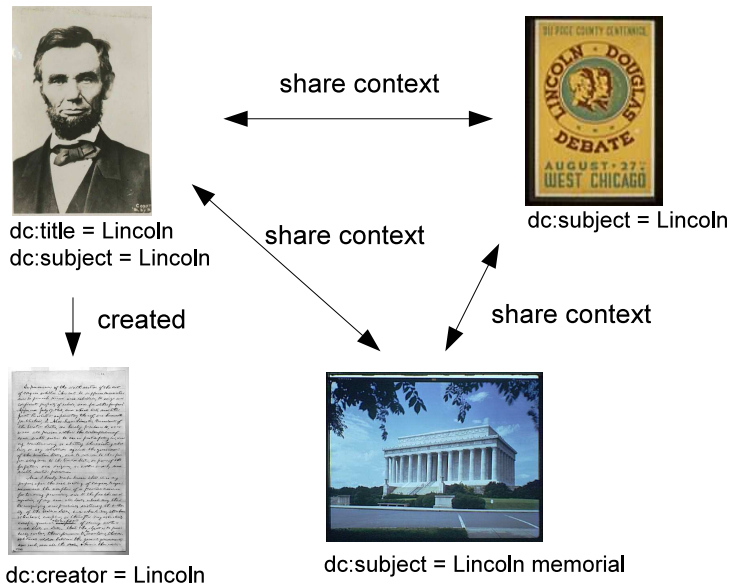


Figure 6.11: Graph representing the inferred relationships on the results of the query “Lincoln”. The top-left media item is considered the topic of the presentation as it participates in most relationships.

which represent, in sequence, an introduction to the topic, the main body and the ending. The specific function of each Scene PS is indicated by a class attribute. This is necessary to distinctively format the resulting HFO that is generated from each Scene PS. Similarly, the Composite PS that groups the inferred relationships is associated with a class attribute denoting the relation.

In contrast to ScalAR, the MIME type of most of the media items are unknown *a priori*. This is important because media items with arbitrary MIME types may cognitively overload the user when presented together, such as multiple videos or audio clips in parallel. Consequently, the formatter should determine at run-time what media items can be presented together without cognitively overloading the user. For this we use a Group PS, which is a special type of Composite PS specifically designed to convey grouping for media items with arbitrary MIME types (see section 5.2.2). In the following section we explain in more detail the formatting of a Groups PS.

6.3.3 Formatting

The style rules that transform this presentation structure to a hypermedia formatting object tree are mostly reused from the stylesheet we used for the ScalAR demonstrator. However, SEMINF uses a Group PS that is absent in ScalAR. Therefore we describe the transformation from Group PS to HFO here in more detail.

The Group PS conveys multiple media items, which have heterogeneous MIME types, as a

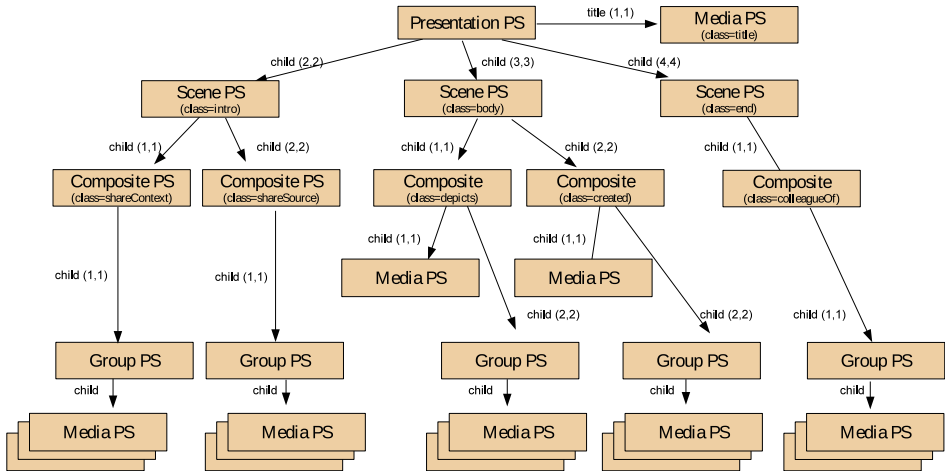


Figure 6.12: Presentation structure for the SEMINF use case

group without cognitively overloading the user. For this it uses modality knowledge that may be associated with each media item [23]. However, typically modality knowledge is unknown for the retrieved media items. Therefore, we use the associated MIME type to guess the modalities of a media item. For example, the modalities of a JPEG image are typically *analogue*, *arbitrary*, *static* and *graphical*, which defines a class of media items that includes photographs, paintings and artistic drawings, whereas MP3 audio files typically are *analogue*, *arbitrary*, *dynamic* and *acoustic* denoting a class of media items that represent music (see section 2.2.2 for an explanation of these modalities). Although this works satisfactorily for most cases, there are exceptions. For example, the query “Abraham Lincoln” returns a media item, that represents a digital scan of the emancipation proclamation written by Lincoln. However, since the MIME type of the media item is `image/jpeg` the media item is incorrectly associated with the modalities representing a realistic image. Consequently, the wrong style rule is applied resulting in a document form that reserves insufficient time for a reader to read both textual media items.

Figure 6.13 presents example style rules that convey grouping of media items. Every style rule expects exactly two PSs, namely `MediaA` and `MediaB` and produces a HFO that conveys the Group PS. Note that a PS may, besides a Media PS, be a Composite PS, which is not *a priori* associated with modalities. In this case the modalities of the Composite PS are dynamically aggregated by collecting the modalities of its children. Also, note that a Group PS often has more than two children whereas the style rules to transform a Group PS only accounts for two children. If a Group PS has more than two children, we recursively generate a Group PS containing a child from the original Group PS and the recursively generated Group PS containing the remaining children. The advantage of this approach is that a stylesheet designer only needs to specify a limited number of style rules to account for all possible combinations of modalities. The disadvantage of this approach is that the resulting HFO does not necessarily presents its children in the most optimal combination. If more optimized combinations are desired, one may consider extending the scope of a style rule. However, the number of style rules to cover all combinations

```

/*
  Modalities are represented by:
  li (linguistic)      - based on existing syntactic-semantic systems of meaning.
  an (analogue)       - complementary to linguistic modalities.
  ar (arbitrary)      - rely on an existing system of meaning.
  nar (non-arbitrary) - accompanied with appropriate representational conventions.
  st (static)         - modalities may be decoded by a user at any time.
  dy (dynamic)        - no freedom of inspection.
  gr (graphical)     - perceived by the visual sense.
  ac (acoustic)      - perceived by the hearing sense.
*/

% stylerule(+MediaA, +MediaB, -HFO)
(1) stylerule(MediaA, MediaB, HFO) :-
    modality(MediaA,[li, nar, st, gr]), % is MediaA a textual label?
    modality(MediaB,[li, ar, st, gr]), % is MediaB a narrative text?
    !, % yes, then ...
    stylerule(MediaA,HFOA), % transform MediaA to HFOA
    stylerule(MediaA,HFOB), % transform MediaB to HFOB
    HFO = ybox(HFOA, HFOB). % present HFOA above HFOB

(2) stylerule(MediaA, MediaB, HFO) :-
    modality(MediaA,[li, nar, st, gr]), % is MediaA a textual label?
    modality(MediaB,[an, ar, st, gr]), % is MediaB a picture?
    !, % yes then ...
    ...
    HFO = ybox(HFOB, HFOA). % present HFOB above HFOA
...

% Default style rules:

(3) stylerule(MediaA, MediaB, HFO) :- % are both media items acoustic?
    modality(MediaA,[ac]),
    modality(MediaB,[ac]),
    !, % yes then ...
    ...
    HFO = tbox(HFOA, HFOB). % present in sequence

(4) stylerule(MediaA, MediaB, HFO) :- % one media item is graphical
    % the other acoustic?
    modality(MediaA,[gr]),
    modality(MediaB,[ac]),
    !,
    ... % yes then ...
    HFO = xbox(HFOA, HFOB). % present HFOA and HFOB in parallel
...

```

Figure 6.13: Simplified (for readability) Prolog representation of style rules that are used to transform a Group PS, which groups media items with heterogeneous modalities. (“...” indicates omissions, numbers in brackets are used for future reference).

grows exponentially with the number of children.

Style rule 1 transforms a Group PS that contains two MediaPSs, `MediaA` and `MediaB` to an HFO. It first checks whether the first MediaPS (`MediaA`) is associated with the modalities *linguistic*, *non-arbitrary*, *static* and *graphical*, which denotes textual media items that should be associated with another media item, such as titles, captions and labels. Then, it checks whether the second MediaPS (`MediaB`) is associated with the modalities *linguistic*, *arbitrary*, *static* and *graphical*, which denotes textual media items that have a narrative. If both succeed then `MediaA` and `MediaB` are transformed to their corresponding HFOs. Finally, HFOA (title) is presented above HFOB (narrative text). Compare this to style rule 2, which presents a media item representing a caption, label or title (`MediaA`) below a media item that represents a photograph, painting or artistic drawing.

Every media item is at least associated with, either a *graphical* modality, or an *acoustic* modality (we ignore the *haptic* modality for practical reasons). Consequently, all combinations of two Media PSs can be expressed by four default style rules. Style rules 3 and 4 present two of the four default style rules that are executed if no other style rules applies¹³.

6.3.4 Serialization

During the serialization phase the document form is serialized from the Prolog representation used by the formatter to an XML format that can be further processed by the transformation-chain. Because there is no significant difference compared to the serialization phase of the ScalAR demonstrator, we omit a detailed description here.

6.3.5 Standardization

The last stage of the transformation concerns the transformation from the HFO representation in XML to a format that can be rendered by the third party player of the client. Because there is no significant difference compared to the standardization phase of the ScalAR demonstrator, we omit a detailed description here.

6.3.6 Discussion

Figure 6.14 presents a screen shot of a SMIL presentation that is generated by the SEMINF demonstrator. The screen shot presents the body of the presentation conveying the `depicts` relationship that is generated based on the query “Abraham Lincoln”. The `describes` relationship is portrayed in a similar way, although the background color is adapted to reflect a different relationship. The presentation consists further of an introduction, which is represented by a sequence of images that have a `shareContext` or a `shareSource` relationship with the topic. Because no `colleagueOf` relationship could be inferred this presentation has no ending.

In contrast to ScalAR, which generates a structured document on the basis of a static template, the SEMINF preprocessor generates the structured document dynamically. Consequently, the structure of the structured document may vary, which is typical for a document engineering

¹³Since the scope of a style rule is variable it may happen that multiple style rules can be applied to transform a Group PS. However, in general this is undesirable, as only the most specific rule should be applied. In Prolog this behavior is achieved by organizing the style rules from specific to generic. Since the Prolog interpreter applies rules in a depth-first fashion the first applicable style rule is the most specific one. To prevent undesirable backtracking to other matching rules, the cut operator (!) is used, which eliminates all choice-points within the scope of the matched rule.

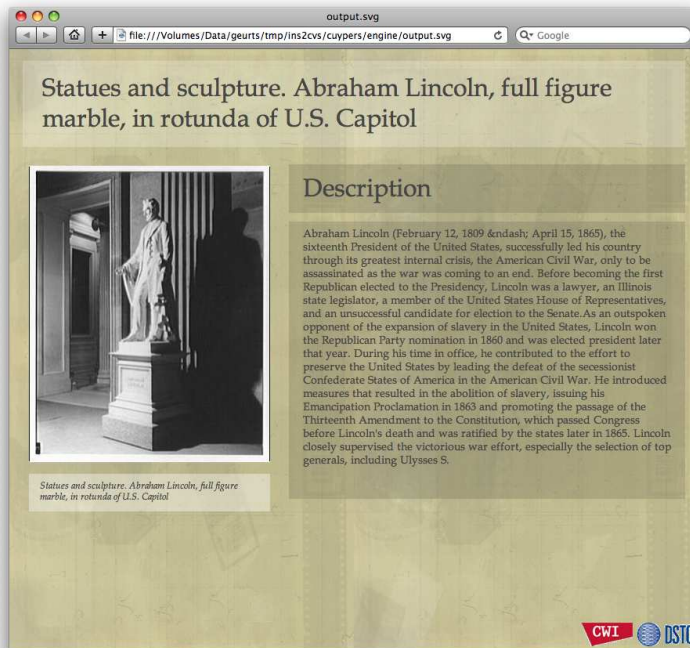


Figure 6.14: Screen shot of Safari playing a SVG document generated using the SEMINF demonstrator.

scenario. This allowed us to test the transformation rules in a more dynamic setting. The variability of the media items required an extension of the rules to handle combinations of items of which the modality is not known a priori.

Table 6.1 presents the number of relations that are inferred for each query. The table shows that the variety in inferred relationships is sparse. As a result, the generated structured documents are in practice relatively homogeneous, when compared with the ScalAR use-case. The reasons for this lack of diversity can be traced back to the quality of the metadata used to infer the relationships.

Firstly, the application of inference rules is hampered by unstructured metadata values and/or the use of incompatible schemas for many of the element values. For example, some data providers use textual values for the *dc.relation*, *dc.identifier* and *dc.source* elements, while others use URIs. This makes comparisons between values problematic. On the other hand, if particular controlled vocabularies or schemes are used for certain Dublin Core elements, it may not be applicable to migrate these values to other Dublin Core elements in order to infer semantically-related resources.

Secondly, there is significant ambiguity over the purpose and content of many of the Dublin Core element's values. For example, in some cases the 'creator' is the creator of the digital

query	#creates	#describes	#depicts	#source of	#share source	#version of	#share context	#colleague of
Lincoln	0	4	0	0	0	2	12	0
Monuments	0	6	0	0	0	2	42	0
Presidents	0	0	0	0	0	0	36	0
Sculpture	0	4	0	0	0	0	36	0
statue	0	6	0	0	0	2	42	0
Washington	0	0	0	0	0	2	54	0
Horydczak	0	0	0	0	0	1	54	0
Gottscho	0	0	0	0	0	2	6	0

Table 6.1: Number of inferred relations of the SEMINF demonstrator executing variable queries (screen size is 1024×768)

media item. In other cases, the ‘creator’ is the person who created the object depicted in the media item. In some archives, the ‘creator’ of a musical recording is the composer, while in others, the ‘creator’ is the primary musician. Likewise the *dc.date* value may be the date of creation of the object depicted by the digital media item or even the date of publishing in the OAI archive.

SEMINF shows that the Cuypers formatter successfully processes structured documents that consist of media items of multiple MIME types. However, the variety of generated structured documents is still relatively low due to the lack of formalized annotations. Therefore, the next demonstrator DISC, uses formalized semantic web based knowledge and technology to infer more detailed semantic relations and, as a result, generate a more complex structured document.

6.4 DISC

Recall that the properties of the generated structured document in our prior use cases were unsatisfactory to validate the reuse of a stylesheet for variable structured documents. Notably, the structured document generated by ScalAR was insufficiently dynamic as only the media items were variable, whereas the structured document generated by SEMINF turned out to be too sparse due to lack of formalized metadata. Therefore, the main purpose of the DISC use case¹⁴ is to generate a relatively large and heterogeneous structured document to validate the reuse of a stylesheet for variable structured documents¹⁵.

In contrast to the metadata used in SEMINF, which was mainly intended for humans, the semantic web specifically aims at providing metadata that is both accessible for humans as well as machines. As a result, automatic inferences are more reliable and therefore complex structured documents are relatively easy to generate (see section 6.5.3 for a qualitative comparison of code complexity for our three use cases).

¹⁴<http://www.cwi.nl/~media/demo/disc/>

¹⁵Parts of the DISC use-case are previously published in [26, 65].

For this demonstrator we used the RDF repository that was developed for the Topia project¹⁶ in combination with the same Rijksmuseum database we used for ScalAR. Because this repository is about art and artists we generate biography-like multimedia documents. The repository contains information about 369 artists and 783 artifacts. For a limited number of artists we complemented the available information with personal relationships, such as spouses and children. The repository is represented in RDF and consists in total of 71,797 triples.

Similar to the prior use cases, the structure of the remainder of this section follows the five steps (aggregation, normalization, formatting, serialization and standardization) from the transformation chain. However, since the normalization, serialization and formatting phase are similar to prior use cases, we omit a detailed description here. This leaves aggregation, which elaborates on the method used to generate the structured document, and standardization, which presents an illustrative example of the generated document form. Finally, in the last section we synthesize the lessons learned from the DISC demonstrator.

6.4.1 Aggregation

Using a web-form, similar to the ones used for ScalAR and SEMINF, a user selects one of the 369 artists and a document genre, such as personal biography or professional biography. The document genre is used to determine the rule-set we use to generate the document. A reference to the topic and the rule-set is sent, together with the properties of the delivery context, to the Cuypers engine.

The rules in DISC are inspired by the communications models developed by Greimas [70], which select relevant information from a repository based on a given context. The rules are grouped in *narrative units* representing a unit of related information that should be presented together. Table 6.2 presents an example rule-set that we use to generate a biography of an artist. The rule-set consists of four narrative units, ‘Biography’, ‘Personal Information’, ‘Career’, and ‘Artefact’, containing in total 11 rules, numbered 1 to 11. At the time of implementation, no standard to describe rules on the web was available, so we choose to represent our rules in RDF¹⁷.

The rules use the concept of an *actor*, which denotes the topic within the scope of the narrative unit and a *role*, which signifies the relevant domain properties related to the actor in this context. For example, suppose the topic of the biography is Rembrandt. In this case Rembrandt is an actor in the role of ‘main character’ (‘main’ in table 6.2). For the section on personal information, rule #3 states that the relevant domain properties (‘Props.’ in table 6.2) are `name`, `birth-year` and `death-year`. However, if the biography is instead about Rembrandt’s wife, Saskia, then ‘Rembrandt’ can still be an *actor*, but now with the *role* of ‘spouse’. Rule #6 states that the relevant domain properties are then: `name` and `marriage-date`. If multiple rules match an actor and role in a narrative unit, such as rules #1 and #2, then they are executed in sequence, in order of occurrence.

The domain specific information associated with a domain property is passed on as a class attribute value in the PS. This makes the PS data structure itself independent from the domain, but the style rules may still use this information to create a HFO that conveys the domain semantics

¹⁶see <http://db.cwi.nl/projecten/project.php4?prjnr=147>

¹⁷The W3C Rule Interchange Format (RIF) is chartered to produce a core rule language plus extensions which together allow rules to be translated between rule languages and thus transferred between rule systems (see <http://www.w3.org/2005/rules/wg/charter.html>). As of 22 September 2008 an editor’s draft is available.

Biography (Presentation PS)						
#	Role	Relation	Progression	New role	Props.	PS
1	main	<i>none</i>	Personal Information	main	<i>none</i>	Scene PS
2	main	<i>none</i>	Career	main	<i>none</i>	Scene PS
Personal Information (Scene PS)						
#	Role	Relation	Progression	New role	Props.	PS
3	main	<i>none</i>	<i>none</i>	<i>none</i>	name, birth-year, death-year	Group PS
4	main	depicted-in	Artefact	illustration	<i>none</i>	Alt. PS
5	main	<i>none</i>	<i>none</i>	<i>none</i>	description	Alt. PS
6	spouse	<i>none</i>	<i>none</i>	<i>none</i>	name, marriage-date	Group PS
Career (Scene PS)						
#	Role	Relation	Progression	New role	Props.	PS
7	main	creator	Artefact	main	<i>none</i>	Comp. PS
Arte-fact (Composite PS)						
#	Role	Relation	Progression	New role	Props.	PS
8	caption	<i>none</i>	<i>none</i>	<i>none</i>	date, title, material, size	Group PS
9	main	<i>none</i>	Artefact	caption	image	Group PS
10	main	<i>none</i>	<i>none</i>	<i>none</i>	description	Alt. PS
11	illustration	<i>none</i>	<i>none</i>	<i>none</i>	image, title	Group PS

Table 6.2: Narrative units and their associated rules.

as the stylesheet designer sees fit.

The *role* (e.g. ‘main character’) and *narrative unit* (e.g. ‘private life’) are included as class attributes in the corresponding presentation structure. This way, the style rules can adapt the styling of the HFO representing the PS based on its specific function. For example, the rule that selects the domain properties `birth-year` and `death-year` for ‘Rembrandt’ in the role of ‘main character’ is associated with the presentation structure Group PS, which expresses that both properties should be presented as a unity.

The Alternative PS allows an author to indicate alternative presentation structures to convey a particular function. This is typically used when multiple versions of essentially the same media item are available, such as images with different resolutions or videos using different encoding schemes. For example, rule #4 in table 6.2 selects an illustration that portrays the main

character. However, there may be multiple illustrations with potentially different resolutions available. The Alternative PS allows to postpone the decision as to which illustration to select until the properties of the delivery context are known. The Cuypers formatter does not distinguish between alternatives provided by the author and alternative style rules provided by the designer.

In addition to selecting relevant domain properties, a rule can also trigger the execution of rules in another narrative unit, producing a progression of the biography. For example, rule #1 in table 6.2 states that if the actor plays the role of ‘main character’ then the biography progresses with the ‘Personal Info’ narrative unit. In this case the actor and its role remain the same. However, rule #4, which additionally specifies a relation `depicted-in`, changes the actor to the object of the `depicted-in` relationship (provided that the current actor is associated with an object through a ‘depicted-in’ relationship). For example, if the current actor Rembrandt has a `depicted-in` relationship with an object depicting Rembrandt, then the biography progresses with the narrative unit ‘Artefact’ where the current actor is the object depicting Rembrandt in the role of ‘illustration’.

Finally, a narrative unit is associated with a presentation structure, which conveys the function of the narrative unit in the multimedia document. For example, the narrative unit ‘Personal Info’ is associated with the presentation structure Scene PS.

6.4.2 Normalization

During the normalization phase the aggregated structured document is transformed into the presentation structures (PS) that may be processed by the Cuypers formatter. Since the aggregation phase already generates these directly, no additional transformation is necessary for DISC.

6.4.3 Formatting

During the formatting phase the PS representing the structured document is transformed to its corresponding HFO. Since the formatting phase is discussed in prior use-cases we omit a detailed description here.

6.4.4 Serialization

During the serialization phase the document form is serialized from the Prolog representation used by the formatter to an XML format that can be further processed by the transformation-chain. Because there is no significant difference compared to the serialization phase of the ScalAR (and SEMINF) demonstrator, we omit a detailed description here.

6.4.5 Standardization

The last stage of the transformation transforms the HFO representation in XML to a format that can be rendered by the third party player of the client.

Figure 6.15 presents a screen shot of the final SMIL presentation that is generated by the DISC demonstrator. The screen shot presents a scene from a biography of Rembrandt related to his work.

6.4.6 Discussion

Below, we give a short account of the lessons learned during the development of DISC.

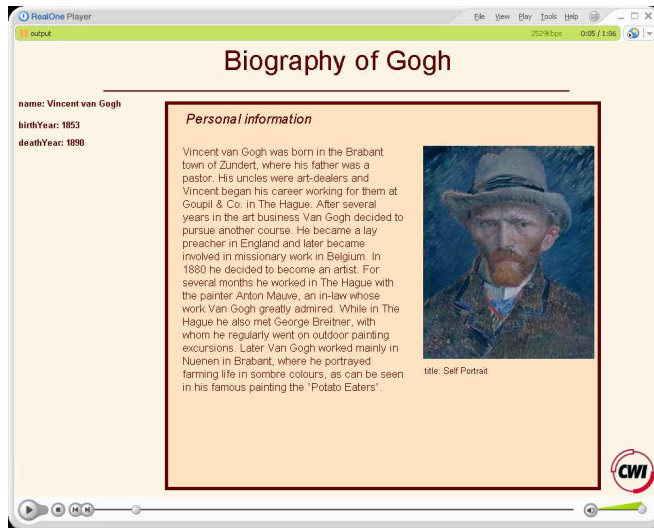


Figure 6.15: Screen shot of RealPlayer playing a biography of Van Gogh generated using the DISC demonstrator. The screen shot shows a fragment of the “Personal Information” scene, which is followed by a scene on the professional “Career” of van Gogh.

Impact of complex structured documents on the Cuypers formatter

In contrast to the prior use-cases *ScalAR* and *SEMINF*, *DISC* generates a dynamic (whereas *ScalAR* was static) and relatively large (whereas *SEMINF* is small) structured document. As a result, the formatting phase may be considered more challenging than the prior use-cases. Table 6.3 shows statistics gathered for variable queries and a static delivery context using the *DISC* demonstrator. The #PS column denotes the number of PSs in the generated structured document.

Furthermore, #HFO denotes the number of HFOs in the document form. In general, the number of PSs is indicative of the number of HFOs, as in most cases a PS is transformed to a single HFO. However, there are exceptions. For example, the formatter may decide to suppress the transformation from a PS with low priority in case there are insufficient spatio-temporal resources. Or, an author may have specified an Alternative PS to indicate that, for example, an image is available in multiple resolutions. Because the formatter selects one of the potential alternatives the number of HFOs may be less than the number of PSs.

The third column denotes both the total number of potential solutions and the number of potential solutions tried before the formatter was successful. The total number of solution is relatively heterogeneous, which is an indication that the structured document is heterogeneous. In contrast, the number of solutions tried is relatively low, which means the constraints imposed by the delivery context are relatively light and a solution could be found easily. This is also reflected in the last column, which denotes the formatting time.

Although in section 6.5 we discuss the performance of the system in more detail, we can already observe that the formatting time seems to be directly related to the number of alternative

screen size	#PS	#HFO	#tried/ #total	time (sec)
Bol	42	42	3/864	6.74
Goltzius	38	38	5/288	8.57
Steen	42	40	9/864	9.97
Mostaert	31	31	9/216	10.12
Gogh	35	35	9/648	10.87
Rembrandt	51	40	9/2592	12.08
Terborch	38	36	9/288	14.03

Table 6.3: Statistics of the DISC demonstrator executing variable queries (screen size is 1024×768)

solutions tried. Therefore, the scalability boundaries of the Cuypers formatter are due to the number of alternative solutions tried rather than the complexity of the structured document.

Domain semantics at relatively low costs

Previously described presentation generation systems, such as Artequakt [89] and DArt_{bio} [18] are adapted to present information within a particular domain (see section 2.3.2). These systems have the advantage that they are adapted to the specific context of the application. Other presentation systems, such as Noadster[128] and Autofocus, present information independently of domain semantics (see section 2.2.3). These systems have the advantage that they can be applied to a large range of application contexts.

In software engineering, domain dependencies are often avoided when possible because they reduce the potential application domain of the system. Compare this to document engineering, which attempts to identify domain dependencies and describe them explicitly in a structure document or stylesheet. By adapting the stylesheet or structured document a document engineering system can be adapted to a specific domain at relatively low costs.

The DISC system follows the document engineering principle by representing relevant domain relationships, and the way they should be conveyed, in rules that are specified externally to the system. By modifying the rules the DISC system can be adapted, at relatively low cost, to convey particular domain semantics for other application contexts.

Designing form conventions for complex structured documents

Similar to text-based documents a complex structured document is typically used to represent a fine grained discourse structure. For a reader to grasp the details of the discourse it is important that form conventions used to convey these details are properly represented. One of the principles a designer uses to achieve this are *repetition and consistency*, which states that objects with a similar function should be presented similarly. Within the Cuypers formatter we can respect this principle for a group of PSs that share a parent because the resulting HFOs are explicitly grouped (i.e. contained) within their parent. As a result we may impose consistency constraints on the children when the parent is created. In contrast, the group of PSs that do not share a parent are not explicitly represented in the containment hierarchy of HFOs in the document form. Consequently, it is impossible to impose explicit constraints on the members of such an

orthogonal group (see section 4.4.2 on page 84). Within the DISC demonstrator this deficiency becomes apparent. For example, the formatter has alternative rules that position a description either below or to the right of a figure. Depending on the available real-estate the formatter decides the position of the description on a case by case basis. Since there are often multiple pairs of figure and description the position of the description in the final presentation seems random and gives a chaotic impression. Worse, if the position of the description is homogeneous except for an incidental exception, a reader most likely will interpret the exception as semantically significant, which was not intended by the author.

The simplistic solution is to disallow alternative positioning for figure description pairs, which will ensure consistent presentation (if possible). However, this decreases the adaptability of the multimedia document, which is undesirable. In addition, a completely consistent design is typically not completely satisfactory either. A designer of a multimedia document may vary the theme of the form convention to keep the interest of the user. Nevertheless, the form conventions should be sufficiently similar to allow the reader to recognize them as being the same. To support such features, explicit grouping of arbitrary HFOs should be possible. Furthermore, the formatter should be able to infer whether the document function is satisfactorily represented, both, on the level of individual media items as well as in the context of a multimedia document as a whole.

We conclude that the Cuypers formatter successfully transforms the complex and relatively large structured documents generated by DISC. However, to ensure that structured documents that exceed this complexity are properly represented by the resulting document form, more advanced grouping and constraints mechanism are necessary.

6.5 Performance analysis

Performance of the formatter is important in a web context where the document form is typically generated on the fly. Therefore, in this section we present the performance analysis of our Cuypers formatter applying the document engineering paradigm.

Because the influence of the delivery context on the formatting process is similar in all three use cases, we focus our analysis of the performance under different delivery contexts on the ScalAR use case. Since the DISC demonstrator generates more heterogeneous structured documents compared to the prior demonstrators, we use the DISC demonstrator to analyze the performance under the reuse of style. A complete overview of performance measurements for all three demonstrators is given in appendix B.

The machine we used to take these measurements runs on a Intel Pentium 1.70GHz processor, with 1GB of RAM memory. The operating system is Linux (Fedora Core 5) with a 2.6.20 kernel¹⁸.

6.5.1 Automatic adaptation to the delivery context

The first set of measurements uses a single structured document and a variable delivery context. Table 6.4 lists performance measurements of the ScalAR demonstrator, executing the query

¹⁸The reported times are doubled when using the on-line demonstrator. This is because the browser application requesting the document typically does not have built-in support for SMIL but needs a helper application to render the document. Once it receives the document and recognizes the unsupported document format (e.g. SMIL) it launches the helper application (e.g. RealOne Player), which is passed the URL of the document. The helper application then requests the document again before it gets rendered.

screen size	aggregation	normalization	formatting	serialization	standardization	total
600 × 600	0.02	0.01	6.13	0.43	0.41	7.00
600 × 900	0.02	0.02	4.48	0.42	0.41	5.35
600 × 1200	0.02	0.01	4.49	0.43	0.41	5.36
600 × 1500	0.03	0.01	4.54	0.44	0.43	5.45
800 × 600	0.03	0.01	3.52	0.43	0.41	4.40
800 × 900	0.02	0.01	3.50	0.43	0.41	4.37
800 × 1200	0.02	0.01	3.47	0.42	0.46	4.38
800 × 1500	0.03	0.01	3.46	0.46	0.39	4.35
1500 × 600	0.02	0.01	3.24	0.45	0.45	4.17
1500 × 900	0.03	0.01	3.29	0.46	0.44	4.23
1500 × 1200	0.03	0.01	3.34	0.47	0.44	4.29
1500 × 1500	0.02	0.02	3.29	0.46	0.44	4.23
2000 × 600	0.02	0.01	3.21	0.43	0.40	4.07
2000 × 900	0.03	0.01	3.28	0.45	0.44	4.21
2000 × 1200	0.03	0.01	3.29	0.45	0.44	4.22
2000 × 1500	0.02	0.02	3.29	0.44	0.44	4.21

Table 6.4: Performance measurements (in seconds) of the ScalAR demonstrator executing the query “Rembrandt van Rijn” and “Chiaroscuro”. The double lines indicate groups that have similar formatting times (indicated in bold).

“Rembrandt and Chiaroscuro”. The first column specifies the *screen size* of the delivery context¹⁹. We selected screen sizes that result in significantly different presentations for this particular query²⁰. However, since the number of returned images and their sizes might be different for other queries, these screen sizes do not necessarily result in significantly different presentations for other queries. The second column specifies the time spent in the *aggregation* phase, which aggregates the input for the to-be-generated structured document. The third column specifies the time spent in the *normalization* phase, which generates the presentation structures. The fourth column specifies the time needed for the *formatting*, that is, the transformation from presentation structures to their corresponding hypermedia formatting objects. This includes the generation and validation of constraints, plus the time needed to invoke alternative style rules if necessary. The fifth column lists the time needed to *serialize* the HFO tree to an XML format. Finally the sixth column contains the *total* execution time of the system. All recorded times, except for the formatting (indicated in table 6.4 in bold font), are relatively homogeneous. We conclude that the performance of the formatting phase is the phase most sensitive to adaptation.

Furthermore, multimedia documents that are adapted to a relatively large screen are pro-

¹⁹Other dimensions, such as bandwidth or expertise, do not influence the spatio-temporal constraints and therefore do not influence the execution time significantly.

²⁰Two presentations are significantly different if the formatter uses at least one different style rule to generate the document form.

screen size	#media	#PS	#HFO	#constraints	#tried/#total	time (sec)
600 × 600	17	30	30	3950	10 /36	7.00
600 × 900	17	30	30	3950	6 /36	5.35
600 × 1200	17	30	30	3950	6 /36	5.36
600 × 1500	17	30	30	3950	6 /36	5.45
800 × 600	17	30	30	3950	2 /36	4.40
800 × 900	17	30	30	3950	2 /36	4.37
800 × 1200	17	30	30	3950	2 /36	4.38
800 × 1500	17	30	30	3950	2 /36	4.35
1500 × 600	17	30	32	4228	1 /36	4.17
1500 × 900	17	30	32	4228	1 /36	4.23
1500 × 1200	17	30	32	4228	1 /36	4.29
1500 × 1500	17	30	32	4228	1 /36	4.23
2000 × 600	17	30	32	4228	1 /36	4.07
2000 × 900	17	30	32	4228	1 /36	4.21
2000 × 1200	17	30	32	4228	1 /36	4.22
2000 × 1500	17	30	32	4228	1 /36	4.21

Table 6.5: Statistics of the ScalAR demonstrator executing the query “Rembrandt van Rijn” and “Chiaroscuro”

duced faster than those adapted to smaller screens. This can be explained by the order in which alternative style rules are invoked. For example, consider two style rules, one that produces an HFO optimized for a relatively small screen and a second that produces an HFO optimized for a larger screen. Since an HFO that satisfies the constraints imposed by a small screen necessarily satisfies the constraints imposed by a larger screen, the first style rule that optimizes an HFO for a smaller screen can be successfully invoked in both cases, despite that in the latter case the available spatial resources are not optimally used. In contrast, if the second style rule is invoked first then it either succeeds, resulting in an HFO optimized for a large screen, or it fails, in which case the first style, which produces an HFO optimized for a smaller screen is invoked. As a result, the formatter produces a multimedia document for relatively large screens faster than for smaller screens.

To analyze the formatting phase in more detail, we consider table 6.5. The first column again specifies the *screen size* of the presentation, corresponding to the same screens as in table 6.4. The second column presents the number of *media* items that are used in the multimedia document. In this case the query is constant and therefore the number of media items is equal for each case. The third column presents the number of generated presentation structures (*#PS*). Since presentation structures are independent of the delivery context this number is again the same for each case. The fourth column denotes the number of hypermedia formatting objects that were generated (*#HFO*). Although the structured document is identical the number of HFOs in the document form may differ. This is because different style rules may be used to generate the document form. For example, the children in a Group PS may be all presented in a single

Box HFO, or alternatively, be split into two separate Box HFOs. The fifth column represents the number of *constraints* that were generated. The sixth column represents both the number of the solutions tried and the total number of possible solutions. The number of solutions is calculated based on the number of failed style rules. The total number of solutions is a theoretical upper bound representing all the possible presentations that can be generated based on the given structured document and style rules. However, if two solutions satisfy the same delivery context, the second solution, despite that it might be a “better” solution, will not be proposed because of the depth-first backtracking mechanism used. The seventh column denotes the same total execution time as presented in table 6.4.

The groups indicated by double lines in table 6.5 correspond to the groups indicated in table 6.4. Note the number of failed style rules (indicated in bold). We may conclude that the number of failed style rules is responsible for the increase in generation time of the formatter. This is explained by Prolog’s depth-first backtracking behavior, which, once a failure occurs, returns to the last applied style rule with an alternative, disregarding the part of the transformation that was performed after this alternative style rule. If an alternative style rule succeeds immediately the disregarded part of the transformation is relatively small and doesn’t significantly influence the performance. If, however, the alternative style rule fails as well, the formatter backtracks to the alternative style prior to that, disregarding a larger part of the transformation.

6.5.2 Reuse of style

subject	aggregation	normalization	formatting	serialization	standardization	total
Bol	0.04	0.02	6.74	0.67	0.67	8.14
Goltzius	0.04	0.01	8.57	0.62	0.55	9.79
Mostaert	0.03	0.01	10.12	0.46	0.45	11.07
Steen	0.05	0.01	9.97	0.67	0.65	11.35
Gogh	0.04	0.01	10.87	0.59	0.51	12.02
Rembrandt	0.07	0.02	12.08	0.65	0.65	13.47
Terborch	0.05	0.01	14.03	0.60	0.58	15.27

Table 6.6: Performance measurements (in seconds) of the DISC demonstrator executing variable queries (screen size is 1024 × 768)

The second set of measurements uses different structured documents from the DISC use case. The same style sheet is reused, and applied with a constant delivery context, as shown in Table 6.6. The first column represents the subject of the generated biographies. The other columns correspond to the columns of table 6.4.

Similar to the ScalAR demonstrator, most time is spent in the formatting phase. In table 6.7, the first column again represents the subject of the biography as in table 6.7. The remaining columns correspond to the columns of table 6.5.

Although the delivery context is constant, the number and sizes of the media items used

subject	#media	#PS	#HFO	#constraints	#tried/#total	time (sec)
Bol	20	42	42	5580	3/864	8.14
Goltzius	18	38	38	5048	5/288	9.79
Mostaert	15	31	31	4113	9/216	11.07
Steen	20	42	40	5304	9/864	11.35
Gogh	17	35	35	4645	9/648	12.02
Rembrandt	20	51	40	5304	9/2592	13.47
Terborch, Gerard	18	38	36	4772	9/288	15.27

Table 6.7: Statistics of the DISC demonstrator executing variable queries (screen size is 1024×768)

may vary. Consequently, the application of a style rule may fail. Similar to the adaptation to the delivery context the formatter applies an alternative style rule to compensate for the formatting failure. So from a performance perspective, varying the input document while keeping the delivery context constant is, in our scenarios, not inherently different from varying the delivery context while keeping the input document constant.

6.5.3 Comparison of the three scenarios

Code-base	#Lines	#Predicates
Cuypers formatter	6147	537
ScalAR	698	22
SEMINF	751	34
DISC	441	14

Table 6.8: Source-code statistics (19-10-2007) “Cuypers formatter” represents the common code-base. “ScalAR”, “SEMINF” and “DISC” represent the code specific for the respective scenario

Table 6.8 presents statistics on the Cuypers engine code-base (code in SWI-Prolog and Logtalk). Cuypers represents the code that is common for all use-cases and ScalAR, SEMINF and DISC represent the code-base specific for each use-case. #Lines presents the number of lines in the code and #predicates represents the number of predicates in the code.

In contrast to ScalAR and SEMINF, the DISC system has, except for the generated title, the logo and background-music, no prior knowledge on the domain. As a result, the system does not make any assumptions on the available domain data, but deals with the availability of data based on the success or failure of generic rules. This approach results in code that is better suited for heterogeneous data and possibly erroneous data. In contrast, ScalAR and SEMINF, which are domain dependent, deal with heterogeneous and possibly inconsistent data on a case

by case basis. As a result, the ScalAR and SEMINF code bases are obfuscated by code dealing with the large number of exception cases. This is reflected in table 6.8, which shows that the number of lines and the number of predicates for DISC is significantly smaller than for ScalAR and SEMINF.

scenario	performance	#media	#PS	#HFO	#constraints	#tried/#total	time (sec)
ScalAR	best	5	10	12	1590	1/27	1.14
	worst	16	26	28	3686	2/18	4.22
SEMINF	best	13	19	27	3581	2/36	3.74
	worst	22	37	46	6112	109/2916	38.69
DISC	best	20	51	42	5580	1/2592	8.38
	worst	19	51	36	4762	2320/2592	153.87

Table 6.9: Best and worst performance of ScalAR, SEMINF and DISC. (For each use-case the query is constant)

Table 6.9 presents the best and worst recorded times for our three use-cases. Based on this data we make two observations. Firstly, the performance of the formatter linearly decreases with more complex structured documents. We use here the number of generated PSs as a metric for the complexity of a structured document (#PS column).

A second observation is that the performance of the formatter significantly deteriorates due to failing style rules. When using a depth-first backtracking strategy, a failing style rule early in the transformation is relatively harmless since the scope of the style rule is limited, however, the performance penalty increases with the scope of the style rule. For example, the style rule that attempts to format the example images using a horizontal-box HFO fails after it tried all possible alternatives within its scope. This includes, for all example images, alternative positions for the associated caption (e.g. caption left-of image, caption right-of image, caption below image). Unfortunately, the alternative style rule that attempts to format the example images using a vertical-box HFO fails as well (including the previously mentioned alternatives to position the caption of an image). Furthermore, some of the alternatives do not make sense in the current context. For example, positioning the caption beside the image in a horizontal-box HFO will not lead to a solution if positioning the caption below the image already failed because both cases require the same amount of spatial resources. However, due to the *depth-first* strategy the formatter cannot adapt the order in which the style rules are applied, resulting in an inefficient strategy in these cases. Future versions of the formatter may attempt to minimize the probability of a style rule failing by selecting the style rule that is most likely to succeed. For this a *best-first* approach may be used that selects the next style rule based on a utility function that predicts the style rule that has most chance of succeeding.

Note that a *best-first* strategy may also be used to improve the quality of the multimedia document (see section 4.2.3). Typically, the metric responsible for selecting the style rule is different for both cases. This might lead to contradictory style rules, one favoring quality above efficiency. Therefore, more research is needed for specifying directives that resolve such contradictions.

6.6 Conclusion

The extended document engineering model intends to reduce authoring and designing time for multimedia documents by applying the document engineering paradigm to multimedia documents. In the previous chapter we presented the Cuypers framework, which implements the extended document engineering model and provides a system to apply the document engineering paradigm. In this chapter, we validated the model and the corresponding framework by the successful implementation of three scenarios that use the Cuypers framework. We showed that a single structured document can be adapted to various delivery contexts while the document function is preserved. Furthermore, multiple heterogeneous structured documents may be transformed to document form using a single stylesheet ensuring a consistent document form.

Although our framework successfully validates our model by demonstrating the document engineering paradigm applied to multimedia documents, these results are obtained in a relatively controlled environment. However, there are practical and conceptual concerns, which currently prevent implementing the paradigm on a larger scale:

Stylesheet complexity The “authoring” complexity for a designer of a stylesheet increases if the structure of the structured document is transformed in the document form. Compare, for example, the relatively simple CSS vocabulary, which preserves the structure of the structured document, to XSL(T), which is sufficiently expressive to adapt the structure of the document form. In addition, the designer of a stylesheet used to generate a multimedia document also needs to consider the potential failure of a style rule. As a result, the design of a stylesheet for multimedia documents is significantly more complex than for text-based documents.

Sparse form conventions Compared to text-based documents, multimedia documents cannot be considered main stream. This is contradictory to our assumption that there would be as much need for publishing multimedia documents as there is for publishing HTML documents, which was the main motivation for starting this work. Although the number of non-textual media increased, mono-media websites (e.g. Flickr, YouTube) are currently still dominant. As a result, form conventions for multimedia documents are not as well established as they are for text-based documents. Consequently, identifying form conventions that may be reused, is not evident.

Not optimized for third party users On a number of occasions people expressed interest in the Cuypers software. However, despite our efforts to package the software for distribution, the prototypical nature of the Cuypers formatter prevented successful adoption by third parties. In order to implement the Cuypers framework on a large scale significant effort needs to be spent in optimizing the framework for third party usage, such as the including documentation and examples.

Besides validating the model, we showed that the performance of the Cuypers framework is sufficiently efficient for the three implemented scenarios. For more complex structured documents, the depth-first backtracking strategy approach used to select and invoke alternative style rules is insufficiently efficient. More research is necessary to effectively balance style rules that favor quality versus style rules that favor efficiency in a best-first backtracking strategy.

Chapter 7

Conclusion

Document engineering technology is highly popular and commonly used for text-based documents. In contrast, multimedia documents are typically still authored on a case by case basis and therefore expensive to produce. In this thesis we have explored to what extent it is possible to apply document engineering techniques to multimedia content: can we separate form and function in the realm of multimedia documents, and use style sheets to automatically generate the form that communicates the function encoded in a structured multimedia document?

Given the state of the art in the research literature and the systems in use at the time of writing, we compiled a set of requirements that need to be met in order to make this possible. Based on these requirements, we describe a model, which defines three key ingredients:

Structured document A structured document captures the functional intention of the multimedia author.

Document form The document form represents these functional intentions in a particular delivery context.

Stylesheet In a stylesheet a designer describes how the first (function) can be transformed automatically into the second (form).

To put our model to the test, we studied three use cases, for which we implemented the full transformation. The iterative design process used for the various implementations allows us to identify the code common to all use cases, which constitutes the core of our Cuypers architecture. Finally, we describe the lessons learned from the three uses cases.

Based on our requirements, model, framework and practical experience we can now revisit the research question posed in the first chapter.

7.1 The research questions revisited

Research question 1 (REQUIREMENTS). *What are requirements for an extended document engineering model and processing framework that include support for multimedia documents?*

Based on the state of the art in related research areas, we formulated a list of requirements in chapter 3 that should be satisfied for a document engineering model and processing framework

that includes multimedia documents. Requirements 1 – 3 identify the desired document engineering properties, reuse of authoring and design effort, by separating document function from its form.

For text-based documents typesetting algorithms, such as hyphenation, ensure that the document form can be automatically adapted to a large variety of delivery contexts (provided that there is no constraint on the number of pages). Such generic strategy for multimedia documents does not exist, which explains why available text-based document engineering technology cannot be applied to multimedia documents. Therefore, requirements 4 – 5 explicitly state that the document form should respect the constraints imposed by the delivery context.

Requirements 6 – 8 identify the properties of transformation rules that transform document function to document form. Similar to text based document engineering, although less explicit, these include a default style rule, which transforms media items represented in the structured document to its corresponding form. Because there is no generic overflow strategy for multimedia documents the successful application of a style rule (i.e. respecting requirements 4 – 5) cannot be guaranteed *a priori* for multimedia documents. Therefore, an additional requirement states that a designer should be able express alternative style rules in case a prior one failed.

The possibility that a style rule may fail has consequences for the vocabularies used to represent the structured document and the document form. These are identified by requirements 9 – 11. Notably, the formatter should have sufficient information to detect a formatting failure. Furthermore, the formatter should have directives to prioritize information in order to preserve the intended function of the author as best as possible.

Finally, as document engineering technology is particularly desirable in a Web environment due to its heterogeneous delivery contexts, requirement 13 identifies the properties of a document engineering framework that is applicable on the Web. The most notable difference compared to text-based document engineering frameworks is that the formatting of multimedia documents is performed server-side. The advantage of this approach is that no unrealistic standardization requirements on the document form are required.

Based on the requirements 1 – 11 above we formulated an extended version of the traditional document engineering model. The more practical requirements 12 – 13 apply more directly to a software architecture built to implement the model.

Research question 2 (MODEL). *What are the properties of such an extended document engineering model?*

Our extended model, described in chapter 4, and the traditional document engineering model share the same goal: reducing authoring and design effort by separating form and function. The content and its functional structure should be encoded by the author in a structured document, of which the style and layout is defined by style rules encoded in a separate stylesheet. Together, a structured document and a corresponding stylesheet should contain sufficient information to generate a final document form automatically. Our model can be characterized by two key extensions. First, where the traditional model implicitly assumes that every style rule can be successfully applied at run time, we explicitly model how the transformation should behave when the application of a style rule fails. Second, the differences in the spatio-temporal layout of the resulting form require a vocabulary that is sufficiently expressive for describing the form of multimedia documents. The properties of these two extensions are defined in chapter 4 and are summarized below:

Explicit spatio-temporal layout We explicitly model the spatial and temporal layout of the

document form by extending the traditional two dimensional bounding box model to three dimensions to account for temporal layout.

Explicit delivery context The delivery context defines the constraints imposed by the environment in which the document is perceived, to the extent that they may influence the formatting of the document. Typical examples include the maximum size of the document form, or the (in)ability of the target device to play a specific media type. In the traditional model, the delivery context remains implicit. In our model, the delivery context explicitly models the information required to ensure that a document form is appropriate to the reader's environment. Note that on the Web, this type of information is not available at authoring time.

Explicit constraint handling and alternative style rules Each style rule being applied may violate the constraints defined by the delivery context. We model the detection of these violations explicitly, as well as the selection of alternative rules to resolve the constraint violations. Furthermore, we account for the possibility that not all media items can be presented, and assign explicit priorities to allow the system to make informed decisions about removal of media items.

Modality-dependent default style rules Typically, media items represented in a structured document do not have an explicitly described style rule, which transforms them to their corresponding form. Similar to the text-based model, in this case a generic default style rule is applied. However, in contrast to the traditional model, heterogeneous media items cannot be treated in a similar fashion (e.g. audio and image have few common properties). Our model takes this into account by allowing multiple default rules to be defined, depending on the modality of the content.

Explicit modeling of media items and metadata The properties of media items (e.g. size, modality) contain crucial information to detect constraint violations. Unlike the traditional document engineering model, which is agnostic about metadata, we model metadata explicitly in the structured document. Furthermore, we explicitly preserve metadata that is available to the structured document in the generated document form. This way we maximize the chances that the results of our transformations can be reused by other applications on the Web.

Research question 3 (FRAMEWORK). *What are the properties of a software architecture that implements the formatter of the extended document engineering model, and fulfills the requirements imposed by a web architecture?*

Document engineering is especially desirable in a Web context. However, the particular architecture of the Web adds architectural requirements on the framework that implements our model. The properties of this architecture are defined in chapter 5 and are summarized below:

Server-side formatting Unlike most other Web document formats, where the exact dimensions and positioning of the bounding boxes of the content is often determined at delivery time by the Web client, for multimedia documents this is typically done at authoring time at the content provider's side. To achieve the required flexibility needed to adapt the document form to the constraints of the client, without the need to change the client, a key property of the Cuypers architecture is its ability to calculate all bounding boxes at the server side, informed by the client's constraints as described by the delivery context.

Standardized delivery context In a Web context server and client are independent entities and communicate through standardized formats and protocols. Consequently, the delivery context describing the properties and preferences of the client-side should be sent in a standardized format to the server. Although standardization efforts are well on their way, at the moment of writing relatively few clients implement these standards¹. As a result, server-side applications that use information from the delivery context, still require a client to explicitly describe its delivery context in a proprietary format.

7.2 Lessons learned

During the implementation of the use cases described in chapter 6, we learned several lessons we think are worthwhile documenting for other (future) applications in this area:

Multimedia design dependencies on several levels of abstraction The vocabularies currently implemented by Cuypers to encode the structured documents (PS), document form (HFO) and style sheet are proof-of-concept vocabularies that might require additions or changes in the future. Here, we mainly observe that the current state of the art makes it much easier to design vocabularies for multimedia document form, while good abstractions to encode multimedia document structures are still in their infancy. Also, our transformation based on constraint solving is a proof-of-concept. We claim that the delicate dependencies in this constraint model are typical multimedia design problems: relatively high-level design decisions (such as the decision to include a specific media item or not) may depend on very low level issues in other parts of the presentation (e.g. another media item being a few pixels too wide or a few seconds too long). Further research is needed to gain more insight in these type of dependencies, not only to better balance high and low-level design decisions, but also to find better trade-offs when mixing aesthetic and functional design decisions.

Web formats serves interoperability To practically employ document engineering technology on the Web, an important property of the underlying software is that it can be fully integrated into the Web. For example, all Cuypers code can be embedded, in several ways, in a Web server. We have used Cuypers as a standalone Web service, fully implemented in SWI-Prolog using Prolog's native HTTP library. Other common configurations include running Cuypers behind a firewall (providing access only through an off-the-shelf web server such as Apache), or, as a single step in a larger Cocoon XML processing chain (using Tomcat or another of the shelf Java servlet container). In all configurations, Cuypers communicates with the external world using commonly accepted Web formats wherever possible: common media formats and RDF metadata as its input, and various flavors of SMIL as its main output format.

7.3 Discussion and remaining challenges

At the end of this thesis, we take a step back and look at our work in a wider context. Two topics are worth discussion: the role of the designer in the development of multimedia content, and the role of multimedia on the Web:

¹<http://www.w3.org/TR/2009/WD-dcontology-20090616/>

Form versus function: Designing multimedia content Designing high quality multimedia content requires very special skills that are hard to overestimate. Good designers are knowledgeable in the formulation of the message, the choice of the medium that is best suited to convey that message, the different ways that determine how that message can be best conveyed using the medium of choice, and the technical specifics of using that medium. Typically, designers are confronted with design problems that require difficult trade-offs among the interests of different parties involved (e.g. the interests of the content owners versus those of the end users), trade-offs among different design goals (e.g. aesthetics versus functionality) and trade-offs mixing different abstraction levels (e.g. high-level decisions on the overall look and feel versus the pixel-level constraints of a single media item). The many levels of dependencies between form and function in multimedia make separating the two much harder than in most text-based applications.

Areas where style sheets are currently successful are those applications where all the important trade-offs are made by the designer, and the machine only needs to make relatively small design decisions (e.g. where to break a line of HTML text given the current width of the browser window). The downside of this success is that most of these applications break down when the high-level design decisions taken are no longer appropriate for a dramatically changed delivery context. For example, many Web pages do not scale to appropriately reflect the current browser width, and those which do often break down for extremely small widths (and thus work suboptimally for many mobile devices). The results presented in this thesis show that it is, in principle, possible to develop applications that can make more higher-level design decisions automatically. It also shows, however, that making such design decisions is significantly more complex than the decisions taking by, say, a processor that processes CSS style sheets. Our works shows that a model that is able to make the intricate trade-offs that are typical in many layout and style decisions, would require much more research. To be practically applicable, such a model should also be usable by designers. This will not only put high requirements on the usability of such a model, it would also require a paradigm shift in media design. While many text authors now understand the advantages of separating content and style, for multimedia there is still a long way to go.

The role of multimedia content on the Web Looking back, multimedia on the Web did not quite develop in the way we expected when we started the research described in this thesis. In the late eighties, the success of the CD-ROM, the increased speed of CPUs, high resolution and full color graphics cards and high quality sound cards made complicated multimedia presentations relatively popular, especially in educational and entertainment purposes. In the late nineties, the relatively static text and image content of the Web, when compared to the audiovisual content available on CD-ROM, was widely regarded as a problem that needed to be addressed. Where the formats used for CD-ROM based content were typically optimized for a specific target platform, it was clear that on the Web, a more platform neutral format was required. Standardization of subsequent versions of SMIL aimed at providing such a platform and vendor-neutral format.

Despite the arrival of formats such as SMIL in 1998, content on the Web remained mainly static. The network bandwidth for end consumers (the infamous “Last Mile”) did not increase as rapidly as some had expected, and maybe even more importantly, developing sound business models for serving multimedia content proved difficult. Serving large amounts of streaming content involves high connectivity fees, costs that are hard to earn

back using advertisements – and this situation has not changed much². Many branches of the media industry are still fighting to reinvent their business models to cope with new order on the Web. Authoring complex multimedia presentations remains a time-consuming process requiring special skills, and thus involves typically even higher costs than mono medium streaming content. Where current simple streamed content is often plagued with incompatible plug-ins and media codecs, the situation is typically worse for composite media formats. So unless the business models and player support change significantly, it is hard to see how composite media formats could become more successful.

Currently, there seems a revival in the interest for multimedia content, mainly in the context of peer-to-peer networks (reducing the network costs) and the “tagging” paradigm of social content sites (e.g. Flickr, YouTube). Here the focus seems, however, to be mainly on mono-media streams of audiovisual content, and less on compound streams composed of multiple individual media items, as discussed in this thesis. So we do not expect that this trend will change the relatively small amount of structured media content (e.g. SMIL) on the Web. Similar arguments apply to small amounts of structured graphics (e.g. SVG) or structured virtual reality and 3D content (e.g. X3D). The focus on single media streams in the recently started “Video on the Web” activity by W3C³ also fits in this trend. Note that adaptivity in these audiovisual streams is, if present at all, limited to variations in the bandwidth requirements and compression techniques used, and a far cry from the more substantive adaptation techniques discussed in this thesis.

The wider application of the models and tools discussed in this thesis would require a clearer perceived need for composite multimedia, and its adaptation to a wide variety of delivery contexts. In the longer term, we expect that as the heterogeneity of Web clients will continue to increase, so will the need to adapt structured content to different clients in ways that go beyond the adaptation currently provided by CSS and other text-based style sheets. Some of the results of this research might therefore also be applicable to other domains. At the time of writing, for example, most modern Web applications feature “single page interfaces”, built using AJAX-based GUI toolkits. Currently, the interfaces developed with most of these widget libraries are hard to adapt to different delivery contexts, so these toolkits might actually benefit from the constraint-based layout techniques proposed in this thesis.

The wider application of the results of our research has also been hindered by the lack of available annotated media content. Partly due to the popularity of web mashups, and in a broader context of reusing and repurposing media assets, there seems to be growing consensus⁴ about the importance of data format interoperability, not only in terms of the media encoding, but also in terms of the metadata formats and vocabularies used for the description of the media content. Combined, the increased need for adaptable content, and the increased availability of annotated assets will also increase the practical relevance of the work described in this thesis.

²At the time of writing, Google’s video website YouTube is still not profitable:

http://findarticles.com/p/articles/mi_qn4176/is_20070322/ai_n18764215/

³<http://www.w3.org/2008/WebVideo/Activity.html>

⁴In 2008, W3C started a Working Group on this topic, see

<http://www.w3.org/2008/WebVideo/Annotations/> and

<http://www.w3.org/2008/WebVideo/Fragments/>

Appendix A

Hypermedia Formatting Objects

In this appendix we provide a complete overview of the style attributes and delivery context attributes that are implemented in the Cuypers multimedia document formatter. Note however, that we do not claim these are all the relevant attributes in the context of multimedia document formatting.

A.1 Style attributes

Style attributes	Type	Inherited?	Default value
<i>Margin – Left Right Top Bottom</i>	<i>Int</i>	No	Min
<i>Border – Left Right Top Bottom</i>	<i>Int</i>	No	Min
<i>Padding – Left Right Top Bottom</i>	<i>Int</i>	No	Min
<i>Delay – Start End</i>	<i>Int</i>	No	Min
<i>Min Max – aspectRatio</i>	<i>Float</i>	No	0.001
<i>Fill – Width Height Duration</i>	{ <i>Min, Max, Center</i> }	No	Min
<i>Fill – Margin Border Padding</i>	{ <i>Min, Max, Center</i> }	No	Min
<i>Align – Horizontal Vertical Temporal</i>	{ <i>Left, Right, Center</i> }	Yes	Min
<i>Content – Color</i>	<i>Int</i>	Yes	Black*
<i>Content – BackgroundColor</i>	<i>Int</i>	Yes	Grey*
<i>Border – Color</i>	<i>Int</i>	Yes	Grey*
<i>Font – Family</i>	<i>Str</i>	Yes	Helvetica
<i>Font – Style</i>	<i>Str</i>	Yes	normal
<i>FontSize</i>	<i>Int</i>	Yes	12
<i>Transition – In Out – Type</i>	<i>Str</i>	Yes	None
<i>Transition – In Out – SubType</i>	<i>Str</i>	Yes	None
<i>TransitionIn Out – Duration</i>	<i>Int</i>	Yes	0

Table A.1: HFO style attributes (*=string representation) (** also used to calculate the required spatial surface for a text HFO)

A.2 Delivery context attributes

Attributes	Type	Initial
Max-Width (pixels)	<i>Int</i>	1024
Max-Height (pixels)	<i>Int</i>	768
Max-Duration (seconds)	<i>Int</i>	200
Bandwidth	{ <i>Fast, Medium, Slow</i> }	Fast
Language (code)	<i>Int</i>	English*
Expertise level	{1, 2, 3, 4, 5}	1
Interactivity level	{1, 2, 3, 4, 5}	1
Output format	<i>String</i>	SMIL2
Read speed (words per minute)	<i>Int</i>	180
Stylesheet	<i>String</i>	default

Table A.2: Delivery context (*=string representation)

Appendix B

Performance Statistics

The machine we used to take these measurements runs on a Intel(R) Pentium(R) M 1.70GHz processor, with 1GB of RAM memory. The operating system is Linux (Fedora Core 5) with a 2.6.20 kernel

screen size	aggregation	normalization	formatting	serialization	standardization	total
600 × 600	0.02	0.01	6.13	0.43	0.41	7.0
600 × 900	0.02	0.02	4.48	0.42	0.41	5.35
600 × 1200	0.02	0.01	4.49	0.43	0.41	5.36
600 × 1500	0.03	0.01	4.54	0.44	0.43	5.45
800 × 600	0.03	0.01	3.52	0.43	0.41	4.4
800 × 900	0.02	0.01	3.5	0.43	0.41	4.37
800 × 1200	0.02	0.01	3.47	0.42	0.46	4.38
800 × 1500	0.03	0.01	3.46	0.46	0.39	4.35
1500 × 600	0.02	0.01	3.24	0.45	0.45	4.17
1500 × 900	0.03	0.01	3.29	0.46	0.44	4.23
1500 × 1200	0.03	0.01	3.34	0.47	0.44	4.29
1500 × 1500	0.02	0.02	3.29	0.46	0.44	4.23
2000 × 600	0.02	0.01	3.21	0.43	0.4	4.07
2000 × 900	0.03	0.01	3.28	0.45	0.44	4.21
2000 × 1200	0.03	0.01	3.29	0.45	0.44	4.22
2000 × 1500	0.02	0.02	3.29	0.44	0.44	4.21

Table B.1: Performance measurements (in seconds) of the ScalAR demonstrator executing the query “Rembrandt van Rijn” and “Chiaroscuro”

screen size	#media	#PS	#HFO	#constraints	#tried/#total	time
600 × 600	17	30	30	3950	10/36	7.0
600 × 900	17	30	30	3950	6/36	5.35
600 × 1200	17	30	30	3950	6/36	5.36
600 × 1500	17	30	30	3950	6/36	5.45
800 × 600	17	30	30	3950	2/36	4.4
800 × 900	17	30	30	3950	2/36	4.37
800 × 1200	17	30	30	3950	2/36	4.38
800 × 1500	17	30	30	3950	2/36	4.35
1500 × 600	17	30	32	4228	1/36	4.17
1500 × 900	17	30	32	4228	1/36	4.23
1500 × 1200	17	30	32	4228	1/36	4.29
1500 × 1500	17	30	32	4228	1/36	4.23
2000 × 600	17	30	32	4228	1/36	4.07
2000 × 900	17	30	32	4228	1/36	4.21
2000 × 1200	17	30	32	4228	1/36	4.22
2000 × 1500	17	30	32	4228	1/36	4.21

Table B.2: Statistics of the ScalAR demonstrator executing the query “Rembrandt van Rijn” and “Chiaroscuro”

query	aggregation	normalization	formatting	serialization	standardization	total
Rembrandt + Chiaroscuro	0.02	0.01	3.47	0.4	0.38	4.28
Rembrandt + Bible	0.03	0.01	3.04	0.36	0.34	3.78
Rembrandt + Christ	0.03	0.0	1.0	0.17	0.18	1.38
Rembrandt + Brush technique	0.02	0.01	2.33	0.37	0.35	3.08
Rembrandt + Light source	0.02	0.01	1.25	0.2	0.18	1.66
Rembrandt + Rembrandt's circle	0.03	0.0	0.87	0.16	0.16	1.22
Rembrandt + Fantasy costume	0.02	0.01	1.23	0.21	0.2	1.67
Rembrandt + Age	0.03	0.0	1.98	0.32	0.31	2.64
Rembrandt + Fire	0.02	0.01	1.24	0.21	0.2	1.68
Rubens + Chiaroscuro	0.02	0.01	0.99	0.18	0.17	1.37
Rubens + Bible	0.02	0.01	0.99	0.17	0.16	1.35
Rubens + Christ	0.02	0.01	0.99	0.19	0.15	1.36
Rubens + Brush technique	0.03	0.0	0.87	0.17	0.15	1.22
Rubens + Age	0.03	0.0	0.88	0.16	0.16	1.23
Eeckhout + Chiaroscuro	0.03	0.0	0.99	0.18	0.17	1.37
Eeckhout + Bible	0.02	0.01	1.0	0.18	0.17	1.38
Eeckhout + Christ	0.02	0.01	1.0	0.18	0.18	1.39
Eeckhout + Rembrandt's circle	0.02	0.0	0.89	0.16	0.15	1.22
Dou + Chiaroscuro	0.02	0.01	0.99	0.18	0.17	1.37
Dou + Brush technique	0.02	0.01	0.87	0.16	0.16	1.22
Dou + Light source	0.02	0.0	0.88	0.16	0.16	1.22
Dou + Rembrandt's circle	0.02	0.01	1.49	0.28	0.27	2.07
Dou + Age	0.03	0.0	0.87	0.16	0.16	1.22
Dou + Genre painting	0.03	0.0	1.0	0.18	0.17	1.38
Brugghen + Chiaroscuro	0.02	0.01	1.0	0.18	0.17	1.38
Brugghen + Christ	0.02	0.01	0.99	0.18	0.17	1.37
Brugghen + Light source	0.02	0.01	0.87	0.15	0.16	1.21
Vermeeyen + Chiaroscuro	0.02	0.01	1.0	0.18	0.17	1.38
Vermeeyen + Christ	0.02	0.01	1.0	0.18	0.17	1.38
Vermeeyen + Light source	0.02	0.01	0.88	0.16	0.15	1.22
Venne + Bible	0.03	0.01	0.99	0.18	0.17	1.38
Vianen + Bible	0.03	0.01	0.99	0.18	0.17	1.38
Hals + Brush technique	0.03	0.0	1.7	0.28	0.27	2.28
Hals + Genre painting	0.02	0.01	1.0	0.17	0.18	1.38
Flinck + Bible	0.02	0.01	0.99	0.18	0.17	1.37
Flinck + Brush technique	0.02	0.01	0.87	0.17	0.15	1.22
Flinck + Rembrandt's circle	0.03	0.0	1.65	0.27	0.28	2.23
Flinck + Fantasy costume	0.02	0.01	1.26	0.21	0.2	1.7
Flinck + Age	0.02	0.01	0.87	0.16	0.16	1.22
Metsu + Age	0.02	0.01	1.24	0.21	0.2	1.68
Metsu + Genre painting	0.02	0.01	0.99	0.17	0.18	1.37
Steen + Genre painting	0.02	0.01	1.38	0.22	0.19	1.82
Vermeer + Brush technique	0.03	0.0	0.88	0.16	0.15	1.22
Vermeer + Genre painting	0.02	0.01	2.13	0.34	0.34	2.84

Table B.3: Performance measurements (in seconds) of the ScalAR demonstrator executing variable queries (screen size is 1024×768)

query	#media	#PS	#HFO	#constraints	#tried/#total	time
Rembrandt + Chiaroscuro	16	26	28	3686	2/18	4.28
Rembrandt + Bible	14	23	25	3293	2/18	3.78
Rembrandt + Christ	6	11	13	1719	1/27	1.38
Rembrandt + Brush technique	13	22	26	3442	1/18	3.08
Rembrandt + Light source	7	13	15	1985	1/27	1.66
Rembrandt + Rembrandt's circle	5	10	12	1590	1/27	1.22
Rembrandt + Fantasy costume	7	13	15	1985	1/27	1.67
Rembrandt + Age	11	19	23	3049	1/18	2.64
Rembrandt + Fire	7	13	15	1985	1/27	1.68
Rubens + Chiaroscuro	6	11	13	1719	1/27	1.37
Rubens + Bible	6	11	13	1719	1/27	1.35
Rubens + Christ	6	11	13	1719	1/27	1.36
Rubens + Brush technique	5	10	12	1590	1/27	1.22
Rubens + Age	5	10	12	1590	1/27	1.23
Eeckhout + Chiaroscuro	6	11	13	1719	1/27	1.37
Eeckhout + Bible	6	11	13	1719	1/27	1.38
Eeckhout + Christ	6	11	13	1719	1/27	1.39
Eeckhout + Rembrandt's circle	5	10	12	1590	1/27	1.22
Dou + Chiaroscuro	6	11	13	1719	1/27	1.37
Dou + Brush technique	5	10	12	1590	1/27	1.22
Dou + Light source	5	10	12	1590	1/27	1.22
Dou + Rembrandt's circle	9	16	20	2654	1/18	2.07
Dou + Age	5	10	12	1590	1/27	1.22
Dou + Genre painting	6	11	13	1719	1/27	1.38
Bruggen + Chiaroscuro	6	11	13	1719	1/27	1.38
Bruggen + Christ	6	11	13	1719	1/27	1.37
Bruggen + Light source	5	10	12	1590	1/27	1.21
Vermeyen + Chiaroscuro	6	11	13	1719	1/27	1.38
Vermeyen + Christ	6	11	13	1719	1/27	1.38
Vermeyen + Light source	5	10	12	1590	1/27	1.22
Venne + Bible	6	11	13	1719	1/27	1.38
Vianen + Bible	6	11	13	1719	1/27	1.38
Hals + Brush technique	9	16	20	2654	1/18	2.28
Hals + Genre painting	6	11	13	1719	1/27	1.38
Flinck + Bible	6	11	13	1719	1/27	1.37
Flinck + Brush technique	5	10	12	1590	1/27	1.22
Flinck + Rembrandt's circle	9	16	20	2654	1/18	2.23
Flinck + Fantasy costume	7	13	15	1985	1/27	1.7
Flinck + Age	5	10	12	1590	1/27	1.22
Metsu + Age	7	13	15	1985	1/27	1.68
Metsu + Genre painting	6	11	13	1719	1/27	1.37
Steen + Genre painting	8	14	16	2114	1/27	1.82
Vermeer + Brush technique	5	10	12	1590	1/27	1.22
Vermeer + Genre painting	12	20	24	3178	1/18	2.84

Table B.4: Statistics of the ScalAR demonstrator executing variable queries (screen size is 1024×768)

screen size	aggregation	normalization	formatting	serialization	standardization	total
500 × 600	0.03	0.01	116.14	0.71	0.69	117.58
500 × 900	0.04	0.01	224.09	0.78	0.65	225.57
500 × 1200	0.03	0.01	273.5	0.79	0.69	275.02
500 × 1500	0.03	0.01	276.81	0.72	0.7	278.27
600 × 600	0.03	0.01	81.01	0.8	0.72	82.57
600 × 900	0.03	0.01	124.36	0.8	0.71	125.91
600 × 1200	0.03	0.01	164.95	0.73	0.7	166.42
600 × 1500	0.03	0.01	166.01	0.73	0.73	167.51
800 × 600	0.03	0.01	32.3	0.81	0.7	33.85
800 × 900	0.03	0.01	69.56	0.8	0.71	71.11
800 × 1200	0.03	0.02	69.84	0.81	0.7	71.4
800 × 1500	0.03	0.01	70.51	0.82	0.7	72.07
1200 × 600	0.03	0.01	7.38	0.76	0.81	8.99
1200 × 900	0.02	0.01	7.26	0.75	0.82	8.86
1200 × 1200	0.03	0.01	7.3	0.83	0.74	8.91
1200 × 1500	0.03	0.01	7.25	0.85	0.73	8.87
2000 × 600	0.03	0.01	7.27	0.85	0.74	8.9
2000 × 900	0.03	0.01	7.0	0.83	0.75	8.62
2000 × 1200	0.02	0.02	7.26	0.83	0.74	8.87
2000 × 1500	0.03	0.01	7.35	0.85	0.75	8.99

Table B.5: Performance measurements (in seconds) of the SEMINF demonstrator executing the query “Abraham Lincoln”

screen size	#media	#PS	#HFO	#constraints	#tried/#total	time
500 × 600	22	37	44	5834	3322/5832	117.58
500 × 900	22	37	44	5834	3160/5832	225.57
500 × 1200	22	37	44	5834	3160/5832	275.02
500 × 1500	22	37	44	5834	3160/5832	278.27
600 × 600	22	37	44	5834	812/2916	82.57
600 × 900	22	37	44	5834	488/2916	125.91
600 × 1200	22	37	44	5834	488/2916	166.42
600 × 1500	22	37	44	5834	488/2916	167.51
800 × 600	22	37	44	5834	164/2916	33.85
800 × 900	22	37	44	5834	164/2916	71.11
800 × 1200	22	37	44	5834	164/2916	71.4
800 × 1500	22	37	44	5834	164/2916	72.07
1200 × 600	22	37	46	6112	1/2916	8.99
1200 × 900	22	37	46	6112	1/2916	8.86
1200 × 1200	22	37	46	6112	1/2916	8.91
1200 × 1500	22	37	46	6112	1/2916	8.87
2000 × 600	22	37	46	6112	1/2916	8.9
2000 × 900	22	37	46	6112	1/2916	8.62
2000 × 1200	22	37	46	6112	1/2916	8.87
2000 × 1500	22	37	46	6112	1/2916	8.99

Table B.6: Statistics of the SEMINF demonstrator executing the query “Abraham Lincoln”

query	#media	#PS	#HFO	#constraints	#tried/#total	time
Abraham Lincoln	22	37	46	6112	109/2916	38.69
Monuments	20	31	38	5026	31/324	22.7
Presidents	13	19	27	3581	2/36	3.74
Sculpture	16	25	34	4516	26/324	22.91
statue	20	31	38	5026	31/324	22.07
Washington	19	28	34	4484	3/36	5.55
Horydczak	19	28	34	4484	3/36	5.21
Gottscho	11	16	22	2912	5/54	3.88

Table B.7: Statistics of the SEMINF demonstrator executing variable queries (screen size is 1024×768)

query	aggregation	normalization	formatting	serialization	standardization	total
Abraham Lincoln	0.03	0.01	36.98	0.85	0.82	38.69
Monuments	0.07	0.01	21.33	0.66	0.63	22.7
Presidents	0.06	0.01	2.9	0.39	0.38	3.74
Sculpture	0.08	0.0	21.84	0.5	0.49	22.91
statue	0.07	0.01	20.79	0.64	0.56	22.07
Washington	0.07	0.01	4.43	0.5	0.54	5.55
Horydczak	0.11	0.01	4.1	0.51	0.48	5.21
Gott scho	0.01	0.0	3.35	0.25	0.27	3.88

Table B.8: Performance measurements (in seconds) of the SEMINF demonstrator executing variable queries (screen size is 1024×768)

screen size	aggregation	normalization	formatting	serialization	standardization	total
500×900	0.08	0.01	152.6	0.59	0.59	153.87
500×1200	0.08	0.02	10.7	0.63	0.61	12.04
500×1500	0.08	0.02	10.75	0.63	0.62	12.1
600×900	0.08	0.02	55.79	0.64	0.62	57.15
600×1200	0.08	0.02	102.23	0.66	0.65	103.64
600×1500	0.08	0.02	102.59	0.63	0.62	103.94
800×600	0.08	0.01	12.73	0.66	0.67	14.15
800×900	0.08	0.02	11.05	0.69	0.68	12.52
800×1200	0.07	0.02	9.03	0.79	0.67	10.58
800×1500	0.08	0.02	8.92	0.78	0.7	10.5
1200×600	0.08	0.01	16.57	0.67	0.65	17.98
1200×900	0.08	0.02	7.47	0.78	0.69	9.04
1200×1200	0.08	0.02	7.65	0.83	0.71	9.29
1200×1500	0.08	0.02	7.89	0.84	0.66	9.49
1500×600	0.08	0.02	7.02	0.7	0.75	8.57
1500×900	0.08	0.02	6.8	0.79	0.69	8.38
1500×1200	0.08	0.02	6.92	0.7	0.75	8.47
1500×1500	0.08	0.02	6.99	0.71	0.73	8.53

Table B.9: Performance measurements (in seconds) of the DISC demonstrator executing the query “Rembrandt van Rijn”

screen size	#media	#PS	#HFO	#constraints	#tried/#total	time
500 × 900	19	51	36	4762	2320/2592	153.87
500 × 1200	20	51	38	5028	592/2592	12.04
500 × 1500	20	51	38	5028	592/2592	12.1
600 × 900	20	51	38	5028	592/2592	57.15
600 × 1200	20	51	38	5028	592/2592	103.64
600 × 1500	20	51	38	5028	592/2592	103.94
800 × 600	20	51	40	5304	297/2592	14.15
800 × 900	20	51	40	5304	153/2592	12.52
800 × 1200	20	51	42	5580	151/2592	10.58
800 × 1500	20	51	42	5580	151/2592	10.5
1200 × 600	20	51	40	5304	9/2592	17.98
1200 × 900	20	51	42	5580	3/2592	9.04
1200 × 1200	20	51	42	5580	3/2592	9.29
1200 × 1500	20	51	42	5580	3/2592	9.49
1500 × 600	20	51	42	5580	1/2592	8.57
1500 × 900	20	51	42	5580	1/2592	8.38
1500 × 1200	20	51	42	5580	1/2592	8.47
1500 × 1500	20	51	42	5580	1/2592	8.53

Table B.10: Statistics of the DISC demonstrator executing the query “Rembrandt van Rijn”

query	aggregation	normalization	formatting	serialization	standardization	total
Rembrandt	0.07	0.02	12.08	0.65	0.65	13.47
Gogh	0.04	0.01	10.87	0.59	0.51	12.02
Steen	0.05	0.01	9.97	0.67	0.65	11.35
Bol	0.04	0.02	6.74	0.67	0.67	8.14
Terborch, Gerard	0.05	0.01	14.03	0.6	0.58	15.27
Mostaert	0.03	0.01	10.12	0.46	0.45	11.07
Goltzius	0.04	0.01	8.57	0.62	0.55	9.79

Table B.11: Performance measurements (in seconds) of the DISC demonstrator executing variable queries (screen size is 1024 × 768)

screen size	#media	#PS	#HFO	#constraints	#tried/#total	time
Rembrandt	20	51	40	5304	9/2592	13.47
Gogh	17	35	35	4645	9/648	12.02
Steen	20	42	40	5304	9/864	11.35
Bol	20	42	42	5580	3/864	8.14
Terborch, Gerard	18	38	36	4772	9/288	15.27
Mostaert	15	31	31	4113	9/216	11.07
Goltzius	18	38	38	5048	5/288	9.79

Table B.12: Statistics of the DISC demonstrator executing variable queries (screen size is 1024×768)

Summary

A Document Engineering Model and Processing Framework for Multimedia Documents

Multimedia documents are different from text-based documents in the sense that they do not have a predefined dominant media type but are composed of multiple media items using different media types, such as, image, text, audio and video. The author of a multimedia document uses media items that are, either specifically created, or (re)used from existing resources, to represent the message she intends to convey. Furthermore, a multimedia document has, besides two spatial dimensions, a temporal dimension. Consequently, the author of a multimedia document should, in addition to the spatial layout, synchronize media items in a meaningful way.

Authoring multimedia documents is in multiple ways different from authoring a text-based electronic document. First, modern text processors allow an author to abstract from typesetting details, such as hyphenation, kerning and leading. The word processor automatically formats the text in such a way that it fits within the designated area, such as a page or screen. In contrast, the author of a multimedia document carefully designs a multimedia document so that it exactly fits the screen size the document is designed for. One presentation may have been created for a screen with a width of 1024 pixels and a height of 768 pixels. A second presentation, which conveys an identical message, but on a screen with a width of 640 pixels and a height of 800 pixels will typically require manual authoring.

Secondly, modern text processors often have the ability to include predefined styles (e.g. corporate identity), which allows an author to abstract from the styling of the document. Consequently, an author does not require design expertise to ensure a consistently formatted and aesthetically pleasing document. In contrast, modern authoring tools for multimedia documents require an author to make both authoring and design decisions.

The reason that authoring and design are intertwined in the production of multimedia documents is that the spatial layout and temporal synchronization between media items is semantically significant. Unlike text, where a sentence or word may be split to continue on the next line or page, breaking the spatio-temporal relations between media items in a multimedia document typically alters the message conveyed by the document. When the presentation does not fit the screen, the author carefully redesigns the presentation in order to maintain these relationships.

Although a multimedia document can be adapted to a particular context, and multiple multimedia documents can be consistently styled, this typically requires significant human investment. The costs involved in authoring and designing multimedia documents are therefore relatively high compared to textual documents. As a result, the production of multimedia documents is

only viable in specific cases, which is unfortunate because multimedia documents are typically effective to convey a particular message.

To address this discrepancy we derived requirements for an extended document engineering model. These include requirements derived from the traditional document engineering model. However, the traditional model assumes generally applicable overflow strategies, which is not the case for multimedia documents. Therefore, the formatting of multimedia documents may, in contrast to text-based documents, fail. An extended document engineering model should thus detect constraint violations and propose alternative formatting when necessary. We defined such an extended model, expressing explicit knowledge on the properties of the delivery context and form constructs that are relevant for detecting constraint violations. As a result, the knowledge requirements in an extended document engineering model are significantly larger compared to traditional document engineering. To reduce the associated costs, an extended document engineering model should support reuse and preserve existing knowledge where possible.

To evaluate the model, we have implemented a multimedia document engineering formatter using a Constraint Logic Programming approach. The formatter is embedded in a client-server framework so that it can be used in a web environment. Based on this framework we demonstrate in three distinctive use cases that the document engineering paradigm may be successfully applied to a number of multimedia documents that are representative for each use case. Successful in this context means that: firstly, a single set of style rules may be used to transform multiple structured documents. Secondly, the intended output is automatically adapted to the delivery context without changing the function that is conveyed.

Compared to the traditional model, our model extends the notions of function, form and style to meet the specific requirements of multimedia documents. We include an explicit and parametrized delivery context that represents the constraints of the environment the document is played in, and the specification of alternative style rules that are automatically invoked by the formatter if the resulting document form does not comply to the hard constraints imposed by the delivery context.

Bibliography

- [1] Adobe Systems Incorporated. Portable Document Format (PDF). See <http://www.adobe.com/products/acrobat/adobepdf.html>.
- [2] Adobe Systems Incorporated. PostScript. See <http://www.adobe.com/products/postscript/main.html>.
- [3] Administrator Nederland B.V. *SeRQL user manual*, April 4, 2003.
- [4] Philippe Aigrain. Content-based representation and retrieval of visual media: A state-of-the-art review. *Multimedia Tools and Applications*, 3:179–202, 1996.
- [5] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–844, November 1983.
- [6] Alia Amin, Lynda Hardman, and Jacco van Ossenbruggen. Semantic Design. Poster presented at MultimediaN Day, June 2006, 2006.
- [7] E. André, W. Finkler, W. Graf, T. Rist, A. Schauder, and W. Wahlster. WIP: The Automatic Synthesis of Multimodal Presentations. In Mark T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 75–93. AAAI Press, 1993.
- [8] Elisabeth André, Jochen Müller, and Thomas Rist. *WIP/PPP: Knowledge-Based Methods for Fully Automated Multimedia Authoring*. London, UK, 1996.
- [9] Apple. Apple QuickTime Player. <http://www.apple.com/quicktime/>.
- [10] K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
- [11] K.R. Apt and E. Monfroy. “Automatic generation of constraint propagation algorithms for small finite domains”. In *Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP’99)*, pages 58–72. Springer-Verlag, 1999. Lecture Notes in Computer Science 1713.
- [12] K.R. Apt and M.G. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, 2006.
- [13] Richard Arndt, Raphaël Troncy, Steffen Staab, and Lynda Hardman. Adding Formal Semantics to MPEG7: Designing a Well-Founded Multimedia Ontology for the Web. Technical Report KU-N0407, KU and CWI, January 2007.

- [14] Richard Arndt, Raphaël Troncy, Steffen Staab, Lynda Hardman, and Miroslav Vacura. COMM: Designing a Well-Founded Multimedia Ontology for the Web. In *The Semantic Web - ISWC/ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 30–43, Busan, Korea, November 11-15 2007.
- [15] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Addison-Wesley Publishing Company, 3rd edition, 2000.
- [16] G.J. Badros and A. Borning. *Cassowary: A Constraint Solving Toolkit*, 1999.
- [17] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison Wesley, 1998.
- [18] John Bateman, Jörg Klein, Thomas Kamps, and Klaus Reichenberger. Towards Constructive Text, Diagram, and Layout Generation for Information Presentation. *Computational Linguistics*, 27(3):409–449, September 2001.
- [19] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, August 1998. RFC 2396.
- [20] Tim Berners-Lee. Web Architecture. Slide 10 of invited talk at Semantic Web - XML2000, 2000. See <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>.
- [21] Tim Berners-Lee. Linked data. <http://www.w3.org/DesignIssues/LinkedData.html>, 2006.
- [22] Tim Berners-Lee and Mark Fischetti. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, San Francisco, Oktober 1999.
- [23] Niels Ole Bernsen. Defining a Taxonomy of Output Modalities from an HCI Perspective. *Computer Standards and Interfaces*, 18:537–553, 1997.
- [24] Frédéric Bes and Cécile Roisin. A Presentation Language for Controlling the Formatting Process in Multimedia Presentations. In *Proceedings of Document Engineering 2002*, 2002.
- [25] C. Bizer, R. Lee, and E. Pietriga. Fresnel — A Browser-Independent Presentation Vocabulary for RDF. In *Proceedings of the Second International Workshop on Interaction Design and the Semantic Web*, Galway, Ireland, November 6–10, 2005.
- [26] Stefano Bocconi. *Vox Populi: generating video documentaries from semantically annotated media repositories*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, November 30, 2006.
- [27] M. Bordegoni, G. Faconti, M.T. Maybury, T. Rist, S. Ruggieri, P. Trahanias, and M. Wilson. A Standard Reference Model for Intelligent Multimedia Presentation Systems. *Computer Standards & Interfaces*, 18(6-7):477–496, December 1997.
- [28] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation, May 12, 1998.

- [29] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 Specification. W3C Recommendation, February 10, 1998.
- [30] Pearl Brereton, David Budgen, and Geoff Hamilton. Hypertext: The Next Maintenance Mountain. *Computer*, pages 49–55, December 1998.
- [31] Jeen Broekstra, Arjohm Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Ian Horrocks and Jim Hendler, editors, *The Semantic Web - ISWC 2002*, number 2342 in Lecture Notes in Computer Science, pages 54–68, Berlin Heidelberg, 2002. Springer.
- [32] John F. Buford. Evaluating HyTime: an examination and implementation experience. In *Proceedings of the Seventh ACM Conference on Hypertext (Hypertext'96)*, pages 105 – 115, March 1996, Washington D.C., 1996. ACM, ACM Press.
- [33] Dick C.A. Bulterman, Jack Jansen, Sjoerd Mullender, and Kees Blom. The AMBULANT Open SMIL Player. <http://www.cwi.nl/projects/Ambulant/>.
- [34] S. Chatman. *Coming to Terms -The Rhetoric of Narrative in Fiction and Film*. Cornell University Press, Ithaca, New York, 1990.
- [35] James Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 16 November 1999.
- [36] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, 16 November 1999.
- [37] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: an overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.
- [38] Open Archives Initiative Community. OAI Protocol for Metadata Harvesting 2.0., 2002.
- [39] Open Archives Initiative Community. OAI Object Reuse and Exchange, 2008.
- [40] Mozilla Corporation. Mozilla, 2005.
- [41] Marc Davis. Media Streams: An Iconic Visual Language for Video Representation. In Ronald M. Baecker, Jonathan Grudin, William A. S. Buxton, and Saul Greenberg, editors, *Readings in Human-Computer Interaction: Toward the Year 2000*, pages 854–866. Morgan Kaufmann Publishers, Inc., 1995.
- [42] Paulo Jorge Lopes de Moura. Logtalk 2.6 Documentation. Technical report, Department of Mathematics and Informatics. University of Beira Interior., July 2000.
- [43] 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.
- [44] 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. Superseded by <http://www.w3.org/TR/xptr-framework/> etc.

- [45] H.L. Dreyfus and S.E. Dreyfus. Making a Mind Versus Modeling the Brain: Artificial Intelligence Back at a Branch-Point. *Artificial Intelligence, Vol. 117, No. 1 (Winter 1988)*, 1988.
- [46] Dublin Core Community. Dublin Core Element Set, Version 1.1, 2003.
- [47] D. J. Duke, I. Herman, T. Rist, and M. Wilson. Relating the primitive hierarchy of the PREMIO standard to the standard reference model for intelligent multimedia presentation systems. *Computer Standards & Interfaces*, 18(6-7):525–535, December 1997.
- [48] Umberto Eco. *A Theory of Semiotics*. Palgrave Macmillan, 1977.
- [49] S.M. Eisenstein. *Film Form: Essays in Film Theory*. Harcourt Brace Jovanovich Publishers, 1949.
- [50] Anton Eliëns. *Principles of Object-Oriented Software Development*. Addison-Wesley, 2nd edition, 2000.
- [51] Jérôme Euzenat, Nabil Layaïda, and Victor Dias. A semantic framework for multimedia document adaptation. In *In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'2003)*, Acapulco, Mexico, August 9-16, 2003.
- [52] Kateryna Falkovych and Frank Nack. Context Aware Guidance for Multimedia Authoring: Harmonizing Domain and Discourse Knowledge. *Multimedia Systems Journal, Special issue on Multimedia System Technologies for Educational Tools, S. Acton, F. Kishino, R. Nakatsu, M. Rauterberg & J. Tang eds.*, 11(3):226–235, 2006.
- [53] Kateryna Falkovych and Frank Nack. Context aware guidance for multimedia authoring: harmonizing domain and discourse knowledge. *Multimedia Systems*, 11(3):226–235, 2006.
- [54] Steven K. Feiner and Kathleen R. McKeown. Automating the Generation of Coordinated Multimedia Explanations. In Mark T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 89–97. AAAI Press, 1993.
- [55] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication Series. MIT Press, 1998.
- [56] Jon Ferraiolo. Scalable Vector Graphics (SVG) 1.0 Specification. W3C Recommendation, 4 September 2001.
- [57] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, Irvine, 2000.
- [58] F. Frasnar, A. Telea, and G.J. Houben. Adapting graph visualization techniques for the visualization of rdf data. In *Visualizing the Semantic Web* [63], pages 64–76.
- [59] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046, November 1996. Obsoletes: 1521, 1522, 1590 Category: Standards Track.

- [60] David R. Fuchs. Device-independent (DVI). See <http://www.ctan.org/tex-archive/systems/knuth/texware/dvitype.web>.
- [61] 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.
- [62] David Garlan and Mary Shaw. An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, 1, 1993. Edited by V. Ambrolia and G. Tortora.
- [63] Vladimir Geroimenko and Chamoi Chen. *Visualizing the Semantic Web — XML-based Internet and Information Visualization*. Springer-Verlag, 2003.
- [64] Getty Research Institute. Art & Architecture Thesaurus (Online). <http://www.getty.edu/research/tools/vocabulary/aat/>, 2000. Version 2.0.
- [65] Joost Geurts, Stefano Bocconi, Jacco van Ossenbruggen, and Lynda Hardman. Towards Ontology-driven Discourse: From Semantic Graphs to Multimedia Presentations. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Second International Semantic Web Conference (ISWC2003)*, pages 597–612, Sanibel Island, Florida, USA, October 20-23, 2003. Springer-Verlag.
- [66] Joost Geurts, Jacco van Ossenbruggen, and Lynda Hardman. Application-Specific Constraints for Multimedia Presentation Generation. In *Proceedings of the International Conference on Multimedia Modeling 2001 (MMM01)*, pages 247–266, CWI, Amsterdam, The Netherlands, November 5-7, 2001.
- [67] Joost Geurts, Jacco van Ossenbruggen, and Lynda Hardman. Requirements for practical multimedia annotation. In *Workshop on Multimedia and the Semantic Web*, pages 4–11, Heraklion, Crete, May 2005. part of 2nd European Semantic Web Conference.
- [68] Joost Geurts, Jacco van Ossenbruggen, Lynda Hardman, and Marc Davis. Video on the Semantic Web - Experiences with Media Streams. Technical Report INS-E0404, CWI, 2004.
- [69] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990. Edited and with a foreword by Yuri Rubinsky.
- [70] J. Greimas. *Structural Semantics: An Attempt at a Method*. Lincoln: University of Nebraska Press, 1983.
- [71] William I. Grosky. Managing multimedia information in database systems. *Communications of the ACM*, 40(12):72–80, December 1997.
- [72] Ubiquitous Web Applications Working Group. More information on <http://www.w3.org/2007/uwa/>.
- [73] Kenneth Haase. Context for semantic metadata. In *Proceedings of the 12th ACM International Conference on Multimedia*, pages 204–211, 2004.

- [74] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994. Edited by K. Grønbaeck and R. Trigg.
- [75] L. Hardman, Z. Obrenovic, F.-M. Nack, B. Kerhery, and K. Piersol. Canonical Processes Of Semantically Annotated Media Production. *Multimedia systems*, 14(6):327 – 340, December 2008.
- [76] 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/>.
- [77] Lynda Hardman, Marcel Worrying, and Dick C.A. Bulterman. Integrating the Amsterdam Hypermedia Model into the Standard Reference Model for Intelligent Multimedia Presentation Sytems. *Computer Standards and Interfaces*, 18(6-7):497–508, 1997.
- [78] Patrick Hayes. RDF Semantics. W3C Recommendation, 10 February 2004.
- [79] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In *The Semantic Web - ISWC 2006*, pages 272–285, November 2006.
- [80] J. Hunter and L. Armstrong. A Comparison of Schemas for Video Metadata Representation. In *The Eighth International World Wide Web Conference*, pages 353–373, Toronto, Canada, May 11-14, 1999.
- [81] Jane Hunter. Adding Multimedia to the Semantic Web — Building an MPEG-7 Ontology. In *International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, July 30 - August 1, 2001.
- [82] David Huynh, David Karger, and Rob Miller. Exhibit: Lightweight structured data publishing. In *16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007. ACM.
- [83] International Organization for Standardization. Information Processing — Text and Office Information Systems — Standard Generalized Markup Language (SGML), 1986. International Standard ISO 8879:1986.
- [84] International Organization for Standardization. Information Technology — Hypermedia/Time-based Structuring Language (HyTime), August 1997. International Standard ISO 10744:1997 (HyTime Second Edition).
- [85] 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.
- [86] ISO/IEC. Information technology – Coding of moving pictures and audio, 1999. International Standard ISO/IEC 14496:1999 (MPEG-4).
- [87] ISO/IEC. MPEG-21 Overview v.5. ISO/IEC JTC1/SC29/WG11/N5231, Shanghai, October 2002.

- [88] ISO/IEC. Overview of the MPEG-7 Standard (version 8). ISO/IEC JTC1/SC29/WG11/N4980, Klagenfurt, July 2002.
- [89] S. Kim, H. Alani, W. Hall, P.H. Lewis, D.E. Millard, N.R. Shadbolt, and M.J. Weal. Artequakt: Generating Tailored Biographies with Automatically Annotated Fragments from the Web. Presented at the Semantic Authoring, Annotation and Knowledge Markup (SAAKM) 2002 Workshop at the 15th European Conference on Artificial Intelligence (ECAI 2002), Lyon, France.
- [90] Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark H. Butler, and Luu Tran. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation, January 2004, 2004.
- [91] Donald E. Knuth. *TeX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley Publishing Company, 1986.
- [92] Donald E. Knuth. *Digital typography*. Center for the Study of Language and Information, Stanford, CA, US, 1999.
- [93] Lev Kuleshov. *Kuleshov on film: Writings by Lev Kuleshov*. University of California Press, 1974.
- [94] Carl Lagoze and Herbert Van de Sompel. The Open Archives Initiative: Building a low-barrier interoperability framework. *JCDL2001*, 2001.
- [95] Leslie Lamport. *LaTeX - A Document Preparation System*. Addison-Wesley Publishing Company, 1985.
- [96] P. Lemmens and G.J. Houben. XML to XML through XML. In W. Fowler and J. Hasebrook, editors, *Proceedings of WebNet 2001, World Conference on the WWW and Internet*, pages 772–777, Orlando, USA, October 2001.
- [97] Craig Lindley, Jim Davis, Frank Nack, and Lloyd Rutledge. The application of rhetorical structure theory to interactive news program generation from digital archives. Technical Report INS-R0101, CWI, January 2001.
- [98] Suzanne Little, Joost Geurts, and Jane Hunter. Dynamic Generation of Intelligent Multimedia Presentations through Semantic Inferencing. In *6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 158–189, Pontifical Gregorian University, Rome, Italy, September 2002. Springer.
- [99] Mark Lutz. *Programming Python*. O’Reilly & Associates, Inc., 1st edition, 1996.
- [100] Macromedia. Flash. <http://www.macromedia.com/software/flash>.
- [101] Clara Mancini. *Cinematic Hypertext. Investigating a New Paradigm*. IOS Press, Amsterdam, 2005.
- [102] 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.

- [103] Oscar Rosell Martinez. Design dependencies within the automatic generation of hypermedia presentations. Master's thesis, Technical University of Catalonia, June 30, 2002. Published as CWI technical report INS-R0205.
- [104] L. Masinter. The 'data' URL scheme. RFC 2397, August 1998.
- [105] Mark T. Maybury. Planning multimedia explanations using communicative acts. In Mark T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 59–74. AAAI Press, 1993.
- [106] S. Mazzocchi and P. Ciccarese. Welkin RDF Browser, 2007.
- [107] Microsoft. Microsoft Office Homepage. <http://www.microsoft.com/office/>.
- [108] Inc. Microsoft. Internet Explorer 6, 2002.
- [109] Paulo Moura. *Logtalk – Design of an Object-Oriented Logic Programming Language*. PhD thesis, Universidade da Beira Interior, 2003.
- [110] Frank Nack and Lynda Hardman. Denotative and Connotative Semantics in Hypermedia: Proposal for a Semiotic-Aware Architecture. *New Review of Hypermedia and Multimedia*, 7:7–37, 2001.
- [111] Frank Nack and Adam T. Lindsay. Everything You Wanted to Know About MPEG-7: Part 1. *IEEE MultiMedia*, pages 65–77, July - September 1999.
- [112] Frank Nack and Adam T. Lindsay. Everything You Wanted to Know About MPEG-7: Part 2. *IEEE MultiMedia*, pages 64–73, October - December 1999.
- [113] Frank Nack and Wolfgang Putz. Designing Annotation Before It's Needed. In *Proceedings of the 9th ACM International Conference on Multimedia*, pages 251–260, Ottawa, Ontario, Canada, September 30 - October 5, 2001.
- [114] P. Naur and B. Randell, editors. *The NATO Software Engineering Conferences*, Garmisch, Germany, October, 7-11, 1968. NATO Science Committee.
- [115] Ulf Nilsson and Jan Maluszynski. *Logic, Programming and Prolog*. John Wiley and Sons Ltd, 2nd edition, 1995.
- [116] OMG (Object Management Group) . Unified Modeling Language (UML), version 1.4, 2001.
- [117] OpenOffice.org. OpenOffice.org. <http://www.openoffice.org/>.
- [118] Oratrix. GRiNS multimedia presentation authoring software. <http://www.oratrix.com/GRiNS/index.html>.
- [119] Roger T. Pédaque. Document: Form, Sign and Medium, As Reformulated for Electronic Documents. Available at:http://archivesic.ccsd.cnrs.fr/documents/archives0/00/00/05/94/index_fr.html.
- [120] Benoît Pellan and Cyril Concolato. Authoring of scalable multimedia documents. *Multimedia Tools and Applications*, 43(3):225–252, 2009.

- [121] Eric Prud'hommeaux and Ryan Lee. W3C RDF Validation Service. <http://www.w3.org/RDF/Validator/>.
- [122] Dave Raggett. HTML 3.2 Reference Specification. W3C Recommendation, Januari 14, 1997.
- [123] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation, April 24, 1998.
- [124] RealNetworks, Inc. RealPlayer G2. See <http://www.real.com/products/player/>.
- [125] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
- [126] Lloyd Rutledge, Martin Alberink, Rogier Brussee, Stanislav Pokraev, William van Dieten, and Mettina Veenstra. Finding the Story — Broader Applicability of Semantics and Discourse for Hypermedia Generation. In *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, pages 67–76, Nottingham, UK, August 23–27, 2003. ACM, ACM Press.
- [127] 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.
- [128] Lloyd Rutledge, Jacco van Ossenbruggen, and Lynda Hardman. Making RDF Presentable – Integrated Global and Local Semantic Web Browsing. In *The Fourteenth International World Wide Web Conference*, pages 199–206, Chiba, Japan, May 2005. IW3C2, ACM Press.
- [129] Ferdinand De Saussure. *Course in General Linguistics*. McGraw-Hill, 1983.
- [130] Guus Schreiber, Alia Amin, Lora Aroyo, Mark van Assem, Viktor de Boer, Lynda Hardman, Michiel Hildebrand, Borys Omelayenko, Jacco van Ossenbruggen, Anna Tordai, Jan Wielemaker, and Bob J. Wielinga. Semantic annotation and search of cultural-heritage collections: The multimedial e-culture demonstrator. *J. Web Sem.*, 6(4):243–249, 2008.
- [131] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval: the end of the early years. *IEEE trans.*, 22(12):1349–1380, 2000.
- [132] Cees Snoek and Marcel Worring. Multimodal Video Indexing: A Review of the State-of-the-art. *Multimedia Tools and Applications*, 25(1):5–35, 2005.
- [133] The Apache Software Foundation. Cocoon web development framework. See <http://cocoon.apache.org>, 1999.
- [134] The Apache Software Foundation. XSP Logicsheet Guide. See <http://xml.apache.org/cocoon/userdocs/xsp/logicsheet.html>, 1999.

- [135] Raphaël Troncy. Integrating Structure and Semantics into Audio-visual Documents. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Second International Semantic Web Conference (ISWC2003)*, pages 566 – 581, Sanibel Island, Florida, USA, October 20-23, 2003. Springer-Verlag Heidelberg.
- [136] Raphaël Troncy, Werner Bailer, Michael Hausenblas, Philip Hofmair, and Rudolf Schlatte. Enabling Multimedia Metadata Interoperability by Defining Formal Semantics of MPEG-7 Profiles. In *Semantic Multimedia - SAMT 2006*, pages 41–55, Athens, Greece, December 6-8, 2006.
- [137] Henri van de Waal. the Iconclass iconographic classification system.
- [138] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup language, 2001.
- [139] Jacco van Ossenbruggen. *Processing Structured Hypermedia — A Matter of Style*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, April 10, 2001.
- [140] Jacco van Ossenbruggen, Joost Geurts, Frank Cornelissen, Lloyd Rutledge, and Lynda Hardman. Towards Second and Third Generation Web-Based Multimedia. In *The Tenth International World Wide Web Conference*, pages 479–488, Hong Kong, May 1-5, 2001. IW3C2, ACM Press.
- [141] Jacco van Ossenbruggen, Joost Geurts, Lynda Hardman, and Lloyd Rutledge. Towards a Formatting Vocabulary for Time-based Hypermedia. In *The Twelfth International World Wide Web Conference (WWW2003)*, pages 384–393, Budapest, Hungary, May 20-24, 2003. IW3C2.
- [142] Jacco van Ossenbruggen, Lynda Hardman, and Lloyd Rutledge. Hypermedia and the Semantic Web: A Research Agenda. *Journal of Digital Information*, 3(1), August 2002.
- [143] Jacco van Ossenbruggen, Lynda Hardman, and Lloyd Rutledge. Towards Smart Style: Combining RDF Semantics with XML Document Transformations. Technical Report INS-E0303, CWI, October 2003.
- [144] Jacco van Ossenbruggen, Frank Nack, and Lynda Hardman. That Obscure Object of Desire: Multimedia Metadata on the Web (Part I). *IEEE MultiMedia*, 11(4):38–48, October – December 2004. based on <http://ftp.cwi.nl/CWIreports/INS//INS-E0308.pdf>.
- [145] Guido van Rossum, Jack Jansen, K. Sjoerd Mullender, and Dick C.A. Bulterman. CMIFed: A Presentation Environment for Portable Hypermedia Documents. In *The First ACM International Conference on Multimedia*, pages 183–188, August 1993.
- [146] M. van Welie and G.C. van der Veer. Pattern Languages in Interaction Design: Structure and Organization. In *Proceedings of Interact '03*, pages 527–534, 2003.
- [147] R. Vdovjak, F. Frasinca, G.J. Houben, and P. Barna. Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering*, 2(1 and 2):3–26, 2003.
- [148] Visual Resources Association. Visual Resources Association Website.

- [149] W3C. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C Recommendation, June 15, 1998. Edited by Philipp Hoschka.
- [150] W3C. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendations are available at <http://www.w3.org/TR>, February 22, 1999.
- [151] W3C. XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0. W3C Recommendation, January 26, 2000.
- [152] W3C. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. Work in progress. W3C Working Drafts are available at <http://www.w3.org/TR>, 15 March 2001. Edited by Graham Klyne, Franklin Reynolds, Chris Woodrow and Hidetaka Ohto.
- [153] W3C. Device Independence Principles. Work in progress. W3C Working Drafts are available at <http://www.w3.org/TR>, 18 September 2001. Edited by Roger Gimson, co-edited by Shlomit Ritz Finkelstein, Stéphane Maes and Lalitha Suryanarayana.
- [154] W3C. Extensible Stylesheet Language (XSL) Version 1.0. W3C Recommendation, 15 October 2001, 2001.
- [155] W3C. Synchronized Multimedia Integration Language (SMIL 2.0) Specification. W3C Recommendation, August 7, 2001. Edited by Aaron Cohen.
- [156] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004. Edited by Dan Brickley and R.V. Guha.
- [157] W3C. Web Ontology Language (OWL) - Overview. W3C Recommendation, 10 February 2004.
- [158] W3C. Delivery Context: Client Interfaces (DCCI) 1.0. W3C Candidate Recommendations are available at <http://www.w3.org/TR>, December 21, 2007. Edited by Keith Waters, Rafah A. Hosn, Dave Raggett, Sailesh Sathish, Matt Womer, Max Froumentin, Rhys Lewis and Keith Rosenblatt.
- [159] L. Weitzman and Kent Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proceedings of the second ACM international conference on Multimedia '94*, pages 443–451, San Francisco, October 15 - 20, 1994.
- [160] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-Based Infrastructure for RDF: Scalability and Performance. In *The SemanticWeb - ISWC 2003*, pages 644–658, Sanibel Island, Florida, USA, October 20-23, 2003. Springer-Verlag Heidelberg.
- [161] Bob Wielinga, Jan Wielemaker, Guus Schreiber, and Mark van Assem. Methods for Porting Resources to the Semantic Web. In *1st European Semantic Web Symposium (ESWS 2004)*, Heraklion, Greece, May 10-12, 2004 (to be published).
- [162] Wikipedia. Wikipedia, the free encyclopedia. website, 2005.
- [163] M.A. Windhouwer, A.R. Schmidt, and M. L. Kersten. Acoi: A system for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & Services*, Yogyakarta, Indonesia, November 1999.

- [164] Marcel Worryng, Carel van den Berg, Lynda Hardman, and Audrey Tam. System Design for Structured Hypermedia Generation. In C. Leung, editor, *Visual Information Systems*, number LNCS 1306 in Springer-Verlag Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1997.

About the Author

Joost Geurts was born on October 11, 1975 in Gouda, the Netherlands. He received his Engineering degree in Computer Science from Saxion Universities of Applied Science in 1998 (Enschede Netherlands). He continued his studies at the University of Amsterdam where in 2002 he obtained his Master degree in Artificial Intelligence specializing in Logic and Language. During his studies he held a part-time position as engineer in the Hypermedia group of the national research center on computer science and mathematics (CWI) where he developed an interest for multimedia documents on the Web. This interest was followed up in his PhD research carried out jointly at CWI in Amsterdam and the Technical University of Eindhoven (TU/e) during which he developed a model and software framework for applying document engineering techniques to multimedia documents. For his research he engaged in a 6 month visit to the DSTC research laboratory in Brisbane, Australia, and a 6 month visit to the University of California in Berkeley. Currently he holds a position as expert engineer working on multimedia search technology at the IMEDIA project team at INRIA Rocquencourt.

SIKS Dissertation Series

====
1998
====

- 1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
- 1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
- 1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations
within the Language/Action Perspective
- 1998-4 Dennis Breuker (UM)
Memory versus Search in Games
- 1998-5 E.W. Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

- 1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products
- 1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
- 1999-3 Don Beal (UM)
The Nature of Minimax Search
- 1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects
- 1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven
Specification of Network Information Systems
- 1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
- 1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent
Mechanism for Discrete Reallocation.

====
2000
====

- 2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
- 2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
- 2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.
- 2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
- 2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.
- 2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
- 2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
- 2000-8 Veerle Coup (EUR)
Sensitivity Analysis of Decision-Theoretic Networks
- 2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
- 2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture
- 2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

- 2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
- 2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
- 2001-3 Maarten van Someren (UvA)
Learning as problem solving
- 2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets

- 2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
- 2001-6 Martijn van Welie (VU)
Task-based User Interface Design
- 2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
- 2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
- 2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes
- 2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design
- 2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

- 2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
- 2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
- 2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
- 2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
- 2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents
- 2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain
- 2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
- 2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas
- 2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
- 2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

- 2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
- 2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
- 2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
- 2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
- 2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
- 2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
- 2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

- 2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
- 2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
- 2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
- 2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
- 2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach
- 2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
- 2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
- 2003-08 Yongping Ran (UM)
Repair Based Scheduling
- 2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour
- 2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
- 2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
- 2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

- 2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models
- 2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
- 2003-15 Mathijs de Weerd (TUD)
Plan Merging in Multi-Agent Systems
- 2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
- 2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing
- 2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

- 2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
- 2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
- 2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
- 2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
- 2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
- 2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
- 2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
- 2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politieële gegevensuitwisseling en digitale expertise
- 2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
- 2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
- 2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
- 2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

- 2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
- 2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
- 2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
- 2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
- 2004-17 Mark Winands (UM)
Informed Search in Complex Games
- 2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
- 2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
- 2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
- ====
2005
====
- 2005-01 Floor Verdenius (UvA)
Methodological Aspects of Designing Induction-Based Applications
- 2005-02 Erik van der Werf (UM)
AI techniques for the game of Go
- 2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
- 2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
- 2005-05 Gabriel Infante-Lopez (UvA)
Two-Level Probabilistic Grammars for Natural Language Parsing
- 2005-06 Pieter Spronck (UM)
Adaptive Game AI
- 2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
- 2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
- 2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
- 2005-10 Anders Bouwer (UvA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
- 2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

- 2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
- 2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
- 2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
- 2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
- 2005-16 Joris Graaumanns (UU)
Usability of XML Query Languages
- 2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
- 2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks
- 2005-19 Michel van Dartel (UM)
Situated Representation
- 2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
- 2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

====
2006
====

- 2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
- 2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
- 2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems
- 2006-04 Marta Sabou (VU)
Building Web Service Ontologies
- 2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines
- 2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling
- 2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering
- 2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
- 2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

- 2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
- 2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
- 2006-12 Bert Bongers (VU)
Interactivation - Towards an e-ecology of people, our technological environment, and the arts
- 2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
- 2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
- 2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
- 2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
- 2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
- 2006-18 Valentin Zhizhkin (UVA)
Graph transformation for Natural Language Processing
- 2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
- 2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
- 2006-21 Bas van Gils (RUN)
Aptness on the Web
- 2006-22 Paul de Vrieze (RUN)
Fundamentals of Adaptive Personalisation
- 2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
- 2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
- 2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
- 2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
- 2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
- 2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

- 2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

- 2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
- 2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
- 2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
- 2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
- 2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
- 2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
- 2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
- 2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
- 2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
- 2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
- 2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
- 2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
- 2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
- 2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
- 2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
- 2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
- 2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations
- 2007-19 David Levy (UM)
Intimate relationships with artificial partners
- 2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
- 2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use:
A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
- 2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramirez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====
2008
====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying: Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior:
Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

- 2008-18 Guido de Croon (UM)
Adaptive Active Vision
- 2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
- 2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht.
Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven
- 2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
- 2008-22 Henk Koning (UU)
Communication of IT-Architecture
- 2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
- 2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
- 2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
- 2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
- 2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
- 2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
- 2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
- 2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
- 2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval
- 2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
- 2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues
- 2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining
- 2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

- 2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

- 2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
- 2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
- 2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
- 2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
- 2009-06 Muhammad Subianto (UU)
Understanding Classification
- 2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
- 2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
- 2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
- 2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications
- 2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web
- 2009-12 Peter Massuthé (TUE, Humboldt-Universität zu Berlin)
Operating Guidelines for Services
- 2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems
- 2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
- 2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense
- 2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
- 2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
- 2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System
- 2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
- 2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controlling Influences on Decision Making
- 2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification
- 2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
- 2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

- 2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations
- 2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping
- 2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
- 2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
- 2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
- 2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
- 2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
- 2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
- 2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
- 2009-33 Khiết Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
- 2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
- 2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
- 2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
- 2009-37 Hendrik Drachslar (OUN)
Navigation Support for Learners in Informal Learning Networks
- 2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
- 2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets
- 2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
- 2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
- 2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
- 2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
- 2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink
Work flows in Life Science