Design
Implementation and Integration

Issues in an Enterprise Environment
Stellingen

behorende bij het proefschrift

Open, Reusable and Distributed Components for Multimedia Information Management: Design, Implementation and Integration Issues in an Enterprise Environment

Gábor Szentiványi

1. Alle belanghebbenden bij multimedia–informatiemanagement hebben een verschillend gezichtspunt en benadrukken verschillende, vaak conflicterende aspecten van multimedia.


3. Efficiëntie en flexibiliteit zijn geen elkaar wederzijds uitsluitende kwaliteiten van een softwaresysteem. Echter, een ontwerp van een systeem dat zowel efficiënt en flexibel is, benodigt aanzienlijk meer tijd en capaciteit.

4. Grady Booch definieert software–engineering als the creëren van de illusie dat een systeem eenvoudig is. Indien we de weg van Microsoft volgen is deze definitie bevredigend. Een goed ontworpen systeem creëert niet alleen de illusie, maar is zelf ook eenvoudig.

5. Rigide systemen voor de eindgebruiker zijn in sommige gevallen productiever dan flexibele systemen. Waarom dan niet slechts rigide systemen bouwen? Het antwoord is simpel: rigiditeit houdt iets anders in voor een ieder van ons. Een flexibel systeem kan altijd worden geconfigureerd om rigide te zijn op veel manieren, terwijl een rigide systeem nooit flexibel kan worden; niet eens rigide op een andere manier.

6. Indien wij componentware beschouwen als de oplossing voor de meeste
software-engineering problemen, realiseren wij ons dat momenteel, componentware slechts een andere marketingvorm is voor ongewenste leveranciers-binding.

7. Software zal nooit een solide basis zijn van onze informatie-maatschappij zolang kwaliteitsborging wordt gerealiseerd door een softwareprodukt te vermijden dat crasht tijdens een demonstratie.

8. De meeste mensen spreken over conventionele systemen en termen, zoals besturingssystemen, middle-ware, clients, toepassingen, omdat dit hen zekerheid geeft. Er is echter geen gemeenschappelijke begripsvorming meer rond deze termen, omdat de geschiedenis deze heeft gediversifieerd en verstoord. Het is daarom raadzaam om de termen te definieren die wij gebruiken. Uiteindelijk is niet de naam van belang, maar zijn betekenis.

9. Volgens Geoffrey Moore is 'het overbruggen van de kloof' de meest kritieke stap voor een nieuwe technologie om te worden overgenomen door 'modale' klanten. Linux heeft de kloof overbrugd, en het is daarom nu slechts een kwestie van tijd voor werelddominantie.

10. De opinies, zoals dat mensen een test-laboratorium van vele miljarden dollars nodig hebben om softwarekwaliteit te waarborgen, en dat diversiteit van software-oplossingen de klant verwarren, zijn allen foutief gebleken: de 'Bazar' is nu gewapend, en klaar om te winnen van de 'Kathedraal'.

11. De gezamenlijke gevolgen van de Microsoft anti-monopolie rechtszaak en de Open-Source beweging zijn kritiek voor een wereld zonder points-of-no-return, ongewenste leveranciers-binding, software-patenten en andere onnatuurlijke verschijnselen.

12. De situatie is hopeloos maar niet ernstig.
Propositions

attached to the thesis

Open, Reusable and Distributed Components for Multimedia
Information Management: Design, Implementation and
Integration Issues in an Enterprise Environment

Gábor Szentiványi

1. All stakeholders of multimedia information management have a different
view and emphasise different, often clashing aspects of multimedia.

2. Multimedia information management often suggests the existence of
multimedia information that needs to be managed. The very soul of
information is being multimedia, therefore the term multimedia can only
be attributed to how information is managed, rather than to information
itself.

3. Efficiency and flexibility are not mutually exclusive qualities of a
software system, however, the design of a system that is both efficient
and flexible takes considerably more time and resources.

4. Grady Booch defined software engineering as the creation of the illusion
of a system being simple. If we go the Microsoft way, this definition is
satisfactory. A well–designed system not only creates the illusion, but is
also simple itself.

5. Rigid systems for the end–user are in some cases more productive than
flexible ones. Why not just build rigid systems then? The answer is
simple: rigidity means something else for all of us. A flexible system can
always be configured to be rigid in many ways, whereas a rigid system
can never become flexible, not even rigid in another way.

6. If we consider componentware as the solution to most software–
engineering problems, we realize that currently, componentware is just
another marketing form of vendor lock–ins.
7. Software will never be a solid foundation of our information society if quality assurance is realized only by avoiding a software product being a show-stopper.

8. Most people talk in terms of conventional systems and terms, such as operating systems, middlewares, clients, applications, because that gives them certainty. However, there usually is no common understanding of these terms anymore, because history diversified and distorted them. It is therefore advisable to define the terms we are using. Finally, it is not the name that matters, but its meaning.

9. According to Geoffrey Moore, crossing the chasm is the most critical step for a new technology to be adopted by mainstream customers. Linux has crossed the chasm, therefore, now, it is just a question of time for world domination.

10. The opinions, such as people need a multibillion dollar test-lab to assure software quality, and diversity of software solutions confuse the customer have all proven completely wrong: the bazaar is now armed and ready to win against the cathedral.

11. The joint impact of the Microsoft anti-trust case and the Open-Source movement is critical to a world without point-of-no-returns, vendor-lock-ins, software patents and other unnatural phenomena.

12. The situation is hopeless but not severe.
Open, Reusable and Distributed Components for Multimedia Information Management:
Design, Implementation and Integration Issues in an Enterprise Environment
Open, Reusable and Distributed Components for Multimedia Information Management: Design, Implementation and Integration Issues in an Enterprise Environment

Proefschrift

Ter verkrijging van de graad van doctor aan de Technische Universiteit Delft op gezag van de Rector Magnificus prof.ir. K.F. Wakker voorzitter van het College voor Promoties, in het openbaar te verdedigen op maandag 9 oktober 2000 om 10.30 uur

door

Gábor SZENTIVÁNYI

Master of Science in Electrical Engineering geboren te Budapest, Hongarije
Dit proefschrift is goedgekeurd door de promotor:
Prof. Dr.-Ing. habil. W. Gerhardt- Häckl

Samenstelling promotiecommissie:
Rector Magnificus, voorzitter
Prof. Dr.-Ing. habil W. Gerhardt- Häckl, Technische Universiteit Delft
Prof. dr. ir. J. van Katwijk, Technische Universiteit Delft
Prof. dr. ir. E. Backer, Technische Universiteit Delft
Prof. dr. ir. J. Biemond, Technische Universiteit Delft
Prof. Dr. H. Meiöner, Universität Bremen
Prof. Dott.-Ing. R. Zicari, Johann Wolfgang Goethe Universität Frankfurt am Main
# Contents

1 Introduction .................................................. 13
   1.1 The Story .............................................. 13
   1.2 The User’s Perspective .................................. 15
   1.3 The System Architect’s Perspective ...................... 17
   1.4 The Conceptual Designer’s Perspective ................... 19
   1.5 Problem Statement: What the Initial Phase of the Project Taught Us .... 21
   1.6 What are the future guidelines for the project? ............ 22
   1.7 Goals and Architecture of the Thesis ..................... 22

2 Fundamental Principles of Software Systems .................. 23
   2.1 The Goal of This Chapter ................................ 23
   2.2 General Terminology ..................................... 23
      2.2.1 Models and Meta-levels .............................. 23
      2.2.2 Modeling in Context .................................. 24
      2.2.3 Domains ............................................. 25
      2.2.4 Models and Architectures in Domains ................ 27
      2.2.5 Actors in Domains .................................... 28
      2.2.6 The Role of Domain Terminology in This Work ......... 29
   2.3 The Object Paradigm ...................................... 29
      2.3.1 Basic Object-Oriented Concepts ...................... 29
      2.3.2 Object Models ....................................... 30
      2.3.3 Domains of the Object Paradigm ..................... 31
         2.3.3.1 Object Modeling .................................. 31
         2.3.3.2 Object-Oriented Programming ..................... 31
         2.3.3.3 Object Runtime .................................. 32
   2.4 Components ............................................ 32
      2.4.1 Component Assets and Architectures ................ 33
      2.4.2 Granularity and Scalability ......................... 34
      2.4.3 Genericity .......................................... 34
      2.4.4 Configurability ...................................... 34
# Analysis of Multimedia Information Management

## 3.1 The Goal of This Chapter

## 3.2 Analysis Fundamentals

### 3.2.1 Short History and Main Kinds of Media

### 3.2.2 A Brief Overview on Information Management

### 3.2.3 Multimedia Information Management

### 3.2.4 The Role of the Stakeholders in Multimedia Information Management

## 3.3 Multimedia Information Management Domains

### 3.3.1 Domains and Environments

### 3.3.2 Content and Meta-data in Multimedia-oriented Domains

### 3.3.3 Analysis Viewpoints

#### 3.3.4 Content Processing and Compositing

##### 3.3.4.1 The Conceptual Designer's Role

##### 3.3.4.2 The User's Role

##### 3.3.4.3 The System Architect's Role

##### 3.3.4.4 Extending Issues

### 3.3.5 Content-based Indexing and Retrieval

#### 3.3.5.1 The Conceptual Designer's Role

#### 3.3.5.2 The User's Role

#### 3.3.5.3 The System Architect's Role

#### 3.3.5.4 Extending Issues

## 3.3.6 Document Management and Workflow

#### 3.3.6.1 The Conceptual Designer's Role

#### 3.3.6.2 The User's Role

#### 3.3.6.3 The System Architect's Role

#### 3.3.6.4 Extending Issues

## 3.3.7 Continuous Media Storage and Streaming

#### 3.3.7.1 The Conceptual Designer's Role

#### 3.3.7.2 The User's Role

#### 3.3.7.3 The System Architect's Role

#### 3.3.7.4 Extending Issues

### 3.3.8 Medialization

#### 3.3.8.1 The Conceptual Designer's Role

#### 3.3.8.2 The User's Role

#### 3.3.8.3 The System Architect's Role

#### 3.3.8.4 Extending Issues
CONTENTS

3.3.9 User Interfaces ........................................... 60
  3.3.9.1 The Conceptual Designer’s Role ......................... 61
  3.3.9.2 The User’s Role ...................................... 62
  3.3.9.3 The System Architect’s Role ......................... 62
  3.3.9.4 Extending Issues .................................... 63
3.3.10 Multimedia Publishing and Presentation ..................... 64
  3.3.10.1 The Conceptual Designer’s Role ....................... 64
  3.3.10.2 The User’s Role ...................................... 65
  3.3.10.3 The System Architect’s Role ......................... 66
  3.3.10.4 Extending Issues .................................... 67
3.4 Conclusion on Multimedia Domains ................................ 68
  3.4.1 Context .................................................. 68
  3.4.2 The Multimedia Application Developer’s View .............. 68
  3.4.3 The Conceptual Designer’s View ......................... 69
    3.4.3.1 Comprehensiveness .................................. 69
    3.4.3.2 Consistence ........................................... 69
    3.4.3.3 Content and Meta-Data .............................. 71
    3.4.3.4 Approaches .......................................... 72
  3.4.4 The System Architect’s View ............................... 72
    3.4.4.1 Amoeba ............................................... 73
    3.4.4.2 Glue .................................................. 73
    3.4.4.3 Army .................................................. 73
3.5 Wrap up: Selecting an Approach ................................ 73

4 Common Abstractions in Multimedia Information Management ....... 75
  4.1 The Goal of This Chapter .................................. 75
  4.2 The User’s Abstractions .................................... 75
    4.2.1 Content, Metaphor and Interaction ...................... 76
    4.2.2 Embedding Abstractions into the Enterprise Environment 77
  4.3 The System Architect’s Abstractions .......................... 78
    4.3.1 Devices .................................................. 78
    4.3.2 Services ................................................ 80
    4.3.3 Embedding Abstractions into the Enterprise Environment 80
  4.4 The Conceptual Designer’s Abstractions ....................... 81
    4.4.1 Independent Structure .................................. 81
    4.4.2 Appearance .............................................. 81
    4.4.3 Activity ................................................ 82
4.4.4 Conceptual Media and Resources .............................................. 82
4.4.5 Typing and Classification Of Conceptual Media ...................... 83
4.4.6 Embedding Abstractions into the Enterprise Environment .......... 83
4.5 Basic Media-Types ................................................................. 83
  4.5.1 Media-Types as Common Abstractions .................................. 83
  4.5.2 Image ................................................................. 84
    4.5.2.1 Structure ......................................................... 84
    4.5.2.2 Appearance ..................................................... 84
    4.5.2.3 Activity ......................................................... 85
    4.5.2.4 Device Content ............................................... 85
    4.5.2.5 Resources ....................................................... 85
  4.5.3 Text ................................................................. 85
    4.5.3.1 Structure ......................................................... 86
    4.5.3.2 Appearance ..................................................... 86
    4.5.3.3 Activity ......................................................... 86
    4.5.3.4 Device Content ............................................... 87
  4.5.4 Graphics ................................................................. 87
    4.5.4.1 Structure ......................................................... 87
    4.5.4.2 Appearance ..................................................... 87
    4.5.4.3 Activity ......................................................... 87
  4.5.5 Animation ................................................................. 87
    4.5.5.1 Appearance ..................................................... 87
    4.5.5.2 Activities ....................................................... 88
    4.5.5.3 Resources ........................................................ 88
  4.5.6 Video ................................................................. 88
    4.5.6.1 Structure ......................................................... 88
    4.5.6.2 Appearance ..................................................... 88
    4.5.6.3 Activity ......................................................... 88
  4.5.7 Sound ................................................................. 88
    4.5.7.1 Structure ......................................................... 88
    4.5.7.2 Appearance ..................................................... 88
    4.5.7.3 Activities ....................................................... 89
  4.5.8 Music ................................................................. 89
    4.5.8.1 Structure ......................................................... 89
    4.5.8.2 Appearance ..................................................... 89
    4.5.8.3 Activities ....................................................... 89
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.8.4 Device Content</td>
<td>89</td>
</tr>
<tr>
<td>4.5.8.5 Resources</td>
<td>89</td>
</tr>
<tr>
<td>4.5.9 Speech</td>
<td>89</td>
</tr>
<tr>
<td>4.5.9.1 Structure</td>
<td>89</td>
</tr>
<tr>
<td>4.5.9.2 Appearance</td>
<td>90</td>
</tr>
<tr>
<td>4.5.9.3 Activities</td>
<td>90</td>
</tr>
<tr>
<td>4.5.9.4 Device Content</td>
<td>90</td>
</tr>
<tr>
<td>4.5.9.5 Resources</td>
<td>90</td>
</tr>
<tr>
<td>4.6 Composite Media-Types</td>
<td>90</td>
</tr>
<tr>
<td>4.6.1 Temporal Composition</td>
<td>90</td>
</tr>
<tr>
<td>4.6.2 Spatial Composition</td>
<td>90</td>
</tr>
<tr>
<td>4.6.3 Activity Composition</td>
<td>91</td>
</tr>
<tr>
<td>4.6.4 Collection Composition</td>
<td>91</td>
</tr>
<tr>
<td>4.6.5 Hypermedia Composition</td>
<td>91</td>
</tr>
<tr>
<td>4.7 Media-Type Conversions</td>
<td>91</td>
</tr>
<tr>
<td><strong>5 Designing Multimedia Components</strong></td>
<td>93</td>
</tr>
<tr>
<td>5.1 The Goal of This Chapter</td>
<td>93</td>
</tr>
<tr>
<td>5.2 Model Basics</td>
<td>93</td>
</tr>
<tr>
<td>5.2.1 Commotion Object (CmtO)</td>
<td>94</td>
</tr>
<tr>
<td>5.2.2 Media Object (MO)</td>
<td>94</td>
</tr>
<tr>
<td>5.2.3 Data Object (DO)</td>
<td>95</td>
</tr>
<tr>
<td>5.2.3.1 Structural Elements</td>
<td>95</td>
</tr>
<tr>
<td>5.2.3.2 Granularity of the Structure</td>
<td>95</td>
</tr>
<tr>
<td>5.2.3.3 Attributes</td>
<td>96</td>
</tr>
<tr>
<td>5.2.3.4 Typical Structure Kinds</td>
<td>96</td>
</tr>
<tr>
<td>5.2.3.5 Structure Behavior</td>
<td>96</td>
</tr>
<tr>
<td>5.2.4 Device Object (DevO)</td>
<td>97</td>
</tr>
<tr>
<td>5.2.5 Device Content Object (DCO)</td>
<td>97</td>
</tr>
<tr>
<td>5.2.6 Stream Object (SO)</td>
<td>97</td>
</tr>
<tr>
<td>5.2.7 Appearance Object (AppO)</td>
<td>98</td>
</tr>
<tr>
<td>5.2.7.1 Visual Appearance</td>
<td>98</td>
</tr>
<tr>
<td>5.2.7.2 Audio Appearance</td>
<td>99</td>
</tr>
<tr>
<td>5.2.7.3 Temporal Appearance</td>
<td>99</td>
</tr>
<tr>
<td>5.2.7.4 Interactive Appearance</td>
<td>99</td>
</tr>
<tr>
<td>5.2.8 Activity Object (ActO)</td>
<td>99</td>
</tr>
<tr>
<td>5.2.8.1 Presentation: Rendering and Interaction</td>
<td>99</td>
</tr>
</tbody>
</table>
CONTENTS

6 Realization and Integration ................................................................. 113
  6.1 The Goal of This Chapter ............................................................ 113
  6.2 Synergy Fundamentals Revisited .................................................. 113
  6.3 Levels of Integration ................................................................. 114
  6.4 Integration into Existing Enterprise Architectures ......................... 114
    6.4.1 Script-based Architectures .................................................. 114
    6.4.2 Active Control-based Architectures ...................................... 115
    6.4.3 Mixed Architectures and Interoperability ................................ 116
  6.5 Abstract Architecture .............................................................. 118
  6.6 The Active Mapping Process ..................................................... 119
  6.7 An Integration Architecture ..................................................... 119

7 Validation, Conclusions, Outlook ................................................... 123
  7.1 The Goal of This Chapter ............................................................ 123
  7.2 Validation Rules ........................................................................... 123
  7.3 Validation from the Conceptual Designer's Perspective ..................... 123
    7.3.1 Abstract model and its Reifications ........................................ 124
    7.3.2 Model Genericity .................................................................... 124
    7.3.3 Domain Reuse ........................................................................ 124
    7.3.4 Integration Support ............................................................... 124
  7.4 Validation from the User's Perspective .......................................... 125
    7.4.1 Generic Issues ....................................................................... 125
    7.4.2 Specific Metaphor Issues ....................................................... 125
  7.5 Validation from the System Architect’s Perspective ......................... 125
    7.5.1 General Issues ....................................................................... 126
    7.5.2 Specific Issues ....................................................................... 126
  7.6 Conclusions .................................................................................. 127
  7.7 Outlook ......................................................................................... 128
Chapter 1

Introduction

"A fool with a tool is still a fool."
(James Coplien)

Multimedia information management is a multi-faceted domain with no clearly defined borders and application fields. In order to strive for clarity, there must be a definition of multimedia information management that is used throughout this work. However, an in-depth definition cannot be given at the beginning for the reason of multimedia information management having different meanings for all their stakeholders. Therefore, in order to give a precise and comprehensive definition of multimedia information management, all the major stakeholders as well as their roles need to be examined. The following chapters describe the initial phase of a virtual project, from the major stakeholders', that is, the user's, the designer's and the architect's perspective.

1.1 The Story

ACME Corp., a well-known software company has been contracted for the development of a novel multimedia system for the central police department, because the existence of such a system became inevitable. The major requirements were the seamless but non-intrusive integration with their existing information system, uniform management of the video, image, audio and text material they possess, and the straightforward extendibility of multimedia capabilities as the system evolves.

Consultants of ACME led discussions with several experts at the police department in order to figure out the capabilities of their current information system. They found out that the existing system is based on an RDBMS to store cases and personal data. The video and audio material is stored on tapes, the textual material is stored in traditional folders classified by investigations. It is obvious that with the increasing complexity of a case, it is getting more difficult to have an overview on all the material attached to a case and to explore interrelationships between separate sets of data.
After having analyzed current business processes for the police department, the chief consultant worked out a use case presenting the capabilities of system to be developed:

The group is alarmed to visit a crime spot. At the spot, the usual investigation process takes place: Technicians equipped with still and video cameras make recordings of the victims, the spot itself and its surroundings. Inspectors talk to witnesses and record their statements about the crime on tape. An answering machine is found on the site, its tape could contain useful hints. Some fingerprints are also found; they are saved on a special material, it may reveal the identity of the persons present at the crime scene. One inspector makes sketches of the spot and its surroundings. Another inspector notices that there is an unusual object in the store room: he finds another dead body. Unfortunately, the body has been lying there for about fifty years, it only has a skeleton and rests of cloth fragments.

Processing of the collected materials begins after arriving back at the police department. The video and audio materials found or recorded are digitized now by the technicians, non relevant parts are cut out (e.g., the camera been forgotten to be switched off while carried around). After doing the raw cut, the chief inspector edits the material. First, he annotates it with basic information, such as time and location of recordings etc. Then, he separates the video clips into segments containing different observations. Within each segment, he first identifies cuts and annotates them with additional information. The audio material is loaded into a speech recognition system to convert the essential statements of witnesses into text. Fingerprints are scanned and stored in the database for further analysis. The tape of the answering machine is also digitized, followed by the separation of different speakers. Handwritten material (e.g., a diary, a notice book) found at the spot is scanned. The sketches made at the spot are checked against an official geographic information system and detailed plans of the location. The skeletal body and the recent victim are sent to the prosecutor. The skeleton is scanned with a 3D scanner, whose data is used to reconstruct the original body.

After digitizing and annotating the material, identification and recognition of the objects follows. If the material recorded at the spot has explicit annotations, such as the name of a suspicious person or witness, the serial numbers of stolen equipment or the name of the street where the crime happened, the information system is queried for further information on identity. If the query is successful, a link between the newly obtained material and the existing information is created. The semantics of the link expresses the relationship, such as «depicts», «may depict» etc. If there is no explicit annotation available, a recognition system is used to identify objects in the material. The recognition system analyses handwritings, fingerprints, voices, facial shots and tries to make similarity matches based on existing material. The recognition process may be automated, interactive or a hybrid process. The similarity matches may reveal explicit information, such as the identity of a person, whose photograph was found to be similar to a photo in the police database, but it can also reveal implicit information, such as a link between the unknown victim of this case and the unknown assailant of another case.

If most of the relevant links among the collected material are exploited, the chief inspector may decide to involve querying external information systems, such as GISes, banking systems or newspaper archives. A GIS might be useful the check alibis based on time and distance data available. Reports of banking systems on suspicious transactions involving certain accounts might help localize money transfer. Newspaper archives can contain journalist discoveries uncovered or unnoticed by the police earlier.

After having found relationships between materials gathered, the chief inspector decides to hold a meeting for the group inspectors, which are involved with the case. He creates a presentation with all the relevant information gathered and found in the initial phase. When the group is familiar with the case, the chief inspector assigns tasks among inspectors. During the investigation process, the chief inspector gathers the daily reports and tries to find further interconnections. When the investigation reaches a certain milestone, a press conference is held.

If necessary, the police uses real-time information sources, such as GPS devices and surveillance cameras that have been secretly mounted in the environment of suspicious persons or spots. These devices can trigger alarms based on several conditions, such as the observed person passing through a zone, the appearance of a person in a give place etc. After having resolved the case, the chief inspector makes a final report containing all material and evidence relevant to the case and hands them over to court.

In order to summarize the textual description, the consultants also depicted some use cases. When the company agreed with the consultants on the initial use case, they started working out the details. They realized soon that due to the diversity of the information and use case variants to be handled, the user, who possesses domain knowledge but lacks general information management knowledge can easily be overwhelmed with new concepts. In order to keep the number of basic concepts low, the consultants examined the use case from the user's perspective first.
1.2. THE USER’S PERSPECTIVE

1.2 The User’s Perspective

The elaboration of the use case led the consultants to defining a few guidelines that emphasize what is essential for the user when managing multimedia information:

1. similarity to real-life artifacts: A major approach to make multimedia information management user friendly is to provide similarity to real-life artifacts. The means to do this should be metaphors, which are computerized concepts or tangible digital artifacts resembling their real-life counterpart. When inspectors view a video clip on their screen, they can control its behavior through schematized VCR controls. Another example is the hypertext or web metaphor that is based on the interactive activation of additional information about the artifact that has been selected. Although metaphors provide an excellent interactive “surface” to manage multimedia assets, the occurrence of too many artifacts can lead to confusion, because the simultaneous facilitation of several metaphors needs additional coordination which is not present in the metaphors themselves.

2. recurring schemas\(^1\): In several situations, humans act according to schemas. If those action schemas are recurring, we call them patterns. Patterns are useful to boost productivity, because the costs to adopt a pattern to a certain situation is much less then creating a customized solution from scratch. Patterns, from human thinking point of view are nothing else but knowledge classified and reified as actions in certain circumstances. In multimedia systems, most common action schemas include the storage, presentation, deletion and manipulation of media related information. Schemas for managing multimedia information should be applicable orthogonally to the metaphor through that the information is viewed or manipulated. Orthogonality contributes to the simplification and reduction of concepts associated with metaphors and actions.

3. free determination of abstraction levels: Different users of a multimedia information management system have different needs what concerns the granularity of information and its management. A technician might want access video at the level of frames, as opposed to inspectors, who might view video as a movie in a movie player. Free determination means that information should be hidden by abstraction, not by encapsulation in order to flexibly support accessing all levels on demand. Therefore, access control to information needs to be provided as a control layer on the top of management, rather than as an intrinsic part of modeling information.

4. consistence between metaphors: If the user views or manipulates information through interacting with several metaphors simultaneously, a crucial requirement is to keep the metaphors consistent, so that changes made through one metaphor update information viewed through other metaphors.

5. integration with existing business systems: Since a multimedia system in an enterprise, in our case the police, rarely stands in its own, there is a need for integrating it with the existing informational infrastructure or business system. The integration from the user’s perspective has the major advantage that the user only has to interact with one consistent system, instead of handling two isolated ones. This improves the efficiency and the comprehendability of the overall process.

After setting up the guidelines, the consultants involved application designers to find the essential metaphors the user can interact with. They set the basic requirement as to support the use cases and follow the guidelines. The application designers found five metaphors that cover all the essential communication types between the user and the multimedia system:

**DOCUMENT:** The artifacts recorded at the spot of the crime document the observations the police made. When the technicians load the material (by first digitizing it if necessary) into the multimedia

\(^1\)These are schemas in general sense having nothing to do with DBMS schemas.
system, they must annotate it in order to make it referable. The simplest form of referable annotation is unique naming, like “interviewing the witness who has seen the murderer running away”. However, there are other essential annotations the inspectors might add to the artifacts, such as time and place of recording, textual description of the artifacts etc. To be more generic, annotations are just a special case of composition which is the basic notion behind documents. When further relationships are discovered among the artifacts (such as “the knife appearing next to the victim on one image is the same knife found in a newspaper article two years ago”), they can be composed to express semantic coherence. The document metaphor defines a logical or semantic container in that digital artifacts, their annotations and their interrelations reside. The document metaphor is a generic concept swallowing more specific metaphors, such as hypermedia and can be subdivided into genres.

**Circuit:** Some of the documents are static, for instance containing a still picture of the victim or the objects around the victim. However, some are changing in time, such as the video recorded, the tape found in the answering machine or the surveillance camera mounted at a suspicious meeting spot of criminals. In order to present all these documents, they need to be streamed from their source (storage location or camera) to the user’s screen along a certain path. A generic metaphor, the circuit covers all aspects of delivering time-aware media information, let it be either natural or synthetic, or live or replayed. The intrinsic control of the stream is the behavior of the circuit metaphor. It is separated from interactive control, which is deferred to the manipulator metaphor described below.

**Manipulator:** The police spokesman holds a press conference showing a preorchestrated presentation of the multimedia artifacts collected at the crime scene, rather than assembling the presentation from raw material during the talk. The presentation does not allow for much interaction, the user is either passive or reactive (restricted to starting, stopping and temporal positioning) during its course. The slideshow and movie metaphors are adequate for providing passive or reactive interaction.

Technicians created an animation that shows the act of the crime reconstructed: the murderer enters the room and fires his gun to kill his victim. The body of the other victim has also been reconstructed from its skeleton in the form of a fully rendered 3D model that can be inspected on the screen. The animation and the rendered 3D model can be explored by changing the view angle and the zoom. Screen objects possess handles that let the user manipulate the models. In case of the 3D model, the naked body can be dressed with different clothes and her hairstyle can be altered. In case of the animation, the persons can be placed in different locations and the direction of the gun can be adjusted before firing it. In these scenarios, the user actively manipulates, rather than passively views the media artifacts. A further distinction in manipulation is the proactive (e.g. browser metaphor) and directive (e.g. canvas metaphor) behavior.

**Medialization:** As mentioned earlier, the technicians visualized the results of the ballistic analysis by creating an animation that showed a reconstruction of how and where the gun was fired, how the bullet entered the body of the victim and how he fell on the floor. In other cases, if the visual medium is already heavily loaded, non-media information can also be sonificated. A simple example is a monotonous sound, whose frequency correlates with the tracked person’s distance from the edges of a territory he is not supposed to leave. A more sophisticated and common sonification is text-to-speech conversion. When chasing a suspicious person, a text/data-to-speech converter can be used to complement or replace the navigator, since it can translate encoded messages coming from a GPS and a navigation system to short, auditory messages, which do not distract the driver from driving. The generic metaphor medialization covers the creation of artifacts of any medium (visual, audio, tactile) from either non-perceptible information (e.g., from encoded data to animation) or another medium (e.g. from text to speech).
1.3. THE SYSTEM ARCHITECT'S PERSPECTIVE

**QUERY:** An important witness saw the murderer running away and she can remember most of the murderer's face features. An inspector assembles a phantom picture of the murderer based on the description and collaboration of the witness. This is medialization, since abstract feature descriptions are turned into tiles of the image that make up the face. Now, the face can be queried against all known faces in the image repository of the police. If similar faces can be retrieved, it might help identify unknown but important persons. The generic query\(^2\) metaphor is based on the notion of sending a request to the system to find assets satisfying conditions or containing certain features.

After having found the basic metaphors, consultants and analysts revisited the use cases to cover all activities needed and to be based on the metaphors described above. They also revisited the guidelines and realized that metaphors themselves cannot provide support for the last three guidelines (abstraction levels, integration and consistence). This is due to the fact that metaphors represent too high level and fixed abstractions. Complying to all guidelines would require a mechanism that is common to all current and future metaphors. They expected these issues to be solved by system architects.

1.3 The System Architect's Perspective

System architects were assigned the task to implement applications along with the metaphors via reusing and adopting existing, or implementing non-existing architectural components. They knew the importance of a reference model for the architecture that could be developed and deployed in order to implement the metaphors upon it. However, in order to implement a reference model for the architecture, all architectural components must be examined. Therefore, architects examined architectural components available for multimedia. They found that components can be classified into five categories based on the functionality they provide:

**PERSISTENCE:** There is a wide range of architectural components that provide persistence for data. The simplest way of storing data is the usage of commodity filing systems. Filing systems provide two functional components: naming and contiguous view on a one dimensional storage space. Relational database management systems (RDBMS) provide more semantics for data, namely the relational model, and functional components, such as querying and transactions. Object-relational DBMSs (ORDBMS) provide a simple object model upon the relational model to account for more flexible semantics. Object DBMSs (ODBMS) dismiss the relational concept and store objects in their natural form, which is proves to be more suitable for many features of object models. Media servers possess a rudimentary model similar to that of filing systems, but filing systems have generic physical layout policies for both read and write operations, whereas media servers have an optimized layout for playout.

**TRANSACTION:** Transaction management introduces a new quality in distributed architectures. Unlike local transactions that are bound to persistence, distributed transactions focus on activities that span several local transactions simultaneously. The functional components of distributed transactions are resource management (e.g. connection pooling) and two-phase commit (2PC). Multimedia systems require a special handling of transactions. This is due to the fact that (1) different aspects of the same information are usually stored using different persistent mechanisms (e.g. a media server not supporting transactions), and (2) the full ACID properties of transactions might not be replaced by more suitable versioning for the management of multimedia data (e.g., real-time manipulation of video). Traditional distributed transaction management is accomplished by transaction processing monitors (TP monitors), which needs to be extended to support the above issues.

\(^2\)Note that the query metaphor is strictly constrained to read-only access, it excludes the adding, deletion and modification of information, as opposed to common query languages, like SQL.
CHAPTER 1. INTRODUCTION

ACCESS: The access of multimedia information underlies size and integrity considerations. There exist three kinds of basic access mechanisms: a multimedia artifact can be transferred by reference, by value and by streaming. Transferring by reference is useful if it is not economical to transfer an artifact due to its unmanageable size (e.g., a query returning ten video clips). Access by value is useful when the artifact has a relatively small size and its content represents a consistent whole, such as an image. Access by streaming is essential when the media artifact is rather large in size, therefore it has to be split into blocks that have an inherent relationship (spatial, temporal) with respect to each other. Blocks are then transferred by value, such that their relationships are constantly accounted for. Important mechanisms that complement the access mechanism are prefetching, buffering and caching. Architectures that provide access in a distributed environment are based on network protocols. Access by reference can be accomplished by open protocols, such as RPC or IIOP. Access by value can also be accomplished by IIOP or HTTP. Access by streaming is either done by the rudimentary generic protocol HTTP, or special proprietary protocols, such as RealAudio [45] are used. There are also endeavors for standardizing open control protocols for streams, such as the CORBA A/V streams service [4].

USERDEFAULT: Presentation and manipulation of multimedia data are the means, through which the user communicates to the whole system. Some typical user interface types for multimedia include stream players, document browsers and active controls. Stream players control dynamic, streamed information, document browsers let the user explore structured documents, and active controls are front-ends for direct manipulation of multimedia information. Players usually offer a rudimentary API to control the stream. Browsers are restricted to black-box rendering of scripts. Active controls embed functionality into the client host environment, their functionality therefore depends on the host environment (e.g. JFC [49]). Since all of these interface-kinds have shortcomings, complex interfaces need a combination of the above.

OVERALLDEFAULT: Overall solutions combine two or more architectural components into a facility that can be reused as a whole system. Overall architectural solutions severely constrain the choice for languages, APIs and exchange formats, but provide a higher level of reuse. Typical solutions are media servers (e.g. Oracle Video Server [28]) that combine persistence, transfer and presentation components. Another example are mono/multimedia DBMSs that merge the storage and the query components, characteristic examples are Jasmine [24] or Informix [46] DBMSs. Audio, video and image editing systems are another good example for the combination of the storage and manipulation components, examples include Matrox’s DigiSuite [55].

In order to support the decision about reuse versus implementation of architectural components, system architects set up three basic guidelines:

1. level of reuse: Reuse of architectural components is based mainly on implementation reuse. There are two contradictory issues in the selection of architectural components. On the one hand, architectural components should be as specific as possible in order to maximize implementation reuse and minimize the cost of mapping applications to the architecture. On the other hand, architectural components need to be as generic as possible in order not to constrain application design, and to leave more room for scalability and maintenance (see below).

2. scalability of the architecture: Scalability copes with quantitative changes in the system. A golden rule in architectural design is that after defining the architecture (probably based on a reference model) it shall be left intact. To provide resilience to change, the scalability must be intrinsic to the architecture itself. There are two main problems to tackle: upsizing and downsizing. From architectural point of view, distribution of architectural components is a straightforward solution for upsizing. Disadvantages are
increased complexity (e.g. need for middleware), run-time costs (e.g. DBMS run-times on a per CPU basis) and resource needs (e.g. unwanted non-linear upscale). In order to eliminate these disadvantages, for the system to scale up, its generic purpose components (e.g. DBMS) must be configurable to be downsized to provide a specific service (e.g. naming service) with minimal overhead, and to optimize component interoperability.

3. maintainability of the architecture: Maintainability copes with qualitative changes in the system. We can distinguish between design-time, configuration-time and run-time maintainability. Often, it is not straightforward to select the right type of maintenance. Most experience has been collected in design-time maintainability, several methodologies, such as the object oriented methodology [77] gives excellent support to it. Configuration-time maintainability is the customization of implementation components in a certain environment. Component based methodologies [77] provide sound basics for how to achieve this. Run-time maintainability is often characterized by hot-pluggable components. Run-time maintainability has two key assets: (1) the system does not need to be deactivated for the replacement of a component, and (2) a component to be maintained does not need to be deactivated for modifications.

After having studied the available architectural components, architects tried to select them for realization according to the guidelines. It sounded reasonable that reuse should be based on enabling technology, rather than product suites. Reusing complete product suites at this complexity results in vendor lock-in, moreover, maintainability and scalability are severely constrained. Although partitioning the system into smaller, autonomous, more generic components represents a lower level in implementation reuse, it allows for finer grain management. With the number of components becoming higher, interoperability of components gains more and more importance. In order for autonomous architectural components to interoperate, they have to dispose of common syntax and semantics. First the syntax needs to be defined, because semantics can only be expressed in terms of a syntax.

They could easily rule out persistence, since the syntax of persistence engines focuses on expressing state, which is an internal feature, as opposed to a syntax that can be used to express features exposed for interoperability. A further problem of persistent engine syntax is the lack of constructs to express transient semantics. User interface architectures can also be ruled out, since their syntax expresses a partial view, rather than a common semantics to all components. Distributed transactions focus on providing optimal usage of distributed resources mainly in terms of throughput, several levels lower than needed. Manipulation, such as user interface is not generic enough to provide syntax that can be used to describe common semantics. Access is the only architectural component that is generic, and it is inevitable for all the other architectural components, therefore, it is the natural component for providing syntax.

System architects came to the conclusion that besides the common syntax, in order to enable (1) the general notion of interoperability between architectural components and (2) flexible maintainability by non-destructive replaceability of components, there also needs to be a notion of common semantics that is shared among all architectural components.

After having a common team meeting, the team realized that they independently came to the same conclusion about (1) having common semantics for making their entities (architectural components and metaphors, respectively) interoperate, and (2) being this beyond their responsibility. Therefore, project leaders suggested an additional group with a distinct responsibility: conceptual designers shall be responsible for finding the syntax and defining the common semantics for application designers and system architects.

1.4 The Conceptual Designer’s Perspective

The first task of conceptual designers was to examine the findings of the other two groups. Conceptual designers found that the common semantics for metaphors and architectural components should be the same, even though the role of semantics is different for the two. The common semantics abstracts the functionality
Figure 1.1: The missing chain between metaphors and the architecture: an initial abstraction on the story of architectural components into services, whereas it refines the metaphors into more manageable, generic assets. This approach complies with the guidelines for both architects and users.

Generic assets can express the common aspects of all metaphors. Using generic assets, metaphors can be consistently reengineered at several abstraction levels. Generic assets can express recurring schemas in a generic way, which can be used orthogonally to each metaphor. Non-perceptible information can be converted to generic assets and communicated to the user consistently, regardless of the metaphor.

Services that abstract architectural components can support scalability by allowing for lightweight as well as robust implementations to realize the same service. Services that abstract architectural components also support maintainability by allowing for configuring services without the need to know the implementation of the service.

The common semantics can also be a key enabler of the integration of the existing business system and the multimedia system. Interoperability can be accomplished between the business logic and the common multimedia semantics. This provides higher consistency and easier maintainability than data exchange at architectural level or view provision at metaphor level. Figure 1.1 shows the common semantics as the missing chain between the architecture, the metaphors and the existing business system.

In order to facilitate maximum design reuse, conceptual designers first tried to exploit existing models that express the common semantics for multimedia services and metaphors in a formal way. After a thorough analysis, they classified the models they came across into four categories:

**Languages:** The creation of models through languages has been one of the major concerns of multimedia. The languages that have been defined for multimedia are mostly scripting languages, interpreted by a «black box». Scripting languages possess several advantageous features, such as simplicity, constraining semantics, run-time execution optimization etc. In multimedia, these languages are typically used for describing presentations, such as SMIL [19], DAMSEL [62] or configurations, such as CINEMA [13].

**APIs:** Application Programming Interfaces are means to embed the capabilities of a system into a given language environment. APIs are provided for accessing certain functionality from a language, such as ODBMS language bindings (e.g., ODMG API [16]), or the Java Media Framework API [73].
ExchangeDefault: Exchange formats enable simplex communication between loosely coupled parts of a distributed system. In simple cases it might be enough to apply the messaging paradigm, but in multimedia, messages tend to have vast content, temporal semantics and fine grained structure. Exchange formats allow for exchanging complex information retaining the advantages of messaging, such as autonomy and asynchronity. Examples of popular exchange formats for multimedia are text-based markup formats, such as XML\textsuperscript{3} [21], which is tailored to represent tree-like structures typical for structured documents.

AbstractDefault: Abstract models cannot be deployed directly. However, they do not necessarily need a programming language, an API or an exchange format to become tangible. A more flexible approach is to use an independent notation as an intermediate step to express the model and its schemas. Appropriate tools can map the model on a specific architecture by mapping it into languages, APIs and exchange formats. One of the most widespread abstract models is UML [39].

After having analyzed existing models, the designers were confronted with the fact that they were not able to find a generic model that could be used to express common semantics. This was due to two facts: the models they encountered were either tightly bound to a language or format, or, although being abstract, covered just a small fraction of the metaphors or services they needed.

1.5 Problem Statement: What the Initial Phase of the Project Taught Us

After the initial phase of the project, due to the several shortcomings found, project leaders organized a meeting in order to summarize the results and to define the future of the project. The shortcomings were summarized as follows:

1. The user is simultaneously confronted with too many metaphors that often reveal contradictory or inconsistent views on multimedia. This hinders the user in developing a consistent mental model of the application or system she uses.

2. Multimedia systems are applications or frameworks that build on specific architectures. Semantics that would abstract architectures in order to connect to applications is usually missing or heavily dependent on the architecture. This makes the design of architecture-independent models and components impossible, resulting in scarce design reuse.

3. Abstract models that can provide a solid foundation for multimedia systems and applications are either constrained to a certain aspect of multimedia or too generic for multimedia reuse.

4. Architectural support for multimedia in mainstream enterprise environments is not sufficient. The main problem is insufficient support for the coherent integration of the document, circuit, manipulator, medialization and query metaphors with middleware architectures.

5. Semantic support for the integration of existing business systems with multimedia is missing; the integration takes place at either data exchange or at view level. Neither of these approaches can provide semantic consistency, which burdens the management and simplicity of architectural components and user interfaces, respectively.

\textsuperscript{3}XML is rather a mixture of language, format and model concepts, but here it is considered as format.
1.6 What are the future guidelines for the project?

After having identified the shortcomings of the existing technology, project managers outlined the roadmap for the future of the project. Three groups: users/consultants, conceptual designers and system architects seem to be inevitable for success, since they all have distinct and key roles in the project. Their responsibilities have been defined as follows:

1. **Conceptual modelers** shall create an abstract model independent of programming languages and architectures, which provides a solid semantic foundation for generic and multimedia specific services, as well as for generic manageable multimedia assets. They shall reify the model in terms of components. Conceptual modelers also shall assure that the abstract model has the semantic power to interoperate with the business system in a straightforward way.

2. **System architects** shall select architectural components and realize the required services using them. They also shall realize the components using those service abstractions as a container. Moreover, they also shall assure that the interoperability with the existing business system is realized through common architectural components.

3. **Users/consultants** shall create application or context specific metaphors from the generic multimedia assets, and realize them in a given environment.

1.7 Goals and Architecture of the Thesis

The overall goal of this thesis is to create generic and highly reusable components for distributed multimedia system development. The introductory chapters above already illustrated the complexity of this topic, as well as the major problems one faces when taking the endeavor to build complex multimedia systems. These problems led the above project to define the future guidelines for success. However, the accomplishment of those tasks necessitates a generalization of the problems found, a thorough investigation of the generalized problem and the synthesis of a solution; all these following a certain methodology.

This thesis identifies several fundamental goals in order to achieve the overall goal. The first fundamental goal is to find a software methodology to support the notation and the process formalism throughout the whole lifecycle of the work. The second goal is to revisit, generalize and reclassify major user concepts with more rigor using an analysis process. These concepts shall lead to a refinement of metaphors to generic multimedia assets. The third goal is to create a consistent conceptual model that provides a solid foundation of multimedia information management. The factoring of this model shall result in abstractions that can resynthesize the metaphors ("height"), and in refinements that can establish services ("depth"). The factoring of the model in both vertical directions shall enable the definition of components as reusable assets in multimedia systems development. The fourth goal of this thesis is the realization of the components on selected architectural components, where services shall constitute the "plugs" between multimedia components and architectural components. The fifth goal of the thesis is the integration and deployment of the components for building complete solutions for enterprise information management. Two case studies, a front-end and a back-end integration shall provide a proof-of-concept for the findings in this work.
Chapter 2

Fundamental Principles of Software Systems

"Method will teach you to win time."

— Goethe

2.1 The Goal of This Chapter

Each domain has its basic terminology, without which no meaningful communication between the participants of the domain can be established. Software development is one of the domains with the most confusion and misunderstanding. In order to establish a strong foundation for the remainder of this work, the concepts, which are relevant shall be defined. There of course exist a number of definitions for each of the terms described below. The intention of this chapter is to select one possible definition to the relevant terms, rather than classifying or evaluating existing definitions. The main guideline is to consider the set of basic definitions as a whole: consistence is the most important feature definitions should possess.

2.2 General Terminology

2.2.1 Models and Meta-levels

We provide an initial definition of model:

**Definition:** A model is a coherent representation of all relevant concepts in the universe of discourse (the target of our investigation). The model represents concepts physically, consistently and fully at the chosen level of precision and viewpoint, but it never aims at completeness or comprehensiveness in an absolute sense, since it would not be possible. Models can represent both existing ("as-is") and future ("to-be") systems.

In order to define our concepts using a model, we need existing, previously known concepts to express our concepts with. These pre-existing concepts are called meta-concepts, their consistent system is a meta-model. Therefore, on the one hand, a model is a certain instantiation of a meta-model, on the other hand, a meta-model is a definition of a model.

A meta-model can define several models, however, a model shall have only one meta-model. Since the instantiation/definition dependence holds for every model-level, we can create models at arbitrary meta-levels recursively. The result will be the creation of meta-meta-model, meta-meta-meta-model etc. It is important to recognize that a meta-model is only called meta-model with respect to its model, therefore, "meta" is a relationship, rather than an absolute identification of a certain model.
2.2.2 Modeling in Context

Since the above chapter defines a model as a physical representation, we can consider anything a model. Although this iterative definition is flexible and comprehensive, due to the \textit{relativeness} and the \textit{continuum} of the models, it cannot easily be used for modeling. Specifying a level as a \textit{reference} would avoid the relativeness, and strongly related meta-levels could be grouped. The specification of a reference-level must include one additional level: the meta-level. The justification of the meta-level is straightforward: whereas the model at the reference-level describes the semantics, the meta-level provides the syntax for describing the semantics. To structure modeling levels, \textit{contexts} are introduced:

\begin{definition}
\textit{Context} is a pair of levels. The model at the reference-level is called the \textit{instance-level} model, the model at the other level is called the \textit{type-level} model. An instance-level and a type-level model together form a context.
\end{definition}

Using contexts, \textit{modeling} can be defined as follows:

\begin{definition}
\textit{Modeling} is the main activity within a context. It is defined as the creation of a \textit{new} instance-level model as an instantiation of an \textit{existing} type-level model. The purpose of the instance-level model is the representation of context-independent concepts using concepts defined in the type-level model.
\end{definition}

Since every model is simultaneously at the instance-level of one context and at the type-level of another context, contexts necessarily overlap. Figure 2.1 shows the iteration of meta-levels along with the contexts. The example below shall demonstrate contexts:

\textbf{Example:} A bank employee needs to describe clients of a bank. She creates a description when people become clients, and deletes the description when they discontinue their relationship with the bank. The set of descriptions is a valid snapshot of the bank's clients at any point in time, we call this a model. She recognizes that all the clients can be described in a similar way, so it is worth creating a meta-model that defines how a client has to be described. She gets a further assignment to extend the description with account information and transaction history of the client, too. Now, she knows that it is sufficient to extend the meta-model once, the model will uniformly reflect the extension in every snapshot. After several extensions, she decides to take a more formal approach for the meta-model, and searches for a modeling language that she can express her meta-model in. This modeling language is the meta-meta-model, and it is specific to the banking environment based on best-practice, such that it efficiently supports the modeling of different entities and processes in the bank. The bank decides to extend its profile and enter the insurance business. She is contacted again and asked to describe insurance-clients as well. The snapshots cannot describe these kind of clients and the meta-model cannot be changed, because it has been created using a bank oriented modeling language that cannot be extended with constructs needed for describing insurance clients. She decides to develop a uniform meta-meta-meta-model, a means to describe the existing bank-oriented modeling language as well as an insurance-oriented modeling language, moreover, it has the potential to incorporate other future modeling languages representing different areas.

This scenario contains three contexts. The level-zero context that contains the snapshots and their definition, the level-one context that contains the banking modeling language and the definition of the snapshots expressed in this language, and the level-two context that contains the uniform modeling language along with the definitions of the the banking as well as the insurance environment languages.
2.2.3 Domains

Systems in real-life differ from the theoretical situation depicted in figure 2.1. Contexts are an adequate means to cope with the complexity caused by meta-levels, since there always is a limit what a human can understand at a given time. However, in real-life systems, this limitation is reflected by establishing domains, rather than by contexts in a seamless model-continuum:

**Definition:** Domain is a field or sphere of activity or influence. Domains are areas dealing with similar problems at similar abstraction levels from similar viewpoints, to which similar solutions need to be found.

Domains have developed historically, reflecting the needs, focal points and limitations defined and maintained by founders and experts active in a particular field. The typical domain structure is depicted in figure 2.2. Every domain possesses «data» that has to be managed within the domain. Data must be created and accessed in a structured way maintaining dependencies between different «data». This is provided by «schemas». In order to create and access «data» through «schemas», «schemas» must be defined in a consistent way. Each «schema» is based on a «model» that defines it. With other words, the primary asset of a domain is «data», «schema» can be thought of as a consistent representation of the assets, and «model» as the language to express the representation.

The sphere of activity that belongs to a certain domain is permanently expanding, implying an increasing complexity of the problems domains have to provide solutions to. Increasing complexity can be coped with by introducing new concepts at either the «model» or the «schema» level within a domain. Without going into details concerning the implications of model and schema extensions on coping with complexity and resilience to change within a domain, the limitations of the traditional domain model structure can be summarized in the following statements:

- a fixed number of levels in the domain structure might not be flexible enough to create complex domains
- a domain dealing with too many levels may easily loose its focus
domains often contain overlapping functionality, domain reuse is scarce and the semantics of domain models are often clashing.

The first two statements are contradictory, and if we also consider the third statement, the only way we can alleviate the limitations is to make simpler and separated domains that interoperate in order to create more complex domains. This approach necessitates two additional notions: domain partitioning and domain interoperability.

Domain interoperability is handled in the following chapter. In domain partitioning, two role-dependencies can be recognized:

application-specific domains: Domains are classified in vertical domains (also called application-specific domains) that are described by functional areas covered by a family of systems (e.g. aircraft navigation system domain), and horizontal domains (also called technology domains) that are described by common functionality with similar requirements across vertical domains (e.g. DBMS, windowing system domains). This classification aims at eliciting reuse across domains through improving reusable assets.

design domains: Domains are classified into design domains that are domains producing models for other domains (target of design) by specifying what domain concepts are present target domain, and realization domains that are domains refining domain concepts by specifying how domain concepts of the target domain are realized in terms of the model of the realization domain. This classification aims at eliciting the separation between design (creational, configurational) and implementation (operational) processes.

The above two classifications are orthogonal, they provide a different view on partitioning domains. Both classifications present dependencies among domains, rather than absolute domain categories. Additionally, since these classifications are relative, domains within one classification can be chained such that one domain takes two roles at a time. This feature is analogous to the overlap of modeling contexts described above.

Although the domain-classifications provide flexibility in the design, implementation and deployment process, they are still rigid to form a generic foundation for software systems. The main reason being that models are only valid within a domain, therefore, if complex domains are replaced with several interoperating domains, there needs to be a model mapping mechanism that enables the «trespassing» of domain boundaries. The next chapter discusses this mapping.
2.2. GENERAL TERMINOLOGY

2.2.4 Models and Architectures in Domains

In order to use models within domains in general in a global scale, domain structure limitations and domain terminology need to be relaxed and generalized, respectively. This can be done in three steps: (1) merging domain and modeling terminology, (2) generalizing domain structures, and (3) making domains interoperable.

As the first step, domain-centric and the context-centric views on modeling need to be merged. Since modeling terminology is more generic, we try to translate domain-centric terminology to the context-centric. The following observations can be made:

1. In case «data» is abstract, «schema» and «model» must also be abstract. This contradicts to our original definition of model, which considers models at any meta-level to be physical representations, thus tangible.
2. In case «schema» and «model» are abstract, «data» must also be abstract, since only a physical notion can instantiate another physical notion.
3. In case «data» is physical, it can only be accessed through a known representation, therefore it makes no sense for «data» to be a separate layer, which however makes «data» also a model\(^1\).
4. In case «data» is physical, «schema» and «model» must also be physical (see item 2), which means that «schema» and «model» can also be considered «data».
5. The «instantiates» and «defines» relationships are inverse, therefore «model»-«schema» and «schema»-«data» relationships are identical, they are just meta-levels of each other, there need not be distinctions between them.

Since the context-centric approach is a more generic view on models, due to the inconsistency found in the observations, the domain-centric approach needs to be led back to the generic terminology of the context-centric approach. The elimination of «model»-, «schema»- and «data» can be accomplished by applying the context-centric view of models along with their meta-levels. Correspondingly, «data» becomes an instance-level model, shortly instance. «Schema» becomes the type-level model, shortly model. «Model» becomes the meta-type-level model, shortly meta-model\(^2\). This transformation only changes the terminology, but preserves relationships by recognizing their inverse semantics.

As a second step, domain structure is generalized. In the traditional domain structure there are three levels. In a generalized domain, there has to be at least one context, that is, two models (instance and type). If the introduction of new levels is necessary, it can be done by contexts, which can be considered as the smallest unit of semantically coherent levels.

As the third step, domain interoperability has to be provided. Interoperability between domains requires the access of models throughout domain boundaries. The interoperation of domains is reified in two notions: mapping and reuse. Mapping for domains is defined as follows:

**DEFINITION:** Domain mapping is a form of interoperability between domains, where a type-level model in the target domain is accessed to create an instance-level model for the representation of another instance-level model in the source domain.

The mapping process has to know the semantics of the target model. In order to achieve consistence as defined above, the mapping usually uses the type-level model to instantiate its target model as an instance-level model.

\(^1\)Quoted «model», «schema» and «data» correspond to domain terminology, as opposed to the unquoted term model, which is the original terminology used throughout this work.

\(^2\)Note that that the above definitions are aligned with most meta-modeling terminologies (e.g. ISO meta-model standard, MOF etc.).
If the target model can fully capture the semantics of the source model, the mapping can be bidirectional. If not, the mapping process has to apply special conventions in order to preserve as much of the semantics as possible. Mapping is typical between design and realization domains.

As opposed to mapping where the one domain is mapped through its model to another domain to be represented there, in case of reuse, one domain reuses the other other domain through its model for its own purposes. Reuse for domains is defined as follows:

**Definition:** *Domain reuse* is a form of interoperability between domains, where an instance-level model in the reused domain is accessed by another instance-level model in the reusing domain for the purpose of the reused domain sharing some aspects of its models with the reusing domain.

Reuse is accomplished through instance-level models, without the involvement of the type-level model. Reuse can also be bidirectional, in this case we call domains fully interoperational. Reuse is typical between horizontal and vertical domains, vertical domains reusing horizontal domains.

With inter-domain dependencies and domain interoperability becoming complex, the overall structure of a software system incorporating several domains needs to be defined in terms of the *architecture*:

**Definition:** *Architecture* is the organizational structure of a system, including its decomposition into domains, the nature and relevant characteristics of the domains, their topology, constraints and interoperability. Architecture defines a style that guides the design and the realization of the software system.

### 2.2.5 Actors in Domains

Actors are stakeholders of a software system. The definitions we give here will be refined several times throughout this work. We start with a general definition of an actor:

**Definition:** *Actor* is defined as a stakeholder of a software system, who carries out a certain activity. Based on domains, there are three actors to be identified: *user, conceptual designer* and *system architect*.

**Definition:** *User* is an actor, who is the primary beneficiary of a domain: she is active in a realized target domain to solve a certain problem the target domain has been created for.

**Definition:** *Conceptual designer* is an actor, who designs an instance-level model in a design domain, which is called the *design*. She does this by instantiating a type-level model, that is, the design language.

**Definition:** *System architect* is an actor, who realizes the design of the conceptual designer by defining an architecture of selected realization domains and mapping the design to those realization domains.

---

3In case of mutual reuse, the domain roles do not hold anymore. This is handled later.
2.2.6 The Role of Domain Terminology in This Work

Figure 2.3 gives the reader an overall view on domains, models and actors in accordance with the domain terminology established and used in this work. The domain terminology described in this chapter takes a major role in domain analysis. However, there are two main reasons for domains not being successful in the management of software systems. The one reason being that there is a lack of accepted standards and uniform management of most domains and specifications of domain boundaries. This leads to overlapping coverage, non-uniform terminology, scarce reuse and interoperability problems. The other reason being that models within domains have not been standardized either, which deepens the problems listed above.

In the remainder of this work, we base our analysis of multimedia information management presented in chapter 3 on the domain terminology described above. A more generic and uniform foundation is also presented here; it is described in the remainder of this chapter. The foundation is based on two advanced concepts orthogonal to domain terminology. These concepts are the object paradigm and components.

2.3 The Object Paradigm

The object-oriented paradigm covers the design, implementation, execution and deployment of systems based on objects. In order to understand objects, their design, implementation, operation and deployment, we first need to explain the basic object-oriented concepts.

2.3.1 Basic Object-Oriented Concepts

The debate on what are the basic object-oriented principles is ongoing. In order to avoid confusion, we define our view on this topic, without attempting to compare different definitions, not even selecting ours based on any formal decision. The two major definitions in this context are devoted to object and type:

*Despite the endeavours (e.g. OMG) to foster the standardization of domains, the statements made on domains still hold in general, especially with respect to multimedia-related domains.*


**Definition:** *Object* is the basic building block of an *object model*: a model that has been created using the object-oriented paradigm. An object is the smallest unit of instantiation within a model and it has a unique identity. The two main features of an object are *state* and *behavior*. An object encapsulates its state and behavior, such that these become *internal* features of the object. An object can be sent messages (also called service requests) of certain kinds by a client, through which the client can manage its state and trigger the behavior of the object.

**Definition:** *Type* is the main feature of an object what concerns its *external* behavior. Type is an abstract classifier that defines the group of messages that can be sent to an object. Type groups objects, to which the same set of messages can be sent. Objects are instances of their (one or more) types.

Generally speaking, there seems to be a consensus on the three basic features of objects underpinning the object-oriented paradigm:

**Definition:** *Inheritance* allows for the incremental definition of an object by inheriting features from objects already defined, modifying these, and appending only those parts that are different. We differentiate between two kinds of inheritance, based on the features inherited. If state and behavior are inherited, we call it *implementation* inheritance, whereas the set of messages the object can receive are inherited, we call it *type* inheritance or subtyping. The main difference between these two inheritance kinds is that whereas implementation inheritance can both modify existing features and append new ones, subtyping can only do the latter.

**Definition:** *Polymorphism* is the ability of something to appear in multiple forms, depending on the context, also the ability of different things to appear the same in a particular context. In object models, polymorphism means (1) that an object implementing a type can be substituted by any of its subtypes without the client that sends the message to the object noticing any difference, or (2) an object that implements several types acts as the type the client expects (in case it is implemented by the target object).

**Definition:** *Dynamic binding* is the deferring the binding of a message to a particular object until runtime, based upon the object receiving the request. Dynamic binding is the main enabling mechanism for polymorphism, it determines the context as the type of the object that receives the message, not the type that is referred to by the client.

2.3.2 Object Models

The beginning of this chapter has dealt with general discussions on models, now we specialize on object models. Object modeling is an up-to-date technique, it superseded data modeling (e.g [66]), which was the major modeling technique for a long time. Object modeling provides all the advantages of data modeling and adds several advanced concepts to improve the expressing power of models and make them more suitable for representing a wide range of concepts. The description of object models presented here has been aligned with UML [39] and Catalyst [32].

In chapter 2.2, the definition of a model was abstract, we only stated that a model is a coherent representation of concepts. In the definition of the object model, we refine the above definition by stating what the representation is and how it achieves coherence:
DEFINITION: Object model is a model in which concepts and their coherence are represented by objects and their relationships, respectively.

In order to describe an object model with respect to the UoD, different views on the model are needed. In different views, objects and relationships represents different concepts. An object model captures a subset of the following views: static, interaction, state-machine and activity.

The static view of an object model represents types, the externally visible behavior of objects, through which objects can be accessed. Types contain information on how to manage the state of objects, that is, to access and to change it, but not how to implement the state. Types also contain information on how to trigger the behavior of objects through methods. The coherence between types are relationships which include inheritance, association and several other dependencies. An association can be considered a state that provides information for an object to access another object.

The interaction view describes the collaboration of objects based on their types. The interaction view considers a group of objects that interact with each other. The interaction can be described either in abstract terms as actions, or by specifying the messages objects sent to each other. The relationship between objects that represents coherence of objects in this view is their collaboration that realizes a higher goal.

The state-machine view, as opposed to the two previous view, focuses on a particular object in isolation. It considers an object's state, that is, on how it changes in time throughout the lifecycle of the object. The change of state is called a transition, the triggering of a transition is an event. The coherence of states within an object is assured by the relationships between states that are transitions.

The activity view is similar to the state-machine view with a different purpose. The activity view is also based on states, but here, states represent the execution of certain parts of the implementation of one or more object's behavior, typically reified as statements. Transitions between activity states represent the completion of an activity and the flow of control to the following activity in either sequential or concurrent fashion. The latter features introduce the concepts of join and fork that synchronize concurrent activities. The relationships that assure coherence between objects is the synchronization of activities between them.

2.3.3 Domains of the Object Paradigm

Some typical domains of the object paradigm are described in the followings.

2.3.3.1 Object Modeling

The object modeling domain is a design domain that produces object models. Object models are produced in a design environment. The design environment provides a certain notation in that the design can be instantiated using the design language.

2.3.3.2 Object-Oriented Programming

Object-oriented programming is a fundamental domain supporting the object paradigm. It is a realization domain for object modeling. This realization domain consists of a programming language using which programs can be instantiated.

The main concept of programming languages is the class. The class is said to be the implementation of a type. Programming languages describe the instantiation and manipulation of objects through classes. These objects then have the type their class implements.
Most object-oriented languages contain an other concept: *interface*. An interface is the realization of a type within the language without an implementation. Interfaces define which messages can be sent to the objects. Therefore, it defines only the external behavior of a class and that for every object that belong to the class. Classes mix external behavior with implementation (e.g. in languages without interfaces, such as C++, this is the only way to go), interfaces have the only task to isolate implementation from external behavior.

There is a broad range of confusion what concerns objects and classes in programming languages. Objects and classes are fundamentally different concepts: objects are instances, whereas classes are programming-domain abstractions. Some environments use the terms «object class» and «object instance», which provides confusion, rather than clarification, since objects are always instances and classes are always abstractions. A proliferation of the confusion is caused by meta-programming. Meta-programming for classes means a single designated class that acts as a meta-class to manage the external behavior of other classes, that is, their interfaces. Meta-classes as any other class can have instances, which are objects. Another source of confusion is that classes are frequently have the name «Object». The latter «alias» is also used in this work, referring to the fact that there always must be in instance (that is, Object) to describe anything, including types and classes.

### 2.3.3.3 Object Runtime

The domain of object-oriented programming programming is again a design environment that is realized in an *object-runtime* domain. The object runtime domain *executes* objects in a certain context. The mapping between these two domains is the compiler that generates executable code for a certain *execution environment*. The only notion object runtimes are aware of is the object\(^5\). The object runtime creates objects with unambiguous localization. The created objects contain their own state, but the implementation of the behavior is shared between objects. Note that the behavior implementation can also define local state which has nothing to do with the state of the object. What makes an object unique is its identity and its state, and these are never shared with other objects. These unique features can be made persistent, such that the particular object can be reconstructed at any time, outliving the runtime context it has been created in.

### 2.4 Components

The object paradigm provides us with powerful, and consistent modeling, programming and runtime domains, however, it does not solve a couple of problems that have either direct or indirect impact on the deployment process in a software system:

1. The implementation and the interface of an object may not be separated (classes without interfaces). This separation cannot be enforced, therefore compromising resilience to change.

2. Dependences between objects can be implicit, especially when using implementation inheritance. This does not allow for independent deployment. Most characteristic problems are the syntactic and the semantic fragile baseclass problem.

3. Self-description of objects depends on the programming language (e.g. Java) or an external run-time facility (e.g. CORBA Interface Repository), rather than an inherent feature of the object.

4. With generic and appropriate modeling languages it seems to be possible to seamlessly model everything from the most to the less abstract concept of a system in a consistent fashion. However, the object paradigm and its domains do not imply the definition of an architecture, that is, a uniform mechanism to partition the model into manageable parts for implementation and deployment in existing environments.

\(^5\) Some runtime structures, such as C++ runtime does not even know about the full object, there only is a notion of dynamic binding through a virtual table (VTBL) and pointers.
5. Modeling techniques and programming languages have been constantly improving (semantics, performance etc.), but mainstream object runtimes are still in their infancy concerning management functionality (e.g. persistence) and architecture (e.g. frameworks).

Components are meant to alleviate the above deficiencies of the domains built on the object-oriented paradigm. Components are a concept of deployment. This does not mean that the concept of components only appears at deployment time. Components are also present in the design and realization of software systems, but their active contribution is to provide a sound base for deployment. In the following chapters we define components and describe the infrastructure that can be built around them.

Components’ role with respect to the object paradigm is twofold: On the one hand, components are orthogonal to objects, because objects are in the design and realization field, whereas components are in the runtime and deployment. Components are not restricted to be designed and realized using the object-oriented paradigm. On the other hand, components are a logical continuation of the object paradigm: objects are a sound foundation for components. First, we give a definition of component:

**DEFINITION:** Component (always in the meaning of software component) is a unit of composition with contractually specified interfaces and explicit context dependencies only. A component can be deployed independently and is subject of composition by third parties.

The basic idea behind components is the provision of dynamically composable building blocks of software throughout the design, implementation and deployment of a software system. For the purposes of independent deployment, a component needs to be a binary unit [77]. To distinguish between the deployable binary unit, and the instances it supports, components are defined to have no mutable state.

Besides the basic features of components described, components alleviate the problems listed above. A summary is given here. Components are composed through interfaces that are completely separated from their implementation. Interface dependencies are always explicit, components rely on interfaces and provide others, but do not implicitly share implementation (e.g. through implementation inheritance). Components describe themselves through explicit semantics as their inherent feature. Components partition object realizations with respect to the architecture of the object runtime. This enables the independent deployment of components on a certain architecture. Besides deployment, components provide flexible execution and management of objects on a particular architecture, without objects needing to depend on particular services and architecture of the object runtime.

### 2.4.1 Component Assets and Architectures

The two main assets of components are reuse and independent extensibility. Reuse can be understood as the attempt to share certain aspects of an approach across various contexts. Independent extensibility means to extend the approach with new aspects in a way that these independent new aspects can be combined. These two assets can be considered as mechanisms on a «substrate» component and «plug-in» components. The substrate component provides reuse because it contains certain aspects that are common and can be shared. Plug-ins support independent extensibility, because they extend the substrate by providing new aspects. The system of substrates and plug-ins is called a component architecture. We give a more relaxed definition:

**DEFINITION:** Component architecture is the overall design of a system. It defines guidelines that together help to achieve the overall targets without having to invent ad-hoc compromises during system composition using components.
In order to achieve an optimal architecture, we have to examine the factors that impact the qualities of reuse and extensibility. In this work we consider four factors: granularity, scalability, genericity and scalability. In the following three chapters, these are described.

2.4.2 Granularity and Scalability

These two factors are mentioned in pair because there is a strong relationship between them. On the one hand, components with finer granularity can be easier associated with a required task they are supposed to be applied for, whereas the chance to find a component with larger granularity to match a complex task is lower, so there likely to appear a certain overhead. As a consequence of this, finer granularity components scale better, because they can be composed together more precisely to provide certain features (design scalability), and it is less effort to replace and distribute a component in order for a system to scale up as well as down (architectural scalability).

On the other hand, there are some other issues involved in the composition of fine-grain components (also of large-scale components, but there, it is of a lesser issue). If more components have to be composed, the communication complexity between components increases. This compromises performance, since intercomponent communication through exposed interfaces is likely to be less efficient than inside components. What compromises performance even more is that components often execute in different protection domains (e.g. processes), and inter-domain communication is usually two magnitudes of order more expensive than intra-domain communication.

2.4.3 Genericity

The usability of a component severely depends on the nature of the service the component offers. If the component is generic, it can be reused for more purposes, however the user of the component must make extra effort to either tailor the component to his needs, or build more specific components that use generic ones. If the component is specific, it can more effectively used for a certain purpose, however, it cannot likely be used for other purposes.

The viability of tailoring depends on the configurability of the component (see later). If the component is not configurable, generic components, while being flexible, cannot enforce consistent semantics, therefore the user of the component must provide bottleneck interfaces to the component that restrict its usage. Generic components are usually less performant and optimizable than specific ones.

2.4.4 Configurability

Configurability is the tailoring of an existing component to the needs of a certain environment, in which the component is deployed. In configuration, we can distinguish three cases\(^6\):

1. \textit{realization-time}: The component is configured as an abstract notion, a given configuration appends and/or modifies generated source code with respect to the existing core. The whole component is then recompiled into a binary with tailored configuration.

2. \textit{deployment-time}: The component is configured as a binary, but there is a distinction between deployment and execution contexts such that runtime switching of these contexts is not applicable.

3. \textit{execution-time}: Configuration on a binary component can be applied concurrently with component execution, without the notion of different contexts. The three methods in the above given order are inversely proportional to the run-time performance of component execution.

\(^6\)Design time configurability would not make any sense for components.
Chapter 3

Analysis of Multimedia Information Management

"Nobody wants a gun, they want a speeding bullet.
Nobody really wants a speeding bullet, they want a bullet hole.
Nobody really wants a bullet hole, they want a dead deer.
Nobody really wants a dead deer, they want a good meal.
Nobody really wants a good meal, they want that wonderful feeling after eating a good meal."

(Rich Gold, Xerox PARC)

3.1 The Goal of This Chapter

Chapter 1 already gave the reader an impression what multimedia information management might mean to different stakeholders of a multimedia system. The chapter even concluded in an initial abstraction on multimedia information management and pinpointed the problems encountered in few highlights. The goal of this chapter is to reexamine multimedia information management. Since the preliminary classifications and decisions made in chapter 1 might have been considered as ad hoc rather than well founded, the need for a comprehensive yet concise analysis has to be satisfied.

In the literature, multimedia is «sliced» in different ways. A common approach is to focus on a group of technology issues that are key to multimedia, e.g. [71], [48]. Another approach takes a more general approach describing media-types, e.g. [70], however there is no clear statement on which basis media-types have been established. Yet other views focus on a particular domain, providing an in-depth analysis on the domain, covering only domain-specific multimedia issues (the corresponding representative literature will be referred to in the domain descriptions below). There are approaches that tackle multimedia components, however, their view is restricted to a certain application domain, usually the programming domain (e.g. [36]).

This work takes a different approach. It attempts to capture all the domains that are relevant to multimedia in order to gain a common understanding of multimedia. The motivation for gaining a common understanding is crucial to establish open and generic components for multimedia that are free of application domain and technology specific issues. The chapter is split into three parts:

Firstly, the fundamentals are described, such as definitions on media and multimedia, and the relationship between information management and multimedia. Finally, the stakeholders of multimedia information management are introduced. These fundamentals are necessary in order to define what it means to be relevant to multimedia.

Secondly, characteristic domains that are relevant to multimedia are described. The description of a domain contains is based on the role of the stakeholders: three separate aspects. This separation helps provide a structured and comprehensive insight into the domain.
Thirdly, a conclusion on the domains is drawn. This conclusion is also structured following its stakeholders. The conclusion is the main trigger for founding the common fundamental abstractions described in chapter 4.

Some might consider this chapter to be too long with respect to its contribution to the whole work. Other than the approach taken here would have been just to provide a conclusion and refer to the corresponding literature concerning all the domains and their concepts. The reason for not having done so is twofold:

On the one hand, the author of this work has not encountered an analysis, which would have examined several domains from a generic multimedia perspective, and from the viewpoints of different stakeholders with special attention to components. Therefore the content of this chapter became an organic part and one main achievement of this work\(^1\).

On the other hand, the conclusions made at the end of the current chapter and the common abstractions defined in the next chapter act more obvious and easy to follow if references given to concepts are described within the same work.

### 3.2 Analysis Fundamentals

The term «multimedia information management» is the combination of the terms «multimedia» and «information management». In order to regard multimedia information management as one coherent term, its roots need to be examined and merged into one consistent, comprehensive notion.

#### 3.2.1 Short History and Main Kinds of Media

After having read chapter 1, its introductory statement «multimedia information management has different meanings for all its stakeholders» became obvious. In order to create order in this versatile field, some fundamental definitions are necessary:

---

**DEFINITION:** *Media* are physical objects that can be easily modulated into different states in order to carry information. *Natural media* are media that naturally exist, such as human voice. *Artificial media* are media that have been artificially created, such as photos, paper and ink, as well as books. *Electronic media* are artificial media that can be modulated by electronic means, such as telephone, television. *Digital media* are electronic media that can be modulated digitally into a finite number of states, such as computer networks, mass storage, framebuffers, multimedia applications, CAVE etc. In case of electronic media, we can distinguish between *perceptible* media, such as a printer, and *non-perceptible* media, such as CD-ROM.

**DEFINITION:** *Artifacts* are things perceived in a particular perceptible medium. Natural artifacts are *artifacts in natural media*. Digital artifacts are *artifacts in digital media*.

**DEFINITION:** *Information* is the means to represent and communicate human knowledge \([2]\). Information corresponds to artifacts, if artifacts are a reflection of a knowledgeable mind. Information only exists if there is a substrate, that is, a medium to carry it and a receiver to perceive it. Information inherently exists only within one certain *medium*.

---

The development of media starting from natural media to digital media has been long. Figure 3.1 depicts the main media classes. The broadest class is media in general; it contains all special media. The only medium

---

\(^1\) The lack of comprehensive views on multimedia becomes obvious if someone looks at leading-edge books on multimedia: more and more of them are loosely coupled collections by different authors.
that exists naturally without the interference of human creativity is the _sensory medium_. The most common sensory media are visual, auditory, tactile and haptic.

_Natural_ media have been fundamental to communicate knowledge through providing permanence over a longer period of time (e.g. drawings in sand, smoke signals etc.), but it have been slowly replaced by media that is more resilient to nature, more energy-effective and can carry information capturing a broader spectrum of human knowledge.

_Artificial media_ exists since the time humans have been trying to conserve and transfer knowledge using media not present in nature. Artificial media facilitates the _capturing_ of natural artifacts for retention (e.g. photograph of our environment, phonograph of our voice) and the _creation_ of artificial artifacts (e.g. drawing symbols on paper with ink). There were two main drives in artificial media becoming prevalent: _realism_ for documentary purposes (e.g. photography replacing portrait painting) and _mass media_ for information dissemination (newspaper replacing handwriting).

First, focusing on the creation of artificial artifacts, most traditional artificial media communicate information to the human through a visual sensory medium. Some of these media _intrinsically combine_ the _presentation medium_ and the _storage medium_, such as paper and ink. However, in most cases, these two media are _separated_ because of (1) the nature of the sensory channel (e.g. auditory) or (2) mass production (e.g. stencil for newspaper press) etc.

If we broaden our view from the artifact creation process to the capturing process, we have to split the presentation medium into two parts: the _input_ presentation medium, which is the medium that captures information from sensory media (e.g. photographic lens and light sensitive material), and the _output_ presentation medium, which emits information into the sensory medium (e.g. a photograph).

Since the _representation_ of information varies along different media, _mapping_ between different media becomes a major issue. Mapping provides room for flexibility in the capturing as well as in the presentation process. The management of flexibility requires knowledge. This knowledge must also be represented by information. The knowledge can be broken down to parameters of the model of the artificial media (e.g. drum/disc revolution and volume in the phonograph, sensitivity of film material), and to the description of the mapping process. The knowledge is represented as _extrinsic_ information, since it depends on knowledge about the media, rather than on knowledge represented by information to be _presented_ by the media itself. In the following, we will see that the extrinsic information and its management play a substantial role in further media kinds, and it impacts the overall qualities and functionality of the management of different media.

_Electronic media_ are the next step in the history of artificial media. Their contribution to media evolution is substantial for many reasons. Firstly, electronic media are neutral. If we consider any non-electronic

---

2The kinds of media as described in this chapter are based on the media definitions of MHEG [3], however, several changes have been applied. Any comparison or reasoning on different definitions is beyond the scope of this work.

3This information is also a reflection of a certain kind of knowledge, but this knowledge is in the technology domain, not in the primary domain (universe of discourse) of the user. As to different kinds of knowledge, the reader finds more material in the section about the user’s abstractions.
media, they usually predefine a sensory channel and suggest a certain way of perception. Electronic media are non-perceptible, therefore they can be conveniently used to store and transfer any information. Information independent storage could have been achieved with non-electronic media, too, but electronic media as carrier are energy-effective, and their energy effectiveness is controllable in fine steps. Secondly, electronic media are rather dynamic that immutable: they can be modulated permanently into different states at a low energy cost. These two main features make electronic media suitable for storing, presenting, and transmitting information over long distances, creating a new medium: the transmission medium.

As mentioned above, electronic media can be used to carry any kind of information. As media and information representation in media become complex, their management demands a new, abstract medium: the representation medium. Representation media are abstractions of (1) the encoding of information in a certain perception medium (intrinsic or primary information), (2) the description the creation or presentation context (extrinsic or secondary information), and (3) the mapping of information throughout several media (extrinsic information). Since these kinds of abstraction depend on the technology (capturing, storage, transmission and presentation), context (perception) and goals (e.g. quality, energy, costs), their realization varies heavily. For instance, in analog television, the representation medium is a TV-standard, like PAL, SECAM or NTSC. Complex capturing (e.g. TV camera) and presentation media (e.g.TV sets) agree on these representations for mapping information from one to the other.

Electronic representation media enables a new kind of information production besides capturing: synthesis. Presentation media do not distinguish between captured or synthetic information in representation media, since in the electronic representation there is no such difference. Having said that, there are two main differences in general: in the quality and in the behavior of information. However, in analog media, these differences cannot be taken advantage of.

There exists a major drawback with analog electronic media, which presents a barrier for their flexible usage. Since electronic media are non-perceptible, if used for retention and transfer, two mappings are necessary, one for capturing, one for presentation. This means that presentation is a reproduction of information that has been captured*. Information in analog media is difficult to reproduce, because analog media are very sensitive to noise, moreover, the resolution of the carrier cannot be influenced. Therefore carrying information in analog media can distort the knowledge it represents without direct notice or the possibility to counteract. This is especially true for extrinsic information, which is a cornerstone of information fidelity. An excellent example of this is the artistic way of configuring an analog synthesizer. Besides this deficiency in analog media being art creating, it is the main a barrier for the automation of multimedia information management.

Digital media possesses all the advantageous features of analog media, but it has a very attractive new feature: digital media, unlike analog media carries information in a deterministic precision. The deterministic precision enables for the representation medium to carry information of arbitrary complexity, and also provides the reconstructability of arbitrary complex information. These make digital media an adequate foundation for automating the management of information in different media.

From the different media listed above, the user's common "interface" with digital media is the sensory medium. In the remainder, we do not facilitate this medium, because we assume that the enterprise environment, into which we place our management, generates context for managing media. In a certain context, there always is a reason for a human to use a medium for carrying information. Therefore, there also always is an information perception process when there is a sensation process. Therefore, we replace sensory medium with an abstract medium: the perception medium. Each perception medium inherently comprises a sensory medium, moreover, also common behavior and intention to create, capture manipulate and present artifacts.

Now, we can return to the differences between captured and synthetic media. With digital media we can achieve arbitrary precision, such that quality becomes a major issue. The difference in quality between captured and synthetic media is only in the eye of the beholder, that is, in the perception media. With appropriate technology, the borders can be blurred (e.g. photorealism). However, there always remains a difference in the behavior of the artifacts. If artifacts have been synthesized, the synthesizer decides the about the behavior

*Disregarding manipulation for simplicity now.
behind perception. With capturing, the behavior is defined, it cannot be changed (e.g. in a synthesized image we might be able to change the viewpoint, in a captured, we cannot). These behavioral differences will be accounted for in the followings.

Since perception medium is not bound to any physical object, we can establish perception media at any abstraction level. There are two fundamental classifications of perception media. We can speak about text, graphics, image, video and animation as visual perception media, and sound, music and speech as audio perception media. Orthogonally to this, we can consider image, video and sound as captured perception media, and graphics, animation, speech and music as synthetic perception media.

DEFINITION: The term monomedium denotes perception monomedium, that is, it refers to the presence of one perception medium. Multimedia has two meanings. The loose definition of multimedia denotes perception multimedia, that is, the simultaneous presence of more than one perception media. The strict definition of multimedia adds a constraint that at least one of the perception media need to be continuous, that is, time dependent.

When referring to the term «multimedia» in the remainder of this work, we use it in the loose meaning, unless explicitly indicated otherwise.

After having described major kinds of media, we can conclude in the notion of media being a fundamental, but not comprehensive abstraction for further analysis. The main reason for this being that it only covers the user's view, rather than being more comprehensive. For the remainder of this chapter, analysis requirements are split among stakeholders.

3.2.2 A Brief Overview on Information Management

Information management has a wide scope, reaching from core technology levels up to strategic levels. In this work, only a brief overview of information management is given, which is sufficient for laying down the basics of multimedia information management. Since the number of different views on information management is endless, we select the view of the Object Management Group (OMG) on information management [38].

According to the OMG, the aim of information management is to enable an enterprise to get good value for its investment in information. The domain of information management can be subdivided into the following areas: information modeling, interchange, storage, retrieval, encoding and representation. In order to position information management in a broader space, OMG mentions vertical market facilities and user interface facilities as two key related areas. From this positioning we can conclude that:

1. information management is of horizontal nature supporting several vertical domains (e.g. manufacturing, simulation, accounting), and

2. interaction/communication with the user is excluded from information management.

3.2.3 Multimedia Information Management

Recently, the term «multimedia information management» has appeared and is being used to emphasize the special issues when dealing with multimedia within information management. However, «multimedia information management» in general carries two major misconceptions.

Firstly, a major part of the literature dealing with multimedia information management is based on separate models and domains for different monomedia (e.g. [35], [60]).

5 Of course, music can also be captured, however the fact that it is music is only in the «ears» of the beholder, for the capturer, it remains sound.
Secondly, information must reside in a medium (in accordance with the definition in chapter 3.2), thus every information is inherently media information. Therefore, the only meaningful definition of «multimedia information management» attributes the modifier «multimedia» to management, rather than to information. With other words, we can speak about managing information in a multimedia-way, but not about managing multimedia information.

Following the previous recognition, we are now focusing on the multimedia-way of information management. Traditional information management is centered around how the representation medium carries information, and puts user communication to the periphery. Multimedia information management, however, puts perception medium in the first place, and derives all other concepts from it (e.g. through the presentation medium), and finally, it leads these concepts back to perception.

If we call perception-based information management multimedia information management, we can call traditional information management business information management. The relationship between business information management and multimedia information management is handled in this work at a later point (chapter 6). In order to make a domain analysis, for now, we just support the above ideas with a single, fundamental definition that both reflects the common and the multimedia-specific features:

**Definition:** Multimedia information management inherits its common features from information management. It defines perception medium as the «common denominator» for information management. Other media, that is, presentation, storage, representation and transmission media are used to uphold, that is, to model, create, store, query, transfer, process and present information carried by several perception media.

3.2.4 The Role of the Stakeholders in Multimedia Information Management

The above definition of multimedia information management represents a general, comprehensive view, which can be used as a solid foundation. However, in order to comprehend the extent and the concepts of multimedia information management deeper, we need to make a multi-faceted domain-analysis. The facets we have chosen are the stakeholders of multimedia information management, that is, the user, the system architect and the conceptual designer. The definitions below reflect the roles of the stakeholders in multimedia information management and extend the definition given in chapter 2.2.5:

**Definition:** The user's role is to deploy multimedia information management to facilitate the expression, communication and presentation of their knowledge by using information in various perception media. The user's access to management is accomplished through the user interface. The user interface provides the user with a consistent look and feel and semantics for accessing information and management functionality, such that the user can develop a consistent mental model of the underlying system.

**Definition:** The conceptual designer's role is to capture all the relevant concepts of multimedia information management and establish a model that describes these concepts consistently. The concepts the conceptual designer establishes are the bridge between the user and the component architecture.

**Definition:** The system architect's role is to provide, that is, create or reuse components that encapsulate information or realize management functionality, and to implement the conceptual model by mapping it on these components.
3.3 Multimedia Information Management Domains

3.3.1 Domains and Environments

Chapter 2.2 defined domains and their terminology in general. In this chapter, we analyse multimedia information management domains. We define domain analysis as follows:

**Definition:** Domain analysis is the process by which information used in the development of software systems is identified, captured and organized with the purpose of making it reusable when creating new systems.

We consider the domains analyzed here to be horizontal domains. This is not an absolute positioning, it solely means that there are vertical domains that reuse the domains analyzed here. The definition of these two different kinds of domain put into the context of multimedia is as follows:

**Definition:** Multimedia-oriented domains are horizontal domains that manage information in a multimedia-way, such that managed information is always primarily associated with a perception medium that carries it. Multimedia-oriented domains provide reusable assets to a family of application domains.

**Definition:** Multimedia-enhanced domains are vertical domains that reuse multimedia-oriented domains in any manners. These vertical domains are usually application domains for multimedia.

Nowadays, there exists an uncountable number of multimedia-enhanced domains. It is beyond the scope of this work to describe them even partially. Moreover, in accordance with our definition of multimedia, every application that communicates with the user is a multimedia-enhanced domain, which broadens the sphere of applications even more. Therefore, in order to keep the scope of this work focused, we deal with multimedia-oriented domains only.

The selected domains are neither complete nor disjunct. Instead of striving for completeness, the goal of this chapter is to cover all the major multimedia-oriented domains. The domains described here might either be overlapping, or reusing parts of other domains' functionality. In order to remain yet consistent, we restrict domains to be as independent as possible: we define core domains that are the targets of the analysis and mention other synergistic ancillary domains that are needed for operation but which are not considered as vital part of the core domain.

In order to limit the validity of our investigations, as indicated in the title of this work, we define the scope of analysis to be the enterprise environment:
**DEFINITION:** *Enterprise environment* is a particular environment in which all the software system, the hardware resources and the domains are situated. In general, this environment is defined by the availability of mainstream hardware and software solutions commonly found or affordable in enterprises. In particular, it refers to the local (intranet), the external (internet), and the global (extranet) networking infrastructure, server machines, workstations, network computers, operating systems.

**DEFINITION:** *Domain environment* is defined as a a group of the core and one or more required ancillary domains that are tailored to the needs of the core domain. In an enterprise environment there can be one or several specific domain environments.

### 3.3.2 Content and Meta-data in Multimedia-oriented Domains

Up to this point, we have already presented several views on domains in general (chapter 2.2) and multimedia-oriented domains is particular (chapter 3.3). There is however, a further, multimedia-specific view on information management. As opposed to the definitions given earlier, this view is more specific, and therefore, cannot be applied to all the domains we are analysing. Despite this, we include it, and will refer to it during analysis where applicable, because it reveals important relationships within and across domains. The basic notion of this view is content and meta-data:

**DEFINITION:** *Content* corresponds to any representation of perceptible media in a multimedia-oriented domain. The exact meaning of content depends on the particular domain. In terms of media, content is defined as the information in the representation medium.

**DEFINITION:** *Meta-data* is any information about content relevant to a particular multimedia-oriented domain. Its definition is also domain-dependent, it depends on the definition or role of content.

The classification given here is a quintessence of several efforts (summarized in [11]), and considered as widely accepted. It can be summarized as follows:

- **content-independent meta-data:** This type of meta-data captures information that does not depend on the content with which it is associated.

- **content-dependent meta-data:** This type of meta-data depends on the content. The information is either automatically extracted from the content or manually annotated. There exist two sub-types:
  - **content-based meta-data:** This type of meta-data is extracted from the content using some algorithm without human interaction. Examples include the color statistics, size of content etc.
  - **content-descriptive meta-data:** This type of meta-data describes content without the direct utilization of the content itself. The creation of these type of information often involves additional human perception and knowledge.
    - **domain-independent meta-data:** This type of meta-data captures information present in content independent of the application domain of the content.
    - **domain-specific meta-data:** Meta-data of this type is described in a manner specific to the application domain of content.
In order to support the above view on meta-data, we provide an example:

**Example:** We start from a video clip as the "raw material" that needs to be extended with meta-data. What concerns content-dependent meta-data, we annotate the video clip with the creator, creation time and the location where the clip was shot. What concerns content-dependent meta-data, we split it into two parts: we add content-based meta-data by extracting features, such as the color histogram of the clip and the number of frames. The other type of content-based meta-data, content-descriptive meta-data we again split into two: we annotate the clip with domain-independent meta-data, such as the main persons that can be recognized in the clip, the duration of their presence etc. We also annotate the clip with domain-dependent meta-data; since we use these clips at the police department's central archive database, we add descriptions of the role of the persons in a particular activity seen in the clip (e.g. victim, beholder, assassin).

### 3.3.3 Analysis Viewpoints

Domain analysis on multimedia information management domains can be done from many, generic and specific viewpoints. The following list gives an overview:

- **domain models:** domain analysis can be done by describing models at different meta-levels (typically instance and type levels) in a particular domain
- **domain interoperability:** domain analysis can be done describing the core domain, and ancillary domains along with their interoperability
- **component reuse and extensibility:** domain analysis can be done describing which components can be reused and extended within a realization domain
- **media-kinds:** domain analysis can be done using media-kinds
- **content and meta-data:** domain analysis can be done separating content and meta-data and describing how meta-data is used to manage content
- **information management functionality:** domain analysis can be done describing how multimedia information management is realized in a domain
- **stakeholder's role:** domain analysis can be done describing the stakeholder's role in a domain

The viewpoints described above are weighted differently in the analysis, the weights may even differ throughout domains. The detailed reasons can be found in the conclusion chapter 3.4. After giving a short overview of the scope of the domain, the analysis is described in the following order:

The primary classifier in domain analysis is the role of stakeholders. First, we elaborate on the conceptual designer, since he lays down the basic concepts for the whole domain. Here, we only focus on the concepts relevant to the domain. The main information management tasks are also explained, as far as they are part of the concepts. If the particular domain can be described using the «content vs. meta-data» view, it is also mentioned in the required depth.

The user's role focuses on how the domain concepts appear to the user, and how the user utilizes the domain for her own purposes. It is however beyond the goals of this analysis to analyse users in a certain application domain, but it is within the scope how the user interacts with the concepts.

As the next viewpoint, we examine the system architect's role. The system architect focuses on how the concepts are realized. This requires that the core domain described by the conceptual designer is extended by ancillary domains that are necessary for the system architect to realize the concepts in their entirety. Besides the description of the ancillary domains, their interoperability is also explained.
Finally, there is a last part describing the extending issues concerning the alleviation of the deficiencies found in domains and desired directions in which the domain can be extended concerning its stakeholder’s viewpoints.

The domains we tackle in the remainder of this chapter are: (1) content processing and compositing (subsection 3.3.4), (2) content-based indexing and retrieval (subsection 3.3.5), (3) document management and workflow (subsection 3.3.6), (4) continuous media storage and streaming (subsection 3.3.7), (5) medialization (subsection 3.3.8), (6) user interfaces (subsection 3.3.9), (7) multimedia publishing and presentation (subsection 3.3.10).

3.3.4 Content Processing and Compositing

Content processing and compositing is centered around the manipulation of content [15]. The overall goal of content manipulation is either to modify existing content, or to create new content that is a result of processing or compositing existing content. If the term «editing» is used in the same context, it denotes interactive processing and compositing. Content processing and compositing is defined on non-synthetic perception media, that is, image, video and sound. Manipulation can also involve representations of synthetic perception media, such as animation, graphics, text and speech, however, these will be considered as one of image, video and sound, that is, non-synthetic. Obviously, if graphics and text are handled as image, and animation as video, speech and music as sound, their peculiarities cannot be taken advantage of.

3.3.4.1 The Conceptual Designer’s Role

The common concepts the conceptual model of this domain contains are the content and the operations that manipulate content. Content in this domain is referred to as information in the representation medium. The notion of meta-data is not present in this domain. Content is subdivided into channels (tracks), which can be separately manipulated by operations. Operations can either modify content or create new content by compositing existing content. Further concepts are specific to perception media, they are elaborated separately below. Operations can be concatenated to a workflow, which determines their sequence of execution and provides a more complex construct for reuse.

There exists another general and relevant processing task: compression [50]. Content compression saves scarce resources (storage capacity and transmission bandwidth) by representing content more efficiently. Compression can be characterized by its ratio and quality. We distinguish lossy and lossless compression kinds. Lossless compression usually does not achieve a high compression ratio in compressing content but has perfect quality. Lossy compression is perception aligned: it contains substantially less information in the representation medium, but preserves almost all information in the perception medium. There is a tradeoff between its ratio and its quality; this can be adjusted.

In case of images there as few as one channel for monochrome or grayscale images, are as many as three color channels and one alpha (or matte) channel for full-color transparent images. Within a channel, there are pixels in a given twodimensional (2D) spatial resolution and color depth. Color values of pixels always relate to a color space, typically RGB, YUV or HSV. The smallest unit of manipulation is a pixel or a pixel component (part of the pixel that belongs to one channel). Several pixels together define a region.

Manipulations of image content fall into two categories: basic manipulation and compositing. Basic manipulations are color-based (addition, gamma correction, inversion, contrast, CLUT, HSV etc.), spatial (convolution, blur, sharpen, median etc.) or geometric (pan, crop, rotate, scale, flip, warp etc.). Composition is a multi-source operation. The sources can be added, mixed, over-ed, multiplied, subtracted, under-ed, xor-ed, min-ed, max-ed, screened etc. The alpha channel (matte) is the key mechanism to most composition kinds, it defines which region shall be visible of which source of the composite to which extent. The creation of the alpha channel is called keying, which can be based on luminance (Y of YUV), chrominance (UV of YUV) values and other, sophisticated mechanisms (difference, edge matte etc.).
3.3. MULTIMEDIA INFORMATION MANAGEMENT DOMAINS

Image compression is both used in lossless and lossy form. Lossy compression is preferred for its efficiency. Quality degradation can be recognized in the decreased number of colors and the lowered contrast of images. Lossless compression can be efficiently used for a images that use a low number of colors, and contain large contiguous regions having the same color.

Video, from manipulation point of view, can be considered as a sequence of images. An image in a video is called a frame. Therefore video manipulations comprise all the image manipulations on individual frames, and add several video-specific ones: temporal manipulations. A sequence of video as a unit is usually called a clip, which is associated with a temporal resolution in frames per second (fps), which determines the length of the clip. Basic manipulations are insertion, removal, cutting and timing (resampling). Video compositing is based on an additional dimension: time. Temporal compositing is based on frames. More complex video manipulations are called digital video effects (DVEs). A special type of DVE is is transition that change the output from one source to another gradually.

An important video processing task is the compression of the video. Single frames of video can be compressed as images, however, the temporality of video can also be taken advantage of for compression. This means the representation of differences (deltas) between frames instead of compressing the whole frame. Since a new feature can also be compressed, the achieved compression ratio is higher than in case of images. Besides image artifacts, quality degradation can be recognized in increased motion blur. Difference compression is not optimally suitable for editing, for this purpose single frame compression is used.

In case of audio, there are can exist an arbitrary number of channels, starting from one (for mono) to 16/32/48 (studio recording). A channel contains samples\(^6\). Several consecutive samples together can form a region. The concept of frame is also present in audio, however, in a different meaning: an audio frame is a region of a given duration. General manipulations take samples or frames as the basic unit. Manipulations of audio include cut, paste, fade, ADSR, gain, echo, reverb, chorus, dynamics expansion/compression, vibrato, pitch bending/shifting, resampling etc. Audio compositing involves multiple sources from different channels. Typical compositing operations include merging, cross-fade etc.

Audio compression also accounts for perceptual features. Lossy algorithms are more popular, because of the achievable compression ratio. There are different compression algorithms for audio in general, and for speech in particular. Speech compression algorithms account substantially decreasing the quality by limiting the bandwidth of speech without the decrease of comprehension. Audio compression in general also aims at preserving esthetic quality. The general compression is based on masking, that is, the suppressive effect of several audio signals close in frequency or time but different in loudness.

3.3.4.2 The User’s Role

The user selects content that is the input for processing and compositing. She can inspect it by viewing it or listening to it, extract information from it and apply certain operations on it. The result can be new content or a modified version of the source content. Viewing (listening) and information extraction are not inherent part of processing and compositing, they are meant for supplying the user with information to decide what operations to apply, and feedback on the result of the operations. From the user’s point of view, the most important feature of manipulation is interactivity: it determines how the user can utilize this domain to manipulate content.

Operations on the content can be applied by the user on-line and batched. On-line application has the advantage that the user can evaluate the results after each step, and proceed accordingly. Batch operations can be applied on a series of contents without changing. By executing the operation, a result content is created that can immediately be applied as the source of another operation. In case of batched operations, the user must create a workflow that defines a sequence of operations. The workflow has to be assembled by the user only once, and can be used for different contents.

\(^6\)Sometimes a sequence of samples is also called sample, but here it refers to the smallest element of audio content.
The user can interactively define several aspects of the processing and compositing process. These can be classified into four parts: content selection, operation selection and parameterizing, workflow definition and timeline definition.

Content selection is the first activity the user applies. Within a certain content, a specific portion can be identified by defining spatial (e.g. image) or temporal (e.g. sound) marks. If marks are set, the operation is applied only on the marked portion of the content. Operation parameterizing is the next fundamental setting: vast majority of the operations requires parameters that need to be set before execution. These parameters fall into three categories: scalar values that can be set through simple input fields or sliders, knobs (e.g. threshold), matrix values that can be set in a matrix editor (e.g. convolution kernel, CLUT), and a series values that can be set in a special editor, typically curve editor (e.g. gamma correction), equation editor (e.g. spline trajectory) etc. If the user decides to apply batched operations, she needs to define a workflow. The workflow is defined by selecting appropriate operations and chaining them visually, denoting the direction of flow between source and result.

Timeline definition is applied on time-aware perception media: audio and video. Timelining is a mapping of time and channels (also called tracks) onto a 2D surface. The user can manipulate tracks by inserting, overlaying, cutting, deleting portions into any channels. An important feature of timelines is the transition between tracks. Transitions can be abrupt and gradual. Abrupt transitions change content within a single frame (video) or sample (audio). If the perceptual context requires seamless changes from the content of one track to the content of another, gradual transitions are used. Another important feature of timelines is the synchronization of tracks, which is usually done by an independent timebase.

3.3.4.3 The System Architect’s Role

Figure 3.2 shows a typical domain architecture. The domain contains a manipulation engine that works on a local content workspace. The workspace is filled with content from content repositories, and the modified workspace is written back there. There also is a local undo buffer that makes undo information persistent, in order to cancel tentative changes. Content and configuration needs to be stored and made available for content processing. A configuration and a manipulation user interface communicate the domain to the user.

Content has three realizations in this domain environment: in the content repository, in the content workspace and in the user interface. Since content can only be manipulated in the workspace that resides in the main-memory of a workstation, content needs to be either copied intact into the workspace or accessed seamlessly. Copying is not suitable if the content to be processed has a vast size, likely for representations of most perception media. Most processing and compositing systems work on repositories based on file-system-based storage, therefore, seamless access can be provided by networking filesystems, such as Samba[65].
A tool is a realization of the concept of an operation (atomic or complex) tailored to work in a given workspace. The manipulation engine is a collection of tools (plug-ins, components) that manipulate the content. Prominent examples are the GIMP image manipulator [52], Adobe's Permiere [23]. Content that is stored in the content repository is usually present in different formats, often compressed. There are two approaches to cope with different formats.

The one approach is that there is a special tool that bidirectionally converts between a uniform format understood by other tools and a diversity of formats content can be represented in. This is called importing and exporting. The uniform format is a format that is easily alignable with the tools, the main property of such uniform formats is that there is a strong relationship between the information represented by the format and the information in the perception media (e.g. no compression).

The other approach is that there exist different versions of the same tool working on different formats of content. Although the latter approach avoids conversions, there are other considerations suggesting the first way: (1) the number of manipulating tools grows at a higher pace than the number of formats, therefore, it is more economical to write less versions of manipulating tools, (2) generally, writing manipulating tools requires more intellectual effort that writing converter tools, and (3) independent extensibility requires that if a new format or manipulation tool is created, no existing tools have to be changed.

If content has a vast size, it provides better scalability to convert to the desired format on-demand. This involves two resource-saving policies: size and computing power. The size of the workspace required is not more than necessary, and only the portions used are converted. The granularity of requested content is jointly defined by the format (e.g. I-I frames in MPEG) and the tool (internal granularity).

Workspace access can happen in two ways: the workspace is either shared between tools, or there are protected segments for exclusive access for tools. Both solutions can be justified in certain situations. If content access is exclusive for a tool, shared access is a good choice, because it avoids copying. However, if there is concurrent access, using separated segments in the workspace is a cleaner solution.

The undo buffer is a temporary content storage, which can be either in main memory (e.g. as part of the workspace), or external memory. The undo buffer has different behavior in case of unlimited undo levels, limited undo buffers and transactional undo. Most realizations have unlimited undo. Undo buffer management is minimal: memory images can be stored and restored.

Storage and graphical user interface components and subjects to reuse in this domain. Content storage in enterprise environments is usually file-system based, content is stored in file-formats and tools use a file access interface. Tool and workflow properties also need persistence for productivity and reuse independent of content. For this purpose, usually file-based serialization is used, domain specific video editing examples are edit decision lists (EDL) [41] and project files [22]. File-systems have a simple interface and are absolute commodities in any environment, however, their support for information management functionality is severely limited.

The reuse of graphical user interfaces in this domain is limited. Since most processing and compositing tasks carried out interactively by the user involve direct manipulation of content, graphical user interface primitives cannot be reused, therefore, each product suite provides its own graphical interface.

Extensibility in this domain is provided by tool plug-ins. The extensibility is independent, since complex workflows can be assembled using several plug-ins. However, there are neither protection boundaries nor bottleneck interfaces for plug-ins, therefore system stability and security cannot be established.

### 3.3.4 Extending Issues

Computing power severely limits the capabilities of this domain. To overcome this limitation, we distinguish between realtime and non-realtime content processing. In case of realtime content processing, temporal requirements have to be fulfilled. In enterprise environments, the computing power might not be sufficient to fulfill all the requirements, and due to the lack of special equipment, two solutions can be found. In one case,
non-realtime processing is done. This kind of processing distinguishes between a preview phase and a final phase. The preview phase works with diminished quality in order to fulfill the temporal requirements. If the preview results are satisfactory, we can enter the final processing phase, which runs in non-realtime mode. Another alleviation of the lack of computing power is the clustering of commodity computers in order to perform distributed processing. Both solutions would require this domain to be rearchitected.

The level of storage technology reuse in this domain is low. The advantages of reusing special DBMS components would be seamless undo management, versioning and integrity. The main reasons for scarce reuse being the nature of tools: content and undo information management is based on filesystems, they cannot be easily changed to use storage technology facilitating another interface. Even if special importer and exporter tools would be developed, transactional issues and undo integration would need severe modifications in the substrate component, rather than solely in the tools. Another shortcoming of the reuse of extended DBMS components for content storage is that these components are heavy in themselves containing functionality that is likely not needed for this domain (e.g. indexing).

Content storage could also be realized at a higher level, using the document management and workflow domain (chapter 3.3.6), it would provide workflow management, versioning and searching capabilities to this domain. However, the document management domain cannot provide realtime access to information in time-aware perception media. Moreover, vast amounts of information cannot be optimally stored on filesystems, for this purpose, the content storage and streaming domain could be reused. However, this domain only supports play-out sessions, rather than editing sessions, therefore no reuse can be utilized.

### 3.3.5 Content-based Indexing and Retrieval

Content based retrieval is information retrieval centered around perceptual features of information. It deals with content-based meta-data that can be automatically extracted from content. It is a major domain in multimedia information management, since all the major tasks of content-based retrieval are based on information in various perception media: image, video, text, audio and speech.

In this chapter we review the stakeholders’ roles as follows: users communicate with a content-based retrieval system through querying, conceptual modelers design models ranging from low-level indices to models expressing high-level semantic concepts in information, and system architects provide facilities to store the models, execute queries on them and present the results.

#### 3.3.5.1 The Conceptual Designer’s Role

The conceptual designer needs to put all the relevant domain concepts into a consistent model. The most important domain concepts in content-based indexing and retrieval are feature, feature-space, feature extraction, indexing, transformation, distance measure and query\(^7\) [29].

Feature is considered a property of content. Each property has a type and a value and it occupies a dimension in feature space. Each content can have several features. Thus, each content is associated with a feature vector in the feature-space. The dimensionality of the feature-space and that of the vector is equal to the number of features, except in cases, where the extreme number of features necessitates dimension reduction. The efficient modeling of the feature space requires flexible, dynamic index structures. Index structures for content-based retrieval are tree-based. To fill in the feature space with features, they need to be extracted from the content. This is done by an extracting algorithm that processes content to compute the feature value. This feature value is inserted to the tree representing the feature-space.

Explicit queries (the mechanism behind textual query languages) query the features directly since queries define a hyperregion in the feature-space which can be accessed by traversing the tree-based feature-space representation. Similarity-based queries (the mechanism behind example-based query) require more elaboration.

\(^7\)There are some concepts, such as precision and recall that are not mentioned because these have no relevance to the goals of the analysis.
They can be evaluated using either the \textit{metric} approach of the \textit{transformation} approach. The metric approach extracts all features from the example and it additionally defines distance measures for similarity in order to compare vectors in the feature-space. The transformation approach is based on the principle that the level of dissimilarity between two entities of content is proportional to the cost of transforming the one content to the other. As opposed to the previous approach, here, the transformation operations are fixed, rather than the similarity measure itself.

\textbf{Image} queries comprise \textit{spatial}, and \textit{perceptual} queries. Both queries utilize certain \textit{features} that can be queried for. Spatial image queries are either \textit{topological} (e.g. adjacency), or \textit{metric} (directions, distances, angles etc.). Perceptual image queries express \textit{n-dimensional points}, \textit{ranges} or \textit{nearest neighbors} in the \textit{feature-space}. The most common perceptual features are \textit{color}, \textit{texture} and \textit{shape}. Query languages and example-based querying can both be used for spatial querying and perceptual querying.

Space modeling for images includes popular tree-structures, such as the R-Tree, SS-Tree, Quadtree and special variations thereof [33]. Their common feature is that they are equally suitable for low-dimensional, that is, 2D and 3D, and high-dimensional feature space representation.

Content queries on \textit{audio} can be considered in two levels. There are low-level \textit{acoustic} features and other \textit{complex} features that derive from the low-level ones. Low-level features are extracted from \textit{frames} (see chapter 3.3.4) of audio-sequences, assuming quasi-static behavior within a frame. The mostly used low-level (frame-level) acoustic features are loudness, pitch, tone and cepstrum. High-level features refer to an audio sequence as a whole. The simplest high-level features provide the distribution of low-level features over the sequence (e.g. gaussian, histograms). Other high-level features are \textit{perceptual}, they focus on a certain perception medium. The most widespread specific audio perception media are \textit{music} and \textit{speech}. Music queries can involve abstract features, such as instruments, rhythm and events. Speech queries can both refer to the \textit{speaker} and the \textit{utterance}.

Since audio is time-aware content, feature values vary in time. There are two possibilities to account for this behavior: The first possibility is that the whole audio content (clip) or a perceptible region (in the second range) is treated as one unit, therefore all features refer to this unit as a whole, no refinement within a unit is possible. The other possibility is to consider frames as units for feature extraction, since they are assumed to have static behavior. In the first case, to extend content description in the temporal dimension, a mono-dimensional \textit{space} is used in form of segment-trees (RS-Tree). For the second case, feature-space modeling is identical with tree structures described for images.

\textbf{Video} content-based querying can fully re-use the concepts of image querying mechanisms: the individual frames of a video-sequence can be treated as standalone images. Additionally, there are several specific features for content-based video retrieval. The basic notion of video querying is based on the \textit{shot}. A shot contains a sequence of frames recorded contiguousously representing a continuous action in time or space. There are two other categories, the \textit{key frame} and the \textit{key object} that extend the notion of shots. Key frames are characteristic frames of a shot, key objects are easily perceptible contiguous regions in motion, throughout several frames in a shot. Features that originate from the shot are color distribution (mean and variance) over a shot, global motion feature (camera pan, tilt, zoom). Key object features are \textit{shape}, \textit{motion}, \textit{life-cycle}, \textit{color} and \textit{texture}. Queries, as in previous cases, can be ad-hoc and example-based. Ad-hoc queries retrieve shots, whose selected features satisfy the query constraints. Example queries are issued on shot similarity, based on key frame or key object features.

With respect to the temporal dimension, feature-space modeling is identical to that of audio content. What concerns \textit{static} features like that of key frames, image feature-space modeling applies.

Content-based \textbf{text} retrieval has the longest history among all perception media. Text retrieval is somewhat different from the perception media discussed above. As with all perception media, we can differentiate between ad-hoc and similarity queries. Both query kinds are based on a main feature called \textit{keyword}. Automatic keyword extraction is based on the following steps: \textit{elimination} of words not carrying any characteristic feature of the content (stop list), word stemming and creation of frequency tables. Since the number of key-
words is vast, dimension reduction techniques are used to lower the dimension to a manageable range. The representation of this reduced feature space happens using TV-Trees.

3.3.5.2 The User’s Role

The user's role in content based retrieval is centered around content-based querying. The user can issue queries to the system, and view the query results retrieved by the system. The query and the visualization interfaces are functionally separate, but in case they are based on the same perception medium they can appear jointly as one interface. In case they differ, they need to be separately presented to the user. A further connection between the two interfaces is that queries can be issued iteratively, using the result set of the previous query.

Query languages are textual and usually exploit SQL-like syntax (ad-hoc querying). They are popular, because of their simple interface and mature background. Their drawback for the user is that the user must have a priori knowledge about the features she want to query the system against, and the syntax for formulating the query. Query by example is more intuitive, therefore better suitable for unexperienced end-users. In this case, the user supplies the system with an example to which similar results are retrieved.

Query languages are typed by the user as text and, if the query is valid (according to the query syntax), is it processed by the system. Example-based query interfaces are more elaborate. We can differentiate several different kinds of example-based query-types. Iconic querying provides a predefined set of icons, typically representing spatial or temporal constraints that can be concatenated together to form a query. Querying by painting and sketching are used for image retrieval and encourage the user to provide examples for colour-based and shape based retrieval, respectively. Querying by texture and images are based on the user's selection of examples in an existing content-space.

For querying audio, more specifically music and speech, the latter examples turn into audio clips against which the system can be queried. In audio queries the modality of the query is either a query language similar to image queries, which queries for features, or an example-based query, that can refer to a query of any abstraction level, including as simple features as frequencies and as abstract as speaker similarity. This means that the user has to supply an audio example with the name or symbol of the intentional feature the system is queried for.

Textual query interfaces let the user enter keywords and keyphrases and search for text documents possessing or missing these words. The example queries are based on similarity, the user selects a reference content from a repository.

3.3.5.3 The System Architect’s Role

Figure 3.3 depicts a typical architecture for a content-based retrieval system. The figure contains two ancillary domains (e.g. GUI and document management domains) that together with the content-based retrieval domain constitute a content-based retrieval architecture. The core domain consists of three elements: the index storage, the search engine and the feature extraction engine. The feature extraction engine accesses the content storage for content, extracts the required features from it, and stores the result in a feature space in form of multi-dimensional index-trees. The index storage provides specialized storage capabilities for indices. The search engine processes the user query coming from the query GUI, searches the index structures, and passes references to satisfying content to the visualization GUI. The visualization GUI accesses the content databases and retrieves content that corresponds to the references. The user, based on the relevance of the results, is either content with the results or refines the query to increase precision. Based on any selection, the visualization GUI can be extended to facilitate issuing query-by-example.

Content storage is not of primary importance to this domain, the only requirement is that content be uniquely referenceable and accessible. The user interface is only required to support the presentation medium, that is, inputting of queries, and viewing results.
3.3. MULTIMEDIA INFORMATION MANAGEMENT DOMAINS

The storage and access of feature-space representations are the focuses of this domain. Two approaches to achieve this are the use of main-memory and external-memory structures. Main-memory structures only work at predictable index sizes, and additional serialization needs to be applied. This solution is not flexible, external-memory structures provide more room for configurability. External memory structures need to be aligned with the underlying storage architecture (e.g. pagesize), and implemented on either file systems or on core storage devices, which puts an extra demand on implementation. The use of DBMSs (Database Management Systems) for managing indices frees the architect from dealing with these structures explicitly. We distinguish relational database systems (RDBMS), extended RDBMS (e.g. ORDBMS), object database systems (ODBMS) and multimedia database systems (MMDBMS).

RDBMSs cannot use arbitrary indexing methods, their indexing architecture is based on B-Trees. Since B-Trees cannot represent multi-dimensional spaces, content features must be represented as a table row with content reference and explicit feature values. This implies that indexing must be done for each feature separately, which, in case of many features makes the index manipulation overhead huge.

Extended RDBMS technology provides components to extend DBMS capabilities, e.g. Informix Datablades [46]. Components can introduce new data-types and new index structures. With these approaches, there are two shortcomings however. On the one hand, despite the use of extension components, the substrate component still remains heavy, there is no way to scale it down. On the other hand, for optimal indexing, opaque data-types have to be used, whose structure the DBMS has no knowledge of, therefore existing DBMS functionality is neglected: not a nice perspective for reuse.

ODBMSs also use B-Trees as the general indexing method, however object structures themselves can be used to represent the required multidimensional tree-structures. This solution is far from optimal, since general objects cannot be optimized for the needs of indices, and DBMS indexing is not reused.

3.3.5.4 Extending Issues

There are three extra-domain issues concerning the user's role in content-based retrieval, which require the extension of this technology in order to exploit its full capabilities. The first issue is the interactivity context for querying. Since similarity queries are not precise, they might require feedback for further search. This feedback shall originate from users, who restrict further search, change the search constraints according to their personal profile (e.g. different sensitivity for a perceptual attribute), according to context profiles etc. These requirements extend the retrieval process from a one-step process to an interactive session that requires dynamic, run-time combination of reconfiguration, session state management, querying and browsing.

The second issue is the involvement of several perceptual media in the querying and retrieval process. Apparent from the above review, the typical querying scenario is to query either using a query language or
an example of the same perception media as the result. However, cross-modal querying could lead to a better utilization of the retrieval process (e.g. [61]), since different modalities may express different features in a more straightforward way (e.g. colorbar for picking color features, comparing speech content with textual transcripts).

The third issue is the visualization of feature-spaces. Since dimensions that are greater than three cannot be easily visualized intuitively, a 2D or 3D subspace is used with the two or three most relevant features. Visualization is handled as a part of chapter 3.3.8.

Concerning the conceptual designer’s role, the major direction is to extend feature-space models with more complex models upholding knowledge-structures. There have been proposals to extend feature-space models with more intelligence (e.g. active indices [17]). This requires merging index structures with higher level semantic structures: a challenge that only optimally interoperationg domains can solve.

### 3.3.6 Document Management and Workflow

Document management [74] is a broad domain, it deals with the life-cycle management of documents. Document management can be well described by the content/meta-data view. As opposed to the content-based retrieval domain, this domain is centered around the content-independent meta-data.

The information documents contain can be characterized by the perception medium. Traditional document managements were based on document imaging, which meant scanning documents and storing them as representation of information in the image perception medium with meta-data. Queries on the meta-data gave pointers to the scanned document that could be accessed.

Nowadays, document management is based on electronic documents that are represented as they best fit the intentional perception medium: there has been a separation of images and text, such that document imaging holds only information in the image perception medium, textual information is handled separately.

#### 3.3.6.1 The Conceptual Designer’s Role

Document management is an abstraction on the storage, transmission and access of documents. The coordination of these abstractions is done by the notion of workflow: the workflow defines an order in which certain tasks have to be applied on the documents. Document management comprises the following main tasks: creation, editing, annotation, classification and versioning.

The document workflow starts with the creation of the document. The creator either captures multimedia documents by digitizing natural artifacts through an input presentation medium into a representation medium (e.g. photograph), or acquires the document from a separate system in a given representation medium (e.g. import from an application), or creates the document by himself. The information in the representation medium is extended by meta-data.

When a document is created, it can proceed to the subsequent tasks. Tasks fall into workflow-aware and workflow-unaware tasks. Editing is the main workflow-aware task. According to the workflow definition, these can be sequential and parallel tasks. When workflow tasks are sequential, the accomplishment of a certain task triggers the next task. In case tasks are parallel, tasks result in different versions of the same document.

Besides editing, users can search for documents, view, review and annotate documents, create workflows for documents. These are workflow-unaware tasks. Both kinds of tasks require access control. Access control describes who has viewing, annotation, reviewing, editing, searching and workflow rights on documents.

Meta-data covers two notions in the document management and workflow domain: management meta-data (content-independent meta-data) and document annotations (general meta-data). Management meta-data concerns the document information, audit information, accessibility, versioning and classification. Document
information holds information about the title of the document, the creator, creation date etc. The audit information contains the list of editors, editing dates, changes etc. Accessability information comprises the access control list for different tasks. Classification groups documents according to the document's genre, keywords etc. Versioning information contains the versions of documents, either by symbols, or also in a tree-like fashion. Annotation is meta-data produced by the users: they are able to insert marginal notes and highlight regions when reviewing documents.

3.3.6.2 The User's Role

Users can apply workflow-aware and workflow-unaware tasks on documents. The folder metaphor is of central importance to the user in both cases. In the workflow-unaware case, folders are hierarchic containing documents according to a classification. Users can browse, search in the folders, view their document contents. In case of workflow-aware tasks, users trigger the workflow by accomplishing a certain task: documents are taken from an incoming folder (check out), a task is carried out on the document by the user, then it is placed into the outgoing folder (check in). The abstraction of transmission and storage and causes folders to be seen as if they were automatically connected to each other according to the defined workflow.

Besides the common viewing, editing and annotating tasks, there also exists a management-console for a privileged user group, where auditing, workflow assembly and access control can be maintained. Another user group, document operators have the special task to create documents.

3.3.6.3 The System Architect's Role

The system architect faces many issues when realizing a document management architecture. The central component of the domain is a document repository that provides consistent persistence for documents and their meta-data. Document editors check out documents, apply a certain task and check documents in. This atomic process is continued along the workflow. Figure 3.4 depicts a generic architecture of the domain.

The separation of content storage and meta-data storage is not a necessity, the document server has the responsibility to provide a consistent view on the document. The traditional approach is to store content on the file-system and RDBMS for meta-data. Extended RDBMS technology merges the two notions: a typical example is the Oracle 8i RDBMS [1].
Lately, ODBMSs are payed more attention, since the management of structured documents is emerging. Structured documents allow for a fined-grain management of content. This contributes to better reuse of document parts, and can enforce consistence in the workflow by uniform document structures. The stored and exchanged in XML [21]. XML is a generic language, a concrete document grammar is defined in the DTD (document type definition). Examples of ODBMS based XML document management backend components are POET [64] and Object Store [59].

User interface is the other component that is reusable in this domain. Since most document management products deal with textual documents, user interface components are mostly standard-based. Exceptions are the capture (scanner) and the edit interface.

Extensibility concerns document formats and document server functionality. In case of non-uniform formats, document servers cannot be extended to manage new formats. In case of generic, structured documents, such as XML, document meta-format can be made uniform, still being able to express different format instances. Document server functionality is nowadays becoming an issue, there are document servers that provide plug-ins for document editors, an example is author plug-in from the POET Content Management Suite [44].

3.3.6.4 Extending Issues

Using current architectures, the document management and workflow domain cannot be extended to support documents in a variety of perception media. The reasons are multifaceted. Document management and workflow abstracts seamless transfer of documents among clients. In case of documents in a variety of perception media, due to the time-awareness and the vast size, seamless transfer cannot be solved straightforwardly, streaming mechanisms need to be applied. Therefore, interoperability between this and the content streaming domain should be realized.

Another issue is the extension of search capabilities. The core functionality of this domain includes content descriptive meta-data based search. Textual document management often includes full-text search. The inclusion of documents in other perception media would necessitate the reuse of content-based retrieval domain.

What concerns document granularity, it seems that traditional document architectures cannot cope with the demand for structured documents. On the other hand, structured document management cannot seamlessly include information in other than text perception medium. A generalization of document management therefore would be required.

3.3.7 Continuous Media Storage and Streaming

Continuous media storage and streaming is a domain with relatively firm boundaries. The goal of multimedia streaming is to transport content from one location, from the stream supplier, to another, to the stream receiver, accounting for the peculiarities of the content being transported. The goal of multimedia storage is to provide a special stream supplier based on recorded content, and be a special stream receiver that records content. The peculiarities of storage and streaming are usually centered around time-aware behavior. The generalized notion of multimedia storage and streaming is capable of storing and streaming time-aware as well as time-unaware content. Multimedia storage and streaming deals with streaming content, there is no notion of metadata in this domain.

3.3.7.1 The Conceptual Designer's Role

The conceptual designer needs to consistently describe the major concepts of streaming. The most relevant concepts of this domain are devices, ports/endpoints, connections, content, streams, formats and quality of service (QoS). A stream can be considered a graph, with devices as nodes and streams as arches. We can distinguish between three kinds of device: source, filter and sink. Source devices supply the stream with content, either from live-source of recorded. Filters modify content while streams flow through them. Sinks
3.3. MULTIMEDIA INFORMATION MANAGEMENT DOMAINS

consume streams for recording or user perception. The graph of streaming is directed, and it can have multiple sources and multiple sinks. This involves the presence of special nodes with more than one inbound and/or more than one outbound streams. Devices possess ports (endpoints) with formats assigned. A device can receive and send content through these ports. Formats realize a contract between devices and streams: format assures that the inbound stream matches the expectations of the device.

A stream is characterized by the quality of service. The three most basic characteristics of the stream are the latency, the delay jitter and the throughput. Latency is the time difference between sending and receiving a stream. Delay jitter is the deviation from the average latency, and throughput is the amount of content that is streamed during a time unit. These characteristics can be defined on the longest path in the graph by using end-to-end characteristics, and also on the shortest path, that is, between two neighboring devices.

The content that is to be streamed varies. There are three important kinds of streamed content: audiovisual streams, audio streams, and textual streams. All three streams can originate from a live or recorded source.

3.3.7.2 The User's Role

The user can take two roles on streaming. In case of the first user role, the user is at the endpoint of a stream, where the content is presented in a presentation medium. The user either perceives content that comes from a repository or live-source by playing it, or records content that comes from a live-source. What the user sees in this case is a presentation context, usually called the movie/player metaphor. In case of the second role, the user assembles the stream, which means that she [or plumbs] together endpoints through which the content will stream. This abstraction is usually referred to as the circuit metaphor. In the following, we handle these two roles separately.

From the consumer user's point of view, streaming can mean asynchronous and isochronous behavior. The asynchronous behavior means that it takes a certain time to transport the whole content after it has been requested, but the user can receive uncontrolled partial content before the transport of the whole content finishes. The typical environments featuring this kind of behavior are HTML browsers when receiving time-unaware text and images. The isochronous behavior is typical for time-aware content, that is, content that can be perceived as audio, video, speech and animation. In this case, the user sees a continuous flow of content, not being able to isolate «snapshots» in the flow (e.g. individual frames in video). Whereas the asynchronous streaming behavior is a necessity caused by technology limitations, isochronous behavior is an inherent perceptual feature of time-aware content that the user wishes to control.

The consumer user's role in streaming are centered around two tasks: stream requesting and controlling. Requesting streaming happens through selecting a name or symbol from a naming service. The name is associated with content that can be streamed. After the content has been located (goes in a 'ready' state), it behaves according to stream controls. User level stream controlling combines playing, stopping, positioning, fast forwarding and rewinding the stream. Additional tasks may comprise perceptual quality settings (e.g. color, resolution in video) that impact streaming.

The circuit user's role are centered around creating streams from building blocks. These building blocks can vary, typical building blocks are nodes and arches, which define the path of the stream. A more precise description is given in the conceptual designer's chapter. The circuit user creates streams by assembling the blocks, accounting for several criteria that are built-in constraints of the visual assembly tool. Some of the constraints include the number of inbound and outbound connections on nodes, the directionality of the arches, plug-formats of nodes etc. In some cases the visual tool can express constraints by being «prewired», that is, it already contains all the nodes and map or net of all the possible connections among nodes. Here, the circuit user is only able to select the nodes and arches that need to be active for a certain configuration, without creating new ones.
3.3.7.3 The System Architect's Role

The architectural designer can tackle the streaming architecture at many abstraction levels. The largest grain approach is to view the architecture as a client and a server component connected through a network. From networking point of view, streaming is a client/server architecture, where the streaming server device (e.g. video server) supplies, and the streaming client device (e.g. movie player) consumes the stream. What is required at this architectural level is a server, a client, and a streaming protocol to stream content (down-stream) and a communication protocol (up-stream or bidirectional) through which the client can initiate streaming and communicate with the server. There are three scenarios at this level.

The first scenario is about play-out sessions: the server is connected to a number of clients that simultaneously request streams. The server has a large storage capacity, high-performance I/O bandwidth and a fast network subsystem [54]. Storage is hierarchic [9], storage layers include main memory (RAM), fast online devices (harddisk, RAID), slow online devices (optical devices, jukeboxes), offline devices (tapes). The presented order shows increasing storage capacity, access time, and decreasing performance, probability of access and costs. High-performance I/O bandwidth addresses the first two storage layers.

The client receives the stream through the network for the purpose of viewing it. The viewer is embedded into a GUI environment, the player has rudimentary control over streaming. In order to eliminate deviations from an optimal isochronous streaming behavior, the client uses a jitter buffer. Due to technology and real-time considerations, the size of this buffer is limited. The received stream is usually compressed, it needs to be decompressed for consumption.

The second scenario is about a recording session: a live streaming source feeds a live stream into the sink in order to store it. To enable this scenario, the storage server should act as a sink and support a recording or maintenance session. The third scenario is a live session, without storing information.

A finer abstraction level has to be considered if the notion of stream with two endpoints does not suffice. This can be the case for performance reasons (e.g. internal components of client and server need to be modified or restructured), or if the stream needs to be manipulated (e.g. noise filtering, decompression), several streams need to be joined (e.g. dynamic subtitles), or a stream needs to be split (e.g. separating audio and video in a compound stream). The finer view distinguishes intra-process, inter-process and inter-port streaming between devices within the server as well as the client. The first two streaming kinds apply within a host, the third kind between networked hosts. These streaming issues are operating system dependent, common mechanisms are memcopy (intra-process), shared memory and mmap (interprocess).

Another challenge is raised by inter-port streaming, which is meant to provide streaming between devices on physically isolated hosts. A typical networking infrastructure that connects the hosts is the internet. The uniqueness of inter-port streaming is that needs a wire-protocol. The wire-protocol of the internet is the TCP/IP network suite. Common protocols include TCP and UDP, but lately RTP has also been added to the selection due to the need for real-time streaming (however this remains a best effort solution). Figure3.5 gives an overview of a common architecture. The figure does not show the storage hierarchy in the server, and processing streams on the client.

The main requirement of the mapping is that the user of the model (typically the circuit user) shall not see implementation differences as to different wire-protocols, memory and process management mechanisms etc. The mapping of the models to the above architectures in a transparent manner poses two main issues. The one issue is the heterogeneity of formats, the other issue is that devices work asynchronously with respect to each other.

In order to plug devices together for streaming, they need to be plug-compatible. Streaming architectures can be multileveled (e.g. ATM-on-IP-on-UDP-on-RTP) including hardware as well as software parts, devices might require compatibility at any level, depending on which architectural level they themselves are. The system architect has to assure that the devices that are provided as reifications for the model expose their formats and communicate it to the model. In this case, the model can realize formal matching using other architectural mechanisms (e.g. through substitutability as a main object-oriented principle).
Plug-compatibility assures that devices can be connected in physical sense, and they stay consistent with the model, through which the user accesses them. It can be recognized that plug compatibility alone cannot fully determine stream behavior, since the stream is a temporally dynamic notion. In general, devices that create, process, forward and swallow streams work *asynchronously* at their own pace. Even if streaming can be perfectly isochronous between two certain devices, the heterogeneity of the entire environment demands that extra care is taken in order to provide isochronous behavior on a global scale. There are two mechanisms that to support this: synchronization and buffering.

Concrete realizations of streaming architectures reflect the two level architecture. Typical examples of the client/server level include Oracle's Interactive TV [26], RealNetworks's RealPlayer [45], Apple's QuickTime [10] etc. These architectures provide a whole suite including streaming format with compression algorithm, streaming server and streaming client. A finer grain architecture is the Java Media Framework [73], including several streaming devices for manipulating the stream, however, this architecture excludes the server architecture and is limited to the Java programming language and execution environment. A CORBA-based architecture is the CORBA A/V Streaming Service [4], which provides a standardized control interface to streaming, however it is meant for the management of end-to-end connections. Storage and other than end-to-end streams are not supported. Another approach is provided by PREMO [5], which contains both the distributive nature of the CORBA A/V service and the elaborate device configurability of JMF. PREMO was initially a language-neutral architecture, implementational issues forced the PREMO developers to tighten the scope to a homogeneous solution, which was Java.

Reuse in the high-level architecture is restricted to reusing two components: the server and the client. Since the protocol and the format define a server and a client pair, there is not much room for flexibility. The finer-grain architectures are pluggable, therefore compression, stream-transfer, players and other components can be independently developed. However, the most complete and open architecture, JMF only exist as Java and only at the client side, and PREMO does not have any implementation.

### 3.3.7.4 Extending Issues

There are two issues that lead the need to improve this domain. The one issue tackles the client server architecture. At the one end, current storage and streaming servers have been designed for a vast number of concurrent streams. At the other end, traditional filesystems cannot deliver optimal performance, due to blocksize limitations and hardwired block-placement algorithms. The high-end solution cannot scale down, whereas the low-end solution cannot scale up. A solution that scales from few streams (1-5) to many streams (100 or more) optimally is yet to be developed.

Streaming is a severe issue in current mainstream operating systems. There are two concurring views: On the one hand, devices should be in one protection area (process) in order to eliminate communication overhead,
and execute sequentially to eliminate possible race conditions. On the other hand, in order to be extensible and scalable, devices should be decoupled from each other implying concurrent execution. Concurrent execution, however, requires more effort in inter-device communication, because of synchronization and crossing protection boundaries. An optimal architecture and communication strategy can only be achieved if devices can be flexibly reconfigured for different architectures. This is not yet solved with currently available realizations.

### 3.3.8 Medialization

Medialization is a term created by the author of this work to denote an «artificial» domain for grouping «historically unrelated» fields with respect to their similar relationship to multimedia. Medialization means the mapping of source information in any representation, regardless of being perceptible or not, to target information in synthetic perception media.

#### 3.3.8.1 The Conceptual Designer's Role

Medialization in general has three distinct parts, which together comprise the creation of information in synthetic perception medium. The first part is defined by the availability of the source information. The second part is the mapping of this information into information of a target model. The third part is the provision of an instance to be perceived. The first part is acquisition, the second part is either modeling or recognition. The third part is rendering.

These parts typically do not exist in isolation, but combined into different subdomains of medialization. Acquisition is a separate process, it is just included for completeness. There are three typical subdomains. The names of the subdomains are either one of the parts, or a new name for the combination, with the result perception medium kind prepended.

In 3D graphical modeling, the goal is the achieve similarity to real-world artifacts. 3D modeling refers to graphics (spatial model) and animation (spatial+temporal model). 2D graphics and animation can be considered as a special case of 3D. The process of 3D modeling creates an instance of a predefined 3D model that can be rendered into information in some synthetic perception medium. There exist several 3D models, nowadays, object oriented, scene-graph based modeling is considered the most straightforward method, e.g. [51]. The scenegraph is a hierarchy with the scene at the top and several nodes placed in a coordinate system defined by the scene. Nodes can contain the following elements: (1) geometry primitives, such as meshes, spheres, cubes, lines etc., (2) structural modifiers, such as translation, scaling, rotation, and (3) visual modifiers, such as light, shading, texture and material. The graph is based on the aggregation (composition) of different elements. The scenegraph expresses static spatial scene, temporal elements in the scenegraph can express temporal behavior. Rendering is based on the descriptions of the scenegraph. Common rendering methods are raytracing, Phong and Gouraud shading.

*Sound synthesis* is based on the usage of acoustic primitives and modifiers for modeling and rendering sound. Sound synthesis is broken down to music and speech synthesis. Speech synthesis is based on either formant synthesis or linear predictive coding, music synthesis is based on a musical score and instruments. There is a parallel between 3D modeling and sound synthesis: musical instruments are like graphical primitives, whereas musical score expresses temporal behavior.

*Visualization* [37] starts with the acquisition of a *data-set* (instance) in a rudimentary model is obtained in a representation medium. In order to medialize the data-set, a simple or complex *mapping* scheme is applied to produce an *instance* of a *target model*. This mapping depends on the acquisition model as well as on the target model. Typically, the input data-set is in a spatial, temporal, or spatiotemporal model. 2D modeling primitives are points, lines, curves, polygons (triangles, rectangles), *circles* and *arcs*. Common 3D modeling primitives are *polygon meshes*, voxels. In the 2D case, rendering features are color, fill, stroke. With these primitives

---

8This term has been invented by the author of this work. The meaning is described below.
and features, we can create bargraphs, scatter plots and contour/trajectory graphs, isosurfaces. In the 3D case, we distinguish between surface and volume models. The target model is usually represented as a subset of the model used in 3D graphical modeling, therefore their rendering is the same. The differences are in the manipulation of the model, since in case of 3D modeling, visual modifiers are implicit in the scenegraph, however, in visualization they are considered as the main features of visualization, therefore they are explicit.

Visualization incorporates temporal behavior in two ways. If animation is rendered, temporal behavior is extracted from source information, it is therefore explicit. If video is rendered, independent visualization snapshots are recorded in a sequence. In the first case, rendering happens in real-time, therefore visualization features can be changed dynamically, in the latter case, the whole sequence has to be rerendered off-line.

Sonification is a relatively new area, its goals and directions have been summarized in [34]. Sonification turns information in any representation media to information in audible perception media. The target model of sonification is sound synthesis, similarly to the visualization/3D modeling pair.

In case of recognition, the source information is in a certain non-synthetic perception medium. The goal of recognition is to change the perception medium to a synthetic one, such that the information produced in the new perception medium is similar to the source information\(^9\). This case is akin to visualization, however, here, the source is also information in some perception medium, and since similarity cannot be an absolute measure, there exist domain-specific meta-data that helps make assumptions about the source information and therefore define some similarity measure. The assumptions are embodied in a media model and domain specific mapping rules. Recognition is typed according to the source perception medium, that is, speech recognition [68], image/pattern recognition or machine vision [57], optical character recognition (OCR) [40], motion recognition [58]. Sometimes, recognition is referred to as conversion, like text-to-speech conversion (TTS) [68].

3.3.8.2 The User's Role

The user can be involved in three types of major scenarios in this domain. This can be a viewer, inspector, conversational or editor role. The viewer role is the simplest, here the user passively perceives the rendered result. This might be the case when viewing relatively simple results, such as a barchart comparing quarterly revenues in a year in a business environment. The inspector role is an interactive role, the user can inspect the rendered information from different viewpoints, aspects, granularities etc. If used in visualization, features can also be dynamically changed. The inspector role is taken when the medialized result is rather complex either spatially (e.g. volume-rendered atmospheric phenomena) or temporally (e.g. sonogram of a talk). The viewer and the inspector roles are typical for visualization. The conversational role puts the medialized information into a communication flow either between two user's or the user and the software system. This is typical for recognition. The editor role is the most complex role, here, the user builds a scene by directly manipulating modeling primitives and composing the scene-graph.

3.3.8.3 The System Architect's Role

The architecture of the domain is centered around the medializer. It receives the data-set and some domain-specific meta-data in case the medialization process requires it. The architecture distinguishes between strongly coupled and loosely coupled modes of operation. If the user takes the viewer or inspector role through the corresponding UI, the medialized content can be viewer off-line from a repository. In case of a conversational role, the medialization engine generates the medialized content in real-time. The two modes of operation does not only depend on the user's role, but also on the performance of the medialization engine. If the medialization needs domain-dependent meta-data (e.g. speaker characteristics or language selection in speech recognition) this must be configured in order for proper operation. In case of editing, there is no data-set needed, the instance that has been created by the modeler can be saved into the content repository, or can be rendered. Figure 3.6 provides an overview.

\(^9\)This is actually a subset of recognition, the complete notion of recognition is not tackled here.
Reuse in this domain is usually scarce, because historically, these domains were maintained in isolation with the main reason that computing power mediation demanded was not available in commodity environments. Nowadays, lower-end mediation can even be accomplished in real-time in enterprise environments, too. However, due to the historic background, despite the similarities and overlaps, there exist modeling, visualizing and recognition environments separately.

The detailed description of these environments is beyond the scope of this work. What can be stated is that these environments show a scarce reuse of components, since the isolated environments do not use components. In the core domain mapper, modeler and renderer components could be reused. As to generic components, persistence could be reused to store scenegraphs. Current environments build their own DBMS functionality to uphold scenegraphs (e.g. [51]). Another, language-specific solution is the serialization of in-memory scenegraphs, this is taken by Java.

### 3.3.8.4 Extending Issues

Medialization today is a separate domain, and since this domain has been artificially created by the author, it can be further broken down to several domains. The reason why this domain has been created, is the common way how the concepts described above are handled.

However, it would be challenging to include medialization into the user interface domain in order to allow the user for seamless manipulation of the medialized content. The interoperability with document management would also the user to switch between different perception media for a certain document containing conceptually the same information, when both the source and the medialized information are in a perception medium. An important issue is the assurance of consistence, such that changing a document in a certain medialization would automatically involve the propagation of changes to other active medializations. A simple form of this notion can be observed in compound documents (e.g. OpenDoc [25]), where for instance, a table and a barchart representation of the same data-set update themselves when one of the representations changes. However, the supported perception media is usually limited, and the change propagation is not bidirectional.

### 3.3.9 User Interfaces

*User interfaces* are of crucial importance for all domains that communicate with users. The goal of user interfaces is to provide the user with perceptual abstractions on a particular system. Early user interfaces were
3.3. MULTIMEDIA INFORMATION MANAGEMENT DOMAINS

textual, but soon, graphical user interfaces (GUIs) become widespread, because of their intuitiveness and expressive power [56]. Since recently, other modalities have also been used to exploit the user's perception and interaction capabilities. The quality of the user interface is impacted by its input and output modalities. In a mainstream enterprise environment only a restricted set of input/output modalities are available, hence this chapter only focuses on these.

3.3.9.1 The Conceptual Designer's Role

The main concepts of the user interface management domain are centered around the provision of metaphors. A metaphor is based suggesting similarity to a real world-object. Metaphors are used to provide information to the user in certain perception media, and let the user view, control, select and manipulate this information directly or indirectly.

Common metaphors can be collected in groups. Command metaphors include command languages, human activity based metaphors are agents, conferencing, theater, container metaphors are folders, libraries, 2D spatial metaphors are forms, spreadsheets, tables, pages, menus, maps, frames, windows, blackboards, 2.5D spatial metaphors are desktops, books, notebooks, card files, machine/tool metaphors are instrument/control panels, pens, typewriters, clipboards, television, players, warming lights, buttons, telephone, postal systems etc. From this listing, we can see that several modalities have to be exploited, often simultaneously in order to construct these metaphors and make them work. In the following, we examine which modalities are available in enterprise environments, and they can be exploited to create metaphors.

In a mainstream enterprise environment, we can distinguish between textual input and output, graphical (2D/3D) input and output, sound (speech music, generic) input and output, 2D spatial input, keyboard input, video input and output. These perception media are bound to input and output presentation media, that is, modalities through virtual devices that abstract hardware I/O devices. Although these perception media and their direction (input/output) can be mixed arbitrarily, traditionally, different user interface kinds have been created, which group perception media and directionality (input/output) together. We first base our investigation on graphical user interfaces (GUIs) as the commodity user interface kind. Traditional GUIs comprise keyboard and mouse (2D spatial input), and 2D graphical output that comprises text output.

To achieve reuse, traditional GUIs provide widgets as simple metaphors for building complex metaphors. These widgets define the information the metaphor can receive, which is called event. Widgets can be sent events through the mouse (mouse events) and the keyboard (keyboard events). Events are generated upon actions of the user. Widgets also define how the metaphor reacts to events in terms of graphics. The system of user actions and widget reactions together are called interaction. Common widgets are window, scroll bar, label, form, list, push button, toggle switch, radio box, dialogs, textpane, menubar, pull-down menus, pop-up menus, and icons. There can be several widgets at the user's disposal, the focus defines, which widget receives the events. An important issue is the consistence of widgets. For this purpose, widgets are grouped and called the widget set. Their consistence is often referred to as the look and feel (L&F) of the widget set.

The widget-based concept described above is not sufficient for describing complex GUIs. The reason being is that with widgets becoming more complex, they themselves are needed to be modeled and be tailorable (e.g. widget style). In order to achieve a more structured and flexible handling of user interfaces, two further concepts are have been necessary: widget hierarchies and the Model-View-Controller (MVC) pattern [30].

Hierarchies allow for widgets to be encapsulated into each other. A compound widget appearing the same as a simple widget at any level in the hierarchy. This is a powerful concept used to flexibly and dynamically layout widgets spatially, and dispatch events to them. With hierarchies, complex metaphors, such as the desktop metaphor can be modeled hierarchically down to the smallest metaphor, such as buttons, taking care of consistence.

The Model-View-Controller (MVC) pattern is the other concept used for complex widgets. The utilization of this object-oriented pattern has the advantage that not only hierarchies, but also collaboration schemes within a widget are revealed and can be controlled. The model is the source of information that needs to be
communicated to the user, the view presents this information in a certain perception medium, the controller receives events resulting from the user’s actions. The collaboration between the three concepts defined by the pattern is that the controller receives an event, it changes the model or instructs it to be changed, and instructs the view to update itself since the model has changed.

The creation of complex views, models and controllers reveals new conceptual issues. The one issue is that with views becoming complex (e.g. tree-view, table-view), they tend to have their own models generically describing concepts that fit into the view. This means an MV pattern in the V of MVC, such that the concept of MVC needs to be made hierarchic (which is a different hierarchy from the one presented above). Another issue is that the view and the controller (VC) of MVC is usually merged because of writing general controllers not knowing about the views proved rather difficult (e.g. Java’s Swing, Microsoft’s MFC). Yet another issue is that with models becoming complex, views must adapt to this complexity in both presentation and manipulation. With the heavy use of information in image, 2D/3D graphics and structured text, views tend to move from a dialog-oriented manipulation to direct manipulation of information. Direct manipulation is more intuitive and in case of complex models the only way to manage information. Direct manipulation defines handles, rather than separate widgets to manipulate information in a view.

Reusable GUI elements cannot provide the user with all the metaphors listed earlier, since they lack the expressive power and modalities for that. If we generalize the notion of UI, the multimodal UI can be considered the general UI, with GUIs and textual UIs as special restrictions on input and output modalities. A multimodal UI contains all the GUI concepts and several other perception media. A multimodal UI comprises text, 2D/3D graphics, speech, image and video perception media as inputs, and text, 2D/3D graphics, video, animation, sound and speech perception media as outputs. The concepts described above also apply for output in any kind of visual perception media. The notion of view has to be generalized for auditory perception media. The MVC pattern also functions with non-visual input, however, a direct manipulation metaphor for non-visual perception media cannot easily be created.

### 3.3.9.2 The User’s Role

In order to make a metaphor usable for the user, the user must perceive it and interact with it. This happens through input and output presentation media that convey modalities. The metaphor must define the mapping from information in input presentation media to the context of a metaphor (e.g. hitting keys on the keyboard corresponds to inputting text, or moving the mouse with a button pushed corresponds to dragging). Moreover, the metaphor also must define consistent behavior when it communicates information to the user by mapping its own context to output presentation media (e.g. echoing text in an input field as the user types, or moving an item on screen while the user drags).

Two common input presentation media are the keyboard and the mouse. Two more advanced input presentation media that are becoming accepted in enterprise environments are the microphone and the still or motion camera. The major output presentation medium is the display, loudspeakers can also be considered accepted in enterprise environments. The user facilitates these input and output presentation media in the context of a metaphor to interact with it.

### 3.3.9.3 The System Architect’s Role

The system architect has several architectural levels at his choice. Figure 3.7 provides an overview. The user interface management system (UIMS) is the component that manages metaphors, widgets, actions (also interwidget actions like drag&drop), events (repaint, invalidate etc.), layouts and styles consistently. The UIMS is tailorable using transient or persistent configurations. One level lower, one can access widgets and I/O device abstractions directly. The engine level provides the system architect to access primitives of the graphical (2D/3D) and the sound subsystem (instruments, generators). The lowest level allows for buffer-level access of the hardware, individual pixels and audio samples. A further task of the UIMS is to coordinate
accesses to the architecture at different levels (widget vs. graphical primitives vs. framebuffer), such that they remain consistent.

The architecture presented above is a simplification of real environments but reveals enough details to relate concrete systems to it and discuss their features. Some architectures do not have a widely configurable UIMS (e.g. MS Windows), or do not present UIMS functionality at all (e.g. X Window intrinsics). What concerns widgets, traditional widget sets (e.g. X Window Motif) are usually hierarchic containing monolithic widgets (e.g. no styling, no reuse of other widgets), more modern widget sets (e.g. QT [78] for X Window, Java’s Swing [31]) already contain styling, however still monolithic in nature. Nowadays, MVC is becoming relevant, it is heavily used in some late GUI environments (e.g. Java’s Swing).

The engines have a different access path, defining separate access for text, 2D graphics, 3D graphics, sound etc. Not all the levels are accessible from every environment. For instance, early X Window [53] and MS Windows’ GDI [63] did not allow for accessing buffers directly from applications. Nowadays, graphical subsystems usually allow for accessing the framebuffer, having simple abstractions on it (e.g. MS Windows’ DirectX [12], Linux’s GGI [8]).

The system architect’s role is to realize user interface concepts at the right level. The right level means the maximization of reuse and the maximal abstraction of lower level details. However, the need for combining information in different perception media requires access through different interfaces in the user interface architecture vertically (levels) as well as horizontally (perception media). Combination requires special widgets (e.g. Java Swing’s Text Editor Kit [27]) that allow for consistent mixing of levels.

### 3.3.9.4 Extending Issues

User interfaces are being generalized and extended to cope with the challenges posed by different perception media and direct manipulation. In traditional GUI widget sets, there is a differentiation between viewer widgets (e.g. image viewer), control widgets (buttons, sliders, forms etc.) that are embedded in a transient dialog or a persistent pane, editor and direct manipulation widgets. Newly, there exist some endeavors to blur the border between these kind of widgets (e.g. Fresco [6]). A further step in blurring is the merger of medialization (chapter 3.3.8) and user interfaces (e.g. Berlin [18]). This seems to be advantageous for the following reasons:

- it can enable seamless access to different abstractions of the GUI level in vertical directions, such that these levels can be mixed in a consistent a straightforward way without using independent interfaces.
it can enable seamless access to different abstractions of the GUI level in vertical directions, handling different perception media in similar ways

- it can enable higher level widget reuse, since unlike in traditional widget sets, views would not be monolithic

- it better enables direct manipulation

- the rigidity of UI leads to the development of own products (e.g. Mozilla [7], video editors [23], etc.)

3.3.10 Multimedia Publishing and Presentation

The Multimedia publishing and presentation domain deals with the authoring, distribution and presentation of information in various perception media forming preorchestrated scenarios. This domain defines content as the main asset to manage. The content to be presented could theoretically be split into «raw-material» and description of how to present the «raw-material». This would then correspond to the content/meta-data terminology. However, this distinction cannot always be made in this domain, since «raw-material» and the description together define the content, which is the indivisible asset of this domain.

The reader can find several parallels between this domain and user interfaces (chapter 3.3.9). The main difference between these two domains is that user interfaces provide simple building blocks and collaboration patterns for creating a front-end to an application for the user, whereas in this domain, the distinction between the application logic and the front-end is rather blurred: the publication as an indivisible unit is published and presented to users.

The user can also find several parallels between this domain and content processing and compositing. There is a clear differentiation between these domains. Content compositing results in information in image, video and sound perception media containing artifacts and their structure, spatial and temporal behavior implicitly, and without the notion of interactivity. This domain possesses a broader view on making these features explicit. This is necessitated by the dynamic, interaction-dependent behavior. Having said that, this domain might make heavy use of the content processing and compositing domain as to a material provider of the publishing for material that cannot be handled dynamically in a presentation environment.

3.3.10.1 The Conceptual Designer’s Role

This domain is the merger of the publishing and presentation domains. The reason why they are handled together being that the two domains are strongly related through the content the one domain produces and the other domain consumes.

The multimedia publishing domain can be characterized by the concepts of the publishing process. The process is made up of authoring, preparation and distribution. Authoring contains three phases. The first phase is the finding, creation and capturing of content. The second phase is the collection of content, the third phase is the manipulation of the collected content. The authoring process specifies the perception media kinds along with their spatial, temporal and interactive behavior of their artifacts.

The output of the authoring process is a new quality reflecting the intellectual and creative processes of the author. After the authoring of newly created content, it is prepared for publication. This consists of annotations and the conversion of the content from a work format to a publishing format. The distribution disseminates the publication either on-line (typically internet) or off-line (typically CD-ROM). A detailed description of the multimedia publishing process can be found in [42].

The source of the presentation process is the content in the representation media. The presentation process is not aware of storage and transmission media\textsuperscript{10}. The presentation process presents the content as it was

\textsuperscript{10}It would be obvious to include streaming as an on-line distribution mechanism, however, this mechanism falls outside of the publishing and presentation domain.
3.3. **MULTIMEDIA INFORMATION MANAGEMENT DOMAINS**

preorchestrated by the authoring process, with respect to the **temporal**, **spatial** and **interactive** behavior of the presentation.

The temporal behavior (also called temporal layout) can be defined using temporal models. We distinguish between **interval-based**, **axes-based**, **control flow-based**, **event-based** and **script-based** synchronization models. These models feature several advantages and disadvantages. Fundamental features are the representation of hierarchies, interactivity and QoS in an intuitive way. A comprehensive comparison is described in [70].

The spatial behavior (also called spatial layout) can be defined in terms of a spatial model. The spatial model describes either the **absolute** spatial position, the **directional** relation or the **topological** relations of the artifacts. The absolute positioning gives the author the maximal precision to position artifacts, other methods allow for higher level specification of spatial layout.

The concept of **interactivity** is described by different qualities of interaction. If the scenario has a **sequential** temporal behavior (timeliness) with no user interaction, we call it **passive**. If the user is encouraged to make selections or adjustments in order to let the presentation proceed (e.g. paging, selection from a list provided), its a **reactive** interaction. If the user can take the initiative to change features independent of the sequential behavior of the presentation, its a **proactive** interaction (e.g. navigation, speed control). If the presentation is completely controlled by the user, it allows for **directive** interaction (e.g. editing artifacts). Multimedia publications exclude proactive interaction, since the directive behavior is typical for the author, rather than the consumer.

Another classification differentiates flow-dependent interactions (e.g. navigation) from flow-independent interactions (e.g. volume setting). Interaction impacts both the temporal and the spatial behavior of artifacts, therefore a publication has to define these in a consistent manner.

The concept of **structure** in the publication bridges the semantics of the authoring and the consuming processes. Structure introduces a useful abstraction for the temporal, spatial and interactive behavior in authoring as well as in consuming. Publication structure can be based on **streams**, **hierarchies** and **links**.

Stream is a linear sequence of artifacts. It defines the «flow» of perception and presentation (e.g. reading). Hierarchy defines a set of artifacts that are in a containment relationship, having a certain root. Typical a hierarchy would contain publication contents, index, chapters, paragraphs etc. Links introduce the **hypermedia** paradigm. This paradigm defines **anchors**, **links**, **link-ends** and **anchor roles**. An anchor identifies an artifact that is to be linked. The granularity of the anchor may vary from an atomic referenceable unit (e.g. character) to a whole publication. Links connect anchors through their link-ends. A link-end is the part of the link that designates an anchor. A link has at least two link-ends. The anchor role specifies the function of a particular link-end. Links can be typed such that the set of possible anchor roles are fixed. An anchor can be reused for several link-ends through different anchor roles.

### 3.3.10.2 The User's Role

We can identify two user roles in the multimedia publishing and presentation domain: the **author** and the **consumer**. Since the two domains are strongly coupled, most features of the publication are common for both roles. The publication's features from both user's view are **navigateability**, **annotateability**, **searchability**, and **adaptability**.

Navigateability allows the user to traverse the structure along a **trail** other than the one defined by streams. This feature has to be supported by the author by providing appropriate structures in the publication. Annotateability means the possibility to create user-centered additions to the publication. These annotations can be made of artifacts in any perception medium, preferred media are text and graphics, typical kinds are highlighting, marginal notes etc. The author cannot account for user annotations, these are personal additions that do not impact the publication. Searchability is a complex feature, it can reach from simple full-text search to content based search described in chapter 3.3.5. The user can only apply search functionality that is either
provided by the application (e.g., index) or provided by a tool at the user's disposal that can access the publication. Adaptability is comprised of two parts: adaptation to personal interest and technical infrastructure. The users' role addresses the first part, the second part is handled in the system architect's role.

Adaptability to personal interest is called personalization and it can either be based on static or on dynamic attributes. Static attributes emphasize the author's role: she must include adaptable alternatives into the publication and provide selectability. Dynamic attributes are calculated at presentation time. Personalization is split into publication personalization and environment personalization. Publication personalization refers to the structure, the temporal and the spatial behavior as well as to the interactivity of the publication.

Publication personalization is based on the separation of structure and appearance. Structure personalization depend on the required amount of knowledge to be transferred, such as overview, medium level, in-depth knowledge etc. Appearance personalization refers to the behavior of the publication, which can be broken down into spatial, temporal and interactive behavior.

Environment personalizations modify the appearance and functionality of the environment in which the publication appears.

The author develops content using a particular development environment that supports the phases of the authoring process. The main features of the authoring environment is productivity. Productivity is impacted by the level and forms of reuse the author can facilitate in the authoring process. Authors can facilitate reuse in two forms: content and design. Content reuse refers to the artifacts, design reuse refers to the structure, the spatial and the temporal behavior of artifacts.

3.3.10.3 The System Architect's Role

Publishing and presentation domains are connected through the format of the publication. From architectural point of view, the publishing domain produces an independent, standalone publication, which is published either off-line through a mobile storage media, or on-line through a transmission media. At the user's site, the publication is executed in a certain context. Figure 3.8 gives an overview of a typical architecture.

The author works with tools that are available in an authoring environment. The authoring environment contains tools as plug-ins that extend its functionality. Authoring environments usually are tailored to a certain format. Popular open formats for multimedia authoring are (D)HTML [20], HyTime [69], MHEG [3] and
SMIL [19]. The key feature of these formats is that they are all text-based scripts defined in the SGML/XML meta-grammar\(^\text{11}\). Scripts only contain text perception medium inherently, other perception media kinds are referenced. Integrity must be enforced by other means (e.g. document management).

Authoring environments are usually monolithic. They do not collaborate with document management tools to find content and to ensure consistence of the authored material, but they provide this functionality on their own (e.g. Adobe's GoLive [47]).

The publishing formats define not only the authoring but also the consuming process. The presentation engine is a black-box-like execution environment that presents the publication. Popular execution environments are web browsers, which are capable of both off-line and on-line publication execution. Web browsers support new formats through plug-ins (based on mime-types). The form of plug-in is a dynamic library that is capable of presenting a certain format. The browser environment reserves a rectangular visual area for the plug-in and dispatches user input if that area gains focus.

The architectural support for personalization is twofold. Publication personalization only concerns behavioral aspects. In case of the text perception medium, separation can straightforwardly be done (e.g. XML/XSL [21]). In case of generic perception multimedia formats, these distinctions are not possible with current technology. Environment personalization depends on the environment itself. This can impact the functionality (e.g. searching capabilities) and the look and feel (e.g. navigation buttons) of the environment.

Authoring environments are usually bound to a certain publication format, there do not exist generic authoring environments for which publishing tools producing a certain format can be attached. Reuse of the authored material is also limited [67], mainly design reuse is missing.

3.3.10.4 Extending Issues

We can recognize several similarities between the content processing and compositing and this domain, which have been discussed at the beginning of this chapter. The differences have been pointed as being interactivity and consumer rendering capabilities. The seamless combination of these two domains would provide the following advantages:

- the author could have the possibility to combine compositing functionality with interaction seamlessly
- the author could decide dynamically which interaction independent parts of the publication have to be rendered in advance, which may remain dynamic
- reuse in the authoring process would be more efficient combining interaction patterns with compositing patterns

The combination of the user interface domain with this domain would also prove several advantages. Uniformity in the presentation would borrow consistence from styling and theming mechanisms.

The combination of the content-based retrieval and the publication domains would allow an extension of the search capabilities. Publication presentation environments (e.g. web browser, PDP viewer) already contain simple fulltext-search functionality, the extension of this search would require the presentation environment to have a tighter integration with the content based retrieval domain.

The combination of this domain with the document management and workflow domain would allow authors to enable finer granularity reuse of the authored material.

Personalization of the publication and the presentation environment are separate. The lack of uniformity might be disturbing, a typical example is the simultaneous presence of navigation buttons in the browser

\(^{11}\text{The XML binding will only be available in MHEG8. DHTML is a marketing term for the «HTML-JavaScript-CSS» triplet, where HTML can be described by XML.}\)
environment an in the publication, providing a different look&feel and functionality. An endeavor that merges the environment with the publication is XPFE/XUL from the Mozilla project [7]. However, the combination of personalization of publication and presentation might threaten the integrity of the publication. A solution to this can be the utilization of a rule-set and distinction of access rights at different granularity levels. These issues have not yet been addressed.

3.4 Conclusion on Multimedia Domains

In this chapter we make several conclusions on the domains described above. For the conclusions, we first set a context and define conclusion viewpoints.

3.4.1 Context

We introduce a new stakeholder of the system, the multimedia application developer, and define his role:

**DEFINITION:** The role of the multimedia application developer is to build an application by designing application logic and realizing it using existing multimedia-oriented domains.

Combining domains for building complex systems is an absolute necessity, since none of the domains provides comprehensive functionality for multimedia; this is obvious from the domain descriptions. In order to be able to build a multimedia-oriented cross-domain application, the application developer needs support from the conceptual designer and from the system architect. For cross-domain application development, we can define three approaches:

- **amoeba:** to take one existing domain and extend it with all the possible architectural and conceptual solutions
- **glue:** to leave everything as is, and provide glues between domains on demand
- **army:** to introduce a uniform, generic domain with common concepts and architecture that can uphold all the multimedia-oriented issues

In order to decide which option to take, we need to examine these from the different viewpoints of the stakeholders.

3.4.2 The Multimedia Application Developer’s View

The multimedia application developer gains maximum productivity, when he is provided with comprehensive and consistent concepts. From his point of view, both the amoeba and the army approaches are consistent and comprehensive, however, in case of the the glue approach, he needs to master inconsistent concepts to build a cross-domain multimedia application.

However, in order to evaluate the cross-domain approaches in other stakeholder’s view, we need to elaborate more on what comprehensiveness and consistence means for them.
### 3.4. CONCLUSION ON MULTIMEDIA DOMAINS

<table>
<thead>
<tr>
<th>IM functionality/domains</th>
<th>text</th>
<th>graphics</th>
<th>animation</th>
<th>video</th>
<th>image</th>
<th>speech</th>
<th>sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>content based retrieval</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>content processing and composing</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>user interface</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>document management and workflow</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>multimedia storage and streaming</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>modeling and mediation</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>multimedia publishing and presentation</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of multimedia-oriented domains with respect to different multimedia kinds

#### 3.4.3 The Conceptual Designer’s View

In order to describe the conceptual designer’s view, we have to define comprehensiveness and consistence. In the context of the multimedia-oriented domains, we relate comprehensiveness to the support for perception media, and management functionality. Consistence in this context is related to how the management functionality is applied to information in the same perception medium across domains, this we call horizontal consistence, and to information in different perception media within the same domain, this we call vertical consistence.

**3.4.3.1 Comprehensiveness**

Table 3.1 gives an overview on the typical support of perception media in different domains. It can be observed in the table that regardless of the level of support given in a domain for different perception media, their presence is not comprehensive.

The management functionality is presented in table 3.2. What can be recognized in the table is that management functionality given for information in different perception media is not comprehensive.

**3.4.3.2 Consistence**

Now examine consistence. Horizontal and vertical consistence can be studied based on the columns and the rows of table 3.3, respectively.

What concerns horizontal consistence (columns of the table), we can recognize that for a certain perception medium, firstly, there is an overlap between functionality, secondly, the same functionality is accomplished differently. While the first assertion can be recognized in the table, the second needs some elaboration, however, we pick some interesting issues, rather than being exhaustive.

---

12 Unfortunately, for technical reasons, horizontal and vertical consistence are not aligned with the layout of the table.
<table>
<thead>
<tr>
<th>IM functionality/ domains</th>
<th>modeling</th>
<th>creation</th>
<th>storage</th>
<th>querying</th>
<th>transfer</th>
<th>processing</th>
<th>presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>feature,</td>
<td>-</td>
<td>multi-</td>
<td>ad-hoc</td>
<td>-</td>
<td>feature ex-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>ad-hoc</td>
<td>query,</td>
<td>-</td>
<td>traction,</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>similarity</td>
<td>query</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>content processing and</td>
<td>operation,</td>
<td>compositing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>keying,</td>
<td>editor</td>
</tr>
<tr>
<td>compositing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>user interface</td>
<td>widget,</td>
<td>UI builder</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>widgets</td>
</tr>
<tr>
<td>document management and</td>
<td>document</td>
<td>editing</td>
<td>DBMS,</td>
<td>on content-</td>
<td>workflow</td>
<td>editing</td>
<td>-</td>
</tr>
<tr>
<td>workflow</td>
<td></td>
<td></td>
<td>file-system</td>
<td>independent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multimedia storage and</td>
<td>stream,</td>
<td>circuit</td>
<td>streaming</td>
<td>server</td>
<td>directory</td>
<td>streaming</td>
<td>-</td>
</tr>
<tr>
<td>streaming</td>
<td>setup</td>
<td>server</td>
<td>server</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeling and medialis-</td>
<td>real-world,</td>
<td>file-system</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>model</td>
<td>viewing,</td>
</tr>
<tr>
<td>ization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>computing</td>
<td>inspecting</td>
</tr>
<tr>
<td>multimedia publishing and</td>
<td>scene,</td>
<td>authoring</td>
<td>-</td>
<td>full-text</td>
<td>on-line</td>
<td>rendering</td>
<td>publication</td>
</tr>
<tr>
<td>presentation</td>
<td>synchronization,</td>
<td></td>
<td>query</td>
<td>query</td>
<td>off-line</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>link, link-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>end, anchor,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>anchor-role</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Summary of multimedia-oriented domains with respect to information management tasks and concepts

<table>
<thead>
<tr>
<th>IM functionality/ domains</th>
<th>text</th>
<th>graphics</th>
<th>animation</th>
<th>video</th>
<th>image</th>
<th>speech</th>
<th>sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>query, index, store index</td>
<td>-</td>
<td>-</td>
<td>query, index, store index</td>
<td>query, index, store index</td>
<td>query, index, store index</td>
<td>query, index, store index</td>
</tr>
<tr>
<td>content processing and</td>
<td>-</td>
<td>-</td>
<td>create (only image-based)</td>
<td>manipulate, compress</td>
<td>manipulate</td>
<td>-</td>
<td>manipulate, compress</td>
</tr>
<tr>
<td>compositing</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>user interface</td>
<td>input, edit, present</td>
<td>present</td>
<td>-</td>
<td>present</td>
<td>present (play)</td>
<td>present (play)</td>
<td>-</td>
</tr>
<tr>
<td>document management and</td>
<td>edit, version, create, store</td>
<td>-</td>
<td>-</td>
<td>scan, recognize, store</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>continuous media</td>
<td>-</td>
<td>-</td>
<td>transfer, store, retrieve by name</td>
<td>-</td>
<td>transfer, store, retrieve by name</td>
<td>transfer, store, retrieve by name</td>
<td></td>
</tr>
<tr>
<td>modeling and medialis-</td>
<td>-</td>
<td>-</td>
<td>create (render), view, edit, inspect</td>
<td>create (render), view, inspect</td>
<td>-</td>
<td>create (render)</td>
<td>create (render)</td>
</tr>
<tr>
<td>ization</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multimedia</td>
<td>edit, layout, create, present</td>
<td>create, present</td>
<td>create, present</td>
<td>-</td>
<td>present</td>
<td>-</td>
<td>present</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of multimedia-oriented domains with respect to different multimedia kinds and information management tasks
3.4. CONCLUSION ON MULTIMEDIA DOMAINS

For text, there are several ways of creation: text is created differently in the user interface domain through widgets, than in multimedia publishing through WYSIWYG editors. Storage is also different, editors usually store text in the file-system in their own format, this is typically different in domains, for instance SMIL in presentation and ASCII in content-based retrieval. As to graphics, the most prominent example is manipulation: in modeling and medialization graphics can be edited and rerendered, in user interfaces, it cannot be changed. Animation plays a different role in modeling and medialization, and content composition: in the first domain, usually only keyframe-based animation is supported, in the second domain full animation rendering is possible. Video can be retrieved by name in the continuous media domain, however, in the content retrieval domain, complex queries can be formulated. Images can be presented in the presentation domain, however, the user interface also gives zooming, replacement support for them during presentation. In the modeling and medialization domain, speech parameters can be changed during playing, however, in the presentation domain, it is not supported. For sound, that same as for video can be applied.

What concerns vertical consistence (rows of the table), a certain domain manages different perception media kinds separately. The content processing and composition domain makes a distinction between audio and video, despite the fact that most mechanisms could uniformly be applied to information in both perception media. In the same domain a further distinction is made between synthetic and non-synthetic perception media (e.g. different environments for video editing and compositing), despite the fact that due to the nature of the domains, both are handled as non-synthetic media. The content based retrieval domain also makes a distinction between different perception media kinds, despite that their indexing structures are similar or identical. The distinction also exists in the retrieval process: retrieval systems are tailored to a single or rigid combinations of cross-modal retrieval kinds, rather than being flexible on the input and output modality. The document management and workflow domain is traditionally based on scanned images and text, even if they are extended to a variety of perception media kinds, their access is different not suggesting exploitable similarities. The continuous media storage and streaming domain is traditionally separated into streaming audio and video, however, their streaming is fundamentally the same. Streaming information in other perception media, such as text requires yet another streaming and storage component. The modeling and medialization domain also differentiates between the dimensionality and the perception of information, such as 2D, 3D graphics, sound in terms of rendering and origin of data. The user interface domain is traditionally based on 2D graphics, however, 3D graphics cannot be handled in the same way, not to speak about sound interfaces and the MVC paradigm. The multimedia publishing and presentation domain is based on historic foundations of text, the domain still handles text uniquely, differentiating it from all other perception media, even if it gained cleaner design and higher reuse not doing so.

3.4.3.3 Content and Meta-Data

Some domains, such as content-based retrieval and document management are historically built on the notion of metadata. In general, there are three main shortcomings of the meta-data classification, presented as an analysis viewpoint. These are its self-consistency, its alignment with our domain terminology, and its focus. The self-consistency problem is that the above classification does not make a clear distinction between instance-level and type-level meta-data. This is due to the fact that the term «data» is not defined as a reference point with respect to a certain domain or model context. The alignment problem is that, due to the previous shortcoming, there is no unambiguous placement of meta-data within a domain. The focus problem is that content-independent meta-data is not multimedia-oriented, it can be attached to any kind of data. Therefore, it is beyond the scope of multimedia from the perspective of this work.

Apparent from the domain analysis, meta-data does not have a central role in all the domains. The conclusion from this is that in the remainder of this paper we abstain from using the term «meta-data» in any context. We separate the notion of content-dependent and content-independent meta-data. The facilitation of content-independent meta-data in beyond the scope of multimedia information management. We handle it only in the integration chapter (chapter 6). In case of content-dependent meta-data, if a certain unit of information describes another, we can place this information into the role of describing other other information without
CHAPTER 3. ANALYSIS OF MULTIMEDIA INFORMATION MANAGEMENT

associating it with a permanent meta-attribute. The role and its application will depend on a given certain context.

3.4.3.4 Approaches

If the conceptual designer takes the amoeba approach, he needs to describe all required concepts with concepts of the selected domain. This approach is often facilitated, because it proves the competitiveness of the domain. An example domain that is usually extended to give more comprehensive perception media support is the publishing and presentation domain: the WWW is a prominent example of this. However, the management functionality for information in different perception media is severely limited, usually only presentation is supported.

There is another domain that is usually used as a foundation for management functionality extensions in multimedia. These are database management systems (DBMS). The author has not included this as a multimedia-oriented domain, because there is no consensus on concepts of multimedia DBMSs, usually it is a partial merger of several domains. What is however clear from the domain descriptions is that even if this domain claims to have incorporated all the necessary concepts of multimedia-oriented domains, in reality, their concepts are all based on the relational model, which is known for its simplicity and straightforwardness, but fails when used for expressing multimedia concepts.

The user interface domain also strives for incorporating several domains, such as presentation, document management. The recognition of the need for blurring the borders between these domains is justified, however, the user interface domain cannot express all the concepts that are necessary. This results in separate developments to extend the notion of widget, which are become bloated and rather monolithic in nature.

If the conceptual designer takes the glue approach, comprehensiveness could be achieved in perception media as well as in management functionality. This would prove advantages if the number of domains to be incorporated is low. The main shortcoming is the need to possess the knowledge of a diversity of concepts originating from independent domains. Another shortcoming is the overlapping of concepts in the domains to be glued. A characteristic example is the Java Foundation Classes [49], where five multimedia-oriented domains have been developed independently: Swing API for the user interface, 2D API for two-dimensional modeling and visualization, the 3D API for three-dimensional modeling and visualization, the Java Media Framework API for streaming, and Java Advanced Imaging API for image content manipulation.

Another shortcoming of the glue approach is the difference in granularity of a certain concepts throughout domains. This means that a concept cannot be moved through domains with either loosing semantics, or carrying unnecessary semantics irrelevant for the concept.

The army approach requires the greatest effort from the conceptual designer, however, this provides the most comprehensive solution for the application developer. This approach is not commonly taken in case multimedia, but rather in a more general sense. An example is CORBA, that provides a uniform solution embracing the most generic as well as the most specific concepts in several domains, in a programming language-independent way. However, it seems, that except for some specific issues, such as the content streaming domain, the OMG has considered multimedia neither as a common facility nor as a domain.

3.4.4 The System Architect's View

The system architect attempts to realize the concepts of the domains. We consider the three approaches described above.

---

13Here, the author means CORBA as an expression of different abstraction levels of a conceptual model for a domain, rather than CORBA as a programming model or as a distributed architecture.
3.5. WRAP UP: SELECTING AN APPROACH

3.4.4.1 Amoeba

The *amoeba* approach would take a certain domain and use its architecture as a realization for all other domains. The analysis showed that what concerns the architecture of domains, domain architectures are specialized and monolithic, the *dimensions of extensibility* are severely limited.

A domain is usually build around a monolithic substrate that is either not extensible, or extension components can only be applied in one dimension. The content processing domain is a typical example, which is built on pluggable components. The dimension of extensibility is low, it is limited to manipulation of a information in a given *perception medium*. The user interface domain is another example of extensibility, there exists the notion of pluggable look&feel, however, this is limited to global styling, finer-grain configuration is not possible. There is another approach, compound documents, that support the notion of user interface components. The drawback of this approach is that is is bound to a structured file-format, that contains the documents, more generic storage technology cannot be facilitated.

3.4.4.2 Glue

The *glue* approach takes existing domains and makes them work together. Unlike the amoeba approach that required management functionality on perception media to be extensible, in the glue approach, special realizations can be glued together.

Glueing can be realized at two levels: the system architect can glue domains or components of domains. If he glues domains, the whole domain architecture needs to be glued. If the number of domains to be glued together is more than a few, it requires a tremendous effort to remain consistent.

Glueing existing components is a finer-grain approach, however, domains are mostly highly monolithic, meaning that substrate glueing substrate components is almost the same effort than glueing the whole domain.

3.4.4.3 Army

The *army* approach would require the system architect to provide a new, generic architecture on which the concepts can be straightforwardly realized. The efforts that would need to be done are comparable with that of the other two approaches, however, the army approach can provide components that guarantee consistence through a component architecture, and extensibility of the architecture in several dimensions.

3.5 Wrap up: Selecting an Approach

In this chapter we analyzed domains from several viewpoints. We defined three approaches for cross-domain multimedia application development. We discussed what it would mean for the stakeholders to provide each of the approaches for cross-domain multimedia application development. The approach on which the remainder of this work is founded is the *army* approach. The main reasons why other approaches have been abandoned can be summarized as follows:

- The *amoeba* approach cannot embrace concepts of several domains and realize it on a selected domain, because domain architectures are too rigid and specialized. Even if concepts can be realized, their granularity depends on the domain and cannot be changed. The amoeba approach needs a new realization of most concepts to fit into the selected domain, moreover, the foundation domain might also need a substantial redesign to augment the dimensions of its extensibility. It seems that the effort that needs to be taken for the amoeba approach is *disproportional* to its results.
The glue approach cannot bind domains together because comprehensiveness and consistence cannot be maintained across domain borders: the provision of perception media and information management comprehensiveness, as well as their consistence across different domains would require an effort disproportional to the added value of the domains to be glued.
Chapter 4

Common Abstractions in Multimedia Information Management

In hanc utilitatem clementes angeli saepe figuris, characteres, formas et voces invenerant proposuerunque nobis mortalibus et ignotas et stupendas nullius rei iuxta consuetum linguae asum significativas, sed per rationis nostrae summam admirationem in assiduum intelligibilium persequitionem, deinde in illorum ipsorum veneracionem et amorem inductivas.

/Johannes Reuchlin, De Arte Cabalistica, Hagenhau, 1517, III./

4.1 The Goal of This Chapter

In the previous chapter we concluded that the «army approach» is the only viable solution to design and realize comprehensive and consistent concepts for cross-domain multimedia application development. The goal of this chapter is to define these concepts.

This chapter takes a different path than the previous one. The multimedia-oriented domain analysis provided us with which concepts, management functionality and user views are required for comprehensiveness, however, have not been able to select a core set of consistent concepts that will serve as a strong foundation. Therefore, we take a different path. We still maintain the notion of stakeholders of the software system, however, their goal in this context is different. At the end of the chapter, the two paths followed by the two chapters will merge again: we will provide fundamental, common abstractions.

4.2 The User’s Abstractions

Further investigations of this work are centered around digital media in an enterprise environment. The limitation to digital media clearly defines the borders of information to be dealt with in this work, but it does not exclude any relevant management field, since even in activities involving analog media (capturing, presentation), digital media is always present. Limiting the environment to the enterprise is a goal and also a necessity. It is a goal, since it is the topic of this work. It is a necessity, since the topic ‘multimedia’ in general cannot be captured within the scope of such a work as this.

The convergence of user needs in an enterprise environment and possibilities of digital media is being driven by two main factors. On the one hand, users are constantly seeking for more effective and comprehensive ways of expressing, conserving and transferring their knowledge: digital media has proven as an excellently adequate means for this purpose. On the other hand, the technology of digital media is getting mature and

75
cost-effective; it has found its way from special, standalone applications to mainstream environments, such as the IT infrastructure of a general enterprise.

As mentioned previously and suggested by the quote of the previous chapter, the "wonderful feeling after eating" for the user in this context is if she can manage her knowledge in the most straightforward way using media. However, with the maturity of digital media, its complexity reached a level, where users are unable to manually accomplish multimedia information management. Automating multimedia information management needs careful examination: An important goal of the analysis from the user's perspective is to identify and understand the different kinds of knowledge required to manage the increased complexity. The identification leads to the separation of two kinds of domain knowledge: knowledge of the user's primary domain, and that of the media domain. Separating media and domain knowledge eases the burden on the user when deploying media for knowledge representation. In the remainder of this section we recall the notions of metaphor, sensory medium and transform them into more usable abstractions that enable defining the requirements.

4.2.1 Content, Metaphor and Interaction

The only media the user perceives is perception media. The elaboration on the process of perception (and cognition) is not in the scope of this work. Here, we break down perception into two notions: content and metaphor.

**Definition:** Content is an abstract term for denoting information (artifacts) independent from any media and representation. The notion of content is a useful means to eliminate the problem of different representations of information in different carriers. Since content, unlike information, is not bound to any concrete media, it is abstract and must be reified in a certain medium to make it accessible.

**Definition:** Metaphor is originally a figure of speech in which a word or phrase denoting one kind of object or idea is used in place of another to suggest a similarity between them. A multimedia metaphor is a digital artifact in one or more perception media that suggests similarity a real-world artifact. Multimedia metaphors define an interactive context by the artifacts through which content can be expressed. In this work, the term metaphor refers to multimedia metaphor, unless noted explicitly differently. As there is no information if there is no media, there is no content if there is no metaphor.

Metaphors are based on the following four experiences: (1) our existence in space with orientations such as up/down, left/right, front/back, (2) our ability to directly manipulate real-world objects, (3) the behavioral and interactional properties of real-world objects to each other and to us, and (4) our ability to organize our activities on real-world objects around desirable states and goals.

Separating these two notions is necessary because perception media is not suitable for further elaboration. The perception of information is a consequence of the metaphor through which it is communicated to the user, rather than its inherent feature. Therefore, the user's primary abstraction is the metaphor, which defines the perception and interaction of information. Content is of secondary interest; users cannot distinguish between content and metaphor, they may only think of content as the abstract source behind the metaphor. The following example describes a typical situation (perception media are denoted by capital initials):

**Example:** When Animation is created or edited, an animation-editor metaphor is used. If we render Animation, we can present it using a player metaphor, in which it acts as Video, if it is realistic enough no one thinks of it as Animation. Animation in a transmission medium can turn Animation
into Text. We can even edit this Text through a text-editor metaphor, provided we know its representation. Artifacts behave as Animation, Video and Text depending on the metaphor through which we perceive and manipulate them.

Figure 4.1 summarizes the user's main abstractions and their interrelations. Metaphors are powerful means to communicate content to the user, because the similarity to a known thing helps the cognition process, makes the use of that media artifact easier and more intuitively. The similarity of a metaphor to the real world can be based on common appearance and functionality. Metaphors provide context for information content, whereby functionality lets the user certain tasks to be performed on the content, and appearance makes the content appear in a certain way. In this work we call functionality activity, in order to avoid confusion with other usage of the generic term «functionality».

The breakdown of perception into content and metaphor, i.e., into the «what to perceive» and «how to perceive» did not yet account for the user's involvement into the perception process. The user can initiate the perception process and react to what she has perceived. The abstraction that covers the user's role as to participation in perception is interaction:

**Definition:** Multimedia interaction (also interaction) is a mutual action or influence between the metaphor and the user. The interactive context the metaphor creates qualifies the interaction and predefines the user's participation in it. The participation can be passive, reactive, proactive and directive. Passive participation means that the user just passively perceives the content through the metaphor but she has no means to react. Reactive participation calls the user for reacting to a certain event. Without the user's decision no further participation is facilitated. Proactive participation gives the user the initiative: she decides when to initiate communication with the metaphor. Directive participation is a creative process: the user obtains the possibility to create and orchestrate artifacts using existing ones thereby establishing new qualities.

### 4.2.2 Embedding Abstractions into the Enterprise Environment

Integration of multimedia information management into the enterprise environment from the user's perspective can be defined as the integration of business and media management. Describing the integration using the user's abstractions requires the definition of business management:
**Definition:** Business management covers the life-cycle management of a business system. In an enterprise environment, a business system contains business logic and resources in a distributed way throughout the enterprise. Life-cycle management of business logic and resources comprises their design, implementation and deployment. Business management excludes (1) tasks that communicate the logic and the resources from the business system to user, and (2) resources that contain information in perception media.

The role of media management with respect to business management is as follows: media management can act:

- **as conveyor:** it turns business logic and resources into content and conveys it to the user in form of metaphors
- **as enrichment:** it provides extra content that cannot be produced by the business system and also conveys it to the user in form of metaphors

### 4.3 The System Architect's Abstractions

The system architect's task is to reify the user's notion of content and metaphor by managing the mappings of information throughout various digital media, such as the storage, transmission, presentation and representation media. The system architect cannot deal with the sole concept of media for two main reasons. Firstly, physical digital media, that is, storage, presentation and transmission media became generic and versatile (e.g. DVD storage medium as a replacement for vinyl and videotape). Secondly, in order to compensate for genericity and versatility, representation media became specific, trying to embrace the peculiarities of different traditional media and information. Therefore, in order to abstract media, we introduce two new concepts: device that combines representation media with one or more other media, and service that abstracts device capabilities and locality, and we put these new abstractions into the context of the enterprise environment. Figure 4.2 gives an overview of the system architect's abstractions.

#### 4.3.1 Devices

**Definition:** Device is a software interface to any number of any media in a computing environment. Devices can have capabilities that express the realization of a certain functionality within the device. We refer to information on a certain device as device content. Devices always contain a representation medium and one or more further media. The representation medium is realized as the format of the device. Compound devices can be build of simple ones, based on their capabilities and format dependence.

**Definition:** Format is the realization of the representation medium within a device. We can distinguish between two kinds of format: internal and external format. Internal format usually has a structure that is either technology or semantically defined and it describes the format inside the device. External format refers to an unstructured, sequential access, it describes the way of interfacing, that is, writing into and reading out of the device. Device content that is contained and processed by the device has the same external format as its corresponding device.
4.3. THE SYSTEM ARCHITECT’S ABSTRACTIONS

The abstraction of different media by devices is a useful concept, because it allows focusing on a certain abstraction level of representation, depending on the device we select. Devices can vary from simple devices whose format is just a bit stream, to devices whose format is complex (e.g. MPEG-2 stream). The information mapping between media becomes the content exchange between devices. Main device capabilities comprise storage, transmission, filtering and (input/output) presentation of device content. Devices are assigned quality measures that impact the overall quality of media management. These quality measures can be broken down into format quality and content exchange quality. Format quality qualifies the format of the device content, based on the precision of representation (e.g. maximum number of colors, image resolution) and the lossiness of the representation (entropy or source encoding). Content exchange quality qualifies the device, more precisely, its capabilities as to throughput (e.g. number of device content units per second), delay (between requesting and receiving device content) and jitter (of delays in case of periodic reception).

Storage devices are an abstraction of storage and representation media with capabilities relating to the representation. Storage devices store their device content in a certain internal format and make it available through their external format. The applicability of the device depends on the genericity of the required external format on the device. Generic purpose or low-level storage devices pose no requirements as to the external format of device content: the smallest content unit usually corresponds to the smallest addressable unit of the given computing environment. Higher level devices are specialized for the storage and access of a specific format. The minimal capabilities of a storage device comprise the access of content (offsetting, naming), read, write, creation and deletion of content. More advanced capabilities comprise the configuration of the devices internal format (e.g. physical layout on a harddisk) and structured semantics for the device.

Network devices are an abstraction of transmission and representation media. Network devices realize a certain protocol for the transmission of device content. Device capabilities include synchronous or asynchronous transmission capabilities over either isochronous or anisochronous transmission channels. Isochronous channels allow for bandwidth reservation for the transmission of a given device content, anisochronous channels transmit content when bandwidth is available. Synchronous transmission maintains a strong relationship between device content on different devices (minimum and maximum delay for the transmission), whereas asynchronous transmission does not. The realization of synchronous transmission over an isochronous channel is a straightforward task, whereas anisochronous channels necessitate the application of auxiliary techniques (buffering, acknowledgment, interrupted transmission etc.).
Presentation devices are an abstraction of input/output presentation and representation media. Presentation devices are devices that present their device content in a perceptible form. Both input and output devices can handle external format in immediate and retained mode. In the first mode, device content is immediately presented or processed upon availability by the output presentation device or the input presentation device, respectively, whereas in retained mode, processing or presentation is retained till a coherent logical set of device content is available with respect to a certain format.

Filter devices are an abstraction that cannot be related to a certain medium, since they are not supposed to directly store, transmit or present device content. Filter devices manipulate device content. Obviously, manipulation might involve the temporary storage of device content, however, this is abstracted by the device.

4.3.2 Services

**Definition:** Service is a logically coherent set of functionality abstracted from device capabilities. A service does not depend on any device locality, and can be realized across several devices and seamlessly distributed. Typical services include streaming, querying, persistence etc. The quality of a service can be computed from the quality measures of device capabilities participating in the realization of the service.

Services make up the infrastructure in a distributed environment. While devices possess location as their inherent feature, services are location independent, they may involve several devices to provide complex functionality. The most basic services include:

- **persistence:** This is the most fundamental service in an enterprise environment. It mainly builds around storage devices and adds location independence.

- **streaming:** This is a service that coordinates the exchange of device content with respect to path selection and end-to-end qualities along several devices.

- **transaction:** This service assures reliability when creating, accessing and modifying device content. The four main features transactions are characterized by are atomicity, consistence, isolation and durability.

- **query:** This service supports the enquiry of existing information as to arbitrary features that are expressible with a given query mechanism. Querying is an essential feature in information management in general.

- **naming/trading:** This service supports localizing device, device content and services based on their names or other arbitrary features.

4.3.3 Embedding Abstractions into the Enterprise Environment

The enterprise environment poses several requirements to the system architect. The embedding of media management into a computing environment has three depths (similarly to [70]): (1) independent management, (2) integrated resource control and (3) integrated resources. Media management should be integrated into the enterprise environment in the third way. An advantageous way of integration is bound to a high level of reuse throughout the enterprise environment: resources (devices and services) are shared among media related and non-media related tasks. This demands the usage of mainstream resources with well-known, open, generic and highly configurable capabilities.
4.4. **The Conceptual Designer’s Abstractions**

4.4.1 **Independent Structure**

The task of the conceptual designer is to establish a *common language* between users and system architects by creating *consistent semantics* for media management that contains *both* the user’s and the system architect’s abstractions. The notion of information appears in both abstractions. In the user’s abstraction, content is the abstract information source for the metaphor and thus it is bound to a metaphor, whereas in the system architect’s abstraction, device content is information representation with respect to a certain device. There needs to be a third notion of information that connects the above two notions that should neither depend on the metaphor nor on the device. This notion is the content *structure*.

**DEFINITION:** (Content) *structure* is a hierarchical division of information regardless of its carrier, format and appearance. The smallest unit of structure corresponds to the smallest piece of content that can appear to the user through any metaphor as well to the the smallest piece of device content that can be processed by a device. There are no upper limits for the biggest unit of structure by definition.

Using the above defined notion as the *common language*, we can express the different stakeholder’s abstractions using *structure* (see also figure 4.3). We can now express *metaphor* as the appearance of the structure and the activity applied on the structure, abstract *content* as the structure and the device content, and *device content* as the *localized* realizations of the structure on devices.

4.4.2 **Appearance**

We first give a definition of appearance:

**DEFINITION:** Appearance is the look and the feel of content structure as it defines the metaphor.
We can describe five basic appearance types:

- **visual**: The most fundamental appearance, the vast majority of content delivery involves metaphors that have visual appearance. Visual appearance is perceived by the user through the visual sensory channel. The main characteristics of visual perception involve the distinction between about 4 million colours, a spatial resolution of about 2 steradian. The perception of depth (3D) can be approximated by special 2D techniques (projections, shades, colors etc.).

- **auditory**: We speak about auditory appearance if content is (among others) communicated through the auditory sensory channel, so that the user can hear it. The frequency of sound humans can perceive ranges from 20 Hz to 20 kHz. Spatial perception of sound can be approximated by several sound sources (2, 3, 4).

- **temporal**: One of the most exciting features of appearance is the temporal appearance, that is, its behavior in time. We can distinguish between time-independent appearance (e.g. in case of text, graphics) and time-dependent appearance (e.g. video, music, animation). Humans can perceive continuous movement above 16 phases in a second.

- **interactive**: Interactive appearance is an abstract appearance, it describes how the metaphor reacts to interaction. The reaction can either be triggered by the user or by a media artifact. If the trigger comes from the user, an input presentation media is used, typically the mouse and the keyboard. Since interactive appearance is abstract, it must be expressed using one or more of the other four types.

### 4.4.3 Activity

Activity is the other constituent of the metaphor. We give a definition:

**Definition**: Activity is the functionality of the that can be applied on content structure as it defines the metaphor.

Activity, like appearance contributes of the definition of metaphors. Activity is a substantial constituent and qualifier of metaphors, since it contributes to the qualification of the metaphor to a certain perception medium. A typical example given below shows how activity contributes to qualification:

**Example**: If the user sees text as part of a metaphor on a screen, not the text itself, but the metaphor defines which perception medium the metaphor belongs to. If the text is shown in a video clip on which the user apply start and stop activities, the metaphor conveys information in video perception medium. If the user is capable of applying activities to change the viewangle of a 3D textual artifact while the clip is being played, it is animation perception medium. If the clip can be stopped and the user can apply an editing activity on the textual artifact, it is text perception medium.

Activities shall reflect the information management functionality, as defined in chapter 3.2.3. The only difference is that modeling is excluded, since the model is explicitly present in terms of structure, appearance, device and device content.

### 4.4.4 Conceptual Media and Resources

In order to simplify the existing concepts, we eliminate redundant ones and combine the rest into few comprehensive ones. We eliminate metaphor, content and media, since they have already been generalized, as the metaphor and content into structure, appearance and activity, or specialized, such as media into device
and service. The remaining six concepts are subdivided into two comprehensive ones: conceptual media and resource (figure 4.3).

**DEFINITION:** Conceptual Media is a complex concept comprising four main abstractions, namely device content, appearance, structure and activity.

**DEFINITION:** Resource is a complex concept comprising two main abstractions, namely device and service. Resources comprise the realization environment for conceptual media.

### 4.4.5 Typing and Classification Of Conceptual Media

Figure 4.3 depicts all the relevant abstractions and the place of conceptual media as a comprehensive abstraction. Since media-types as qualifier are compound having many features, the selection of the type-qualifier is not straightforward. In the following section, we need an initial primary basic classification for the analysis of the media-types. It is initial, because there exist other classifications which are defined later. It is primary, because it creates a primary order and naming convention for media-types, but it does not have any impact on the analysis of the features. It is basic, since it does not strive for completeness, it just focuses on the basic media-types.

**DEFINITION:** Type is a concept used to classify other concepts based on sharing common features. Common features (e.g. behavior) are the enabler for the substitutability of concepts in a certain context. Type always implies qualification, that is, a decision whether a concept belongs to a certain type. The term type can refer to the description of the type as well as to the set of concepts that belong to that type.

**DEFINITION:** Media-type (MT) is a compound and abstract type that classifies conceptual media. Qualifying conceptual media can be based on any of the four features that constitute conceptual media.

**DEFINITION:** Media management is the life-cycle management of conceptual media in several media-types.

### 4.4.6 Embedding Abstractions into the Enterprise Environment

The abstractions of the conceptual designer on media management complement the abstractions of business management. Therefore, conceptual media abstractions should not be merged with business abstractions, they shall be kept as independent notions. The main endeavor of the conceptual designer is to enable seamless interoperability between the abstractions of business and conceptual media.

### 4.5 Basic Media-Types

#### 4.5.1 Media-Types as Common Abstractions

Media-types can be considered as common abstractions because of two reasons, details will be shown in this chapter:
media-types are considered common abstractions for all the stakeholders, that is, the user, the conceptual designer and the system architect, since media-types contain all their abstractions.

media-types are considered common abstractions for all the domain concepts described in the domain analysis earlier and the abstractions of this chapter.

There is no theoretical limit for the number of media-types. Our initial typing is based on the perceptual feature of the appearance. We select perception as the main qualifier in spite of having it already broken down into more precise abstractions in chapter 4.4.2. The reasons are that (1) the selected feature shall not be too fine-grain (resulting in too many types), the feature shall be (1) intuitively and historically distinctive, (2) and it shall depend on user issues, rather than technology. In this chapter, we describe image, text, graphics, animation, video, speech, music and speech media-types.

We do not include cross references in the following chapters describing media-types, since the name or reference to the stakeholder and the domain name unambiguously identify the references. The abbreviations we use are CBR (content-based retrieval), CPC (content processing and compositing), DMW (document management and workflow), CMS (continuous media storage and streaming), MED (medialization), UIF (user interfaces) and MPP (multimedia publishing and presentation).

4.5.2 Image

4.5.2.1 Structure

The basic structure of an image is the common basis for almost all domains, it is described in the CPC domain. The smallest unit of an image is a pixel. A pixel is an atomic point in a 2D presentation surface. The biggest unit of the structure is the image itself, a clearly defined 2D region. Images are usually rectangular, but they can have an arbitrary shape. The main structure of the image is defined by the channels (planes), whose superposition defines the final image result. The channels contain color and alpha channel information.

The basic structure is extended (MPP, CBR) with spatial and content semantics. An image can be divided into smaller (typically rectangular but also freely shaped) areas. These areas can be of uniform size (grid), hierarchically subdivided in a tree-like fashion, or independent, even overlapping. This extension can serve as a basis for spatial querying (CBR), or interactivity (MPP).

The CBR domain extends the structure to a collection of images. An image can be described by points in feature space. Therefore, the feature space is a representation of an image collection. The efficient representation of a feature space is a tree-like index structure.

4.5.2.2 Appearance

Visual appearance is the main feature of images. As described in the CPC domain, visual appearance comprises the size, the zoom (x-zoom and y-zoom), the alpha channel and the color space of the image (bitonal, grayscale, indexed color, hi-color, true-color).

Interactive appearance, as based on the MPP and UIF domain, can be triggered by input devices. Common interactive appearances are triggered by mouse activities. These are the change of the image (e.g. pressed vs. depressed), the Pop-up of a new image, the disappearing of an image, and the systematic change of colors (e.g. inversion, area-based color change). These interactive appearance kinds can be triggered by mouse click, drag and drop, point and shoot, and move over.
4.5. BASIC MEDIA-TYPES

4.5.2.3 Activity

The creation of images is broken down to capturing and editing. Capturing creates device content in a device dependent format. Editing can also be used to create images in an interactive way on a pixel-basis. The created images can be stored on a storage device in a certain device content format. Transfer of images is can be asynchronous, since they do not possess time-aware behavior. Presentation of images needs a 2D surface.

The processing of images is a collection of all domains. Interactive processing is editing, it is described by the CPC domain and it is done on a pixel, or pixel-group basis. Another processing kind is feature extraction, originating from CBR. The compression of images is demanded by all domains that store and transfer images (DMW, MPP).

Querying on images is also composed of several domains. CPC provides pixel-based queries, such as color values, image-size, channels etc. Content-based queries (from CBR) query the index-structure of image collections for relevant features. An interactive querying type is hypermedia (MPP), that provides further information based on image hotspots.

4.5.2.4 Device Content

Device content plays a role in all domains where content needs to be stored, transferred or presented. Image device content is either device dependent or device independent. We separate those two issues.

Device dependent content is used in all the domains, where content is manipulated and presented. Device dependent image representations are either raw (RGB) or perception aligned (YUV) bitmaps. Each component may have a separate resolution.

Device independence is used for storage and transfer. This is mainly used in the DMW domain. Device independence allows for transferring the content intact through several devices. In order to contain representation parameters, such as size, color model, colordepth etc. content formats have a header prepended to the bitmap content. Device dependent content may be indexed, the header also contains a palette, and the content information is a set of indices to the palette.

Compressed device independent content is very popular, since compression reduces bandwidth and storage needs. Common lossless compression formats are GIF and PNG, the main lossy compression formats is JPEG. All these representations can be asynchronously streamed between devices.

4.5.2.5 Resources

The devices that are needed to store device content of images can be generic. In case structure and appearance are not contained in the device content, they need to be stored as well in a serialized form. Generic network devices are able to transfer images using common network protocols.

Filtering devices fall into two categories. The one that work on device-dependent content, such as content manipulation, and devices that convert between different representations.

The simplest and most specific presentation device is the framebuffer device. Its usage predetermines a color model and a color depth. Generic presentation devices can accept device content in device independent formats and make the conversion themselves.

4.5.3 Text

Text may not be the visually most exciting media type, but it has a paramount role: the vast majority of information available in machine-readable format is text. Text media-type is based on the fact that the language of human communication can be transcribed to a visual form.
4.5.3.1 Structure

Text has a rich structure. The smallest element of text is the character. Characters are either alphanumeric (representing the letters of the alphabet of a certain language and the numbers to the base ten), or special comprising punctuation, currency signs etc. Characters are used to form all other structural elements of text, such as words, sentences, paragraphs, sections, chapters, parts etc. This structure is inherent to the DMW and the MPP domains.

The traditional reading order is sequential over a hierarchical structure (kind of depth-first traversal), however, hypertext introduced in the MPP domain created a new dimension to text structure.

Another fundamental kind of structure is the table. Tables are two-dimensional structures for alphanumeric characters. The basic structure, as opposed to the hierarchy, is grid-like, so that elements of the table, cells, can be addressed by specifying their rows and columns. The first row and columns can be special, containing descriptions for the measures and names of the dimensions.

Collection of texts is structured according to the representation of their features in the feature space.

4.5.3.2 Appearance

The basic notion of text appearance is the glyph. Glyphs are the rendered form of characters on a presentation medium (typically paper and screen) that define their appearance. The coherent set of glyphs are fonts. Fonts possess features that are described by visual appearance. It deals with how the font rasterizing engine of the graphics subsystem renders glyphs. This is described by their type (e.g. Helvetica), size (e.g. 12 pt), style (e.g. slanted), color (e.g. black).

Since text is read in a sequential way (e.g. in western languages from left to right then top-down), it defines a flow of glyphs. The flow is based on lines. The most basic flow feature is kerning, that is, the distance of glyphs in a line, and word distance, the width of spaces between words in a line. The next notion is the paragraph that is made of a certain number of lines. The notion of paragraphs is based on a maximum width of lines. Within this width, paragraphs can be aligned to the left or to the right, and can also be centered. Paragraphs also feature indentation, which is the position of the first line with respect to the rest of the lines. The (top-down) flow of paragraphs requires separation using horizontal inter-paragraph spaces.

The notion of page contains paragraphs that should be presented as the largest visual unit in the presentation medium. If the presentation medium is traditional, such as the paper, then the page is physical. In case of display devices, there exist virtual pages that do not necessarily physically fit on one presentation device, but reading one page does not require any interrupt for the paging interaction.

Interactive appearance of text is based on text typing and selection. When typing, glyphs appear in the presentation device in a specified flow order. Selection changes the color of the glyph and its background.

4.5.3.3 Activity

The text media-type provides rich activity. Since text is a non-temporal media-type, its storage and exchange does not pose extra requirements. Generic storage and exchange devices support text as it was a byte-stream. For presentation, text has to be rendered on a presentation device according to its appearance. Editing of text is a complex, interactive process. Some editors provide WYSIWYG functionality, some focus on the structure. WYSIWYG editors render content on-line, structure based editors let the user concentrate on the structure, the rendered result cannot destruct her from focusing on the inherent structure of text. Text querying can be done using full text search, that is, querying at character level disregarding higher structure, or structural query, where the structure of text is targeted at. Mixed methods can also be applied, where the characters as well as the structure of text is accounted for.
4.5.3.4 Device Content

The atomic element of text is the character. Text is written language, therefore, we have to be able to represent each character of the written form of each language. Early representations of text did not cover characters of all languages, ASCII, a widely used representation encodes characters on 7 bit, that is, 128 characters can be distinguished. Unicode can represent all characters of all alphabets.

Device content often includes formatting instructions for text. Formatting might be based on appearance (HTML), or on structure (XML, \LaTeX).

4.5.4 Graphics

4.5.4.1 Structure

Graphics being a synthetic media-type is inherently structured. As discussed in the MED domain, the structure of graphics is based on a scene as the highest abstraction, and a graphical primitive as the lowest abstraction. The same notion is used in the UIF domain.

4.5.4.2 Appearance

The appearance of graphics is complex. The appearance of graphics is described by visual appearance. The most important graphics appearances include: color, lighting, shading, material, mapping (bump, texture, environment, shadow). Texture mapping is based on images, therefore, for this, composite media-types are used (see below). An additional appearance is the viewpoint, from where the scene is viewed.

4.5.4.3 Activity

The creation of graphics can be interactive using an editor (MED). Creation comprises the selection, aggregation of primitives, and the definition of their appearance. The storage of graphics must contain its structure and its appearance. In case the appearance contains refers to non-graphics, such as texture, it must be handled separately (see composite media-types later). The transfer of graphics

4.5.5 Animation

Animation is not an independent media-type, it can be thought of as an extension of the text, image and graphics media-types by appearance, activities and resources.

4.5.5.1 Appearance

The temporal appearance is the qualifier for animation, therefore only that is described here. Temporal appearance can be described in a variety of ways, the most common descriptions are trajectory-based or keyframe-based. The trajectory can be defined as functions, or distinct points that need to be connected by a path. Keyframes specify graphics snapshots through which has to be interpolated. If animation complex, the nodes of a scenegraph can be animated in relation to each other, specifying constraints between them.

There is one visual appearance feature that extends that of image and graphics. This is motion-blur. Motion blur is required to make animation look more real.

Interactive appearance is typical for animation, because animation can vividly express temporal behavior as a result of interaction (UIF, MED). A common scenario is triggered by mouse behavior, such as clicking, and move over.
4.5.2 Activities

The creation of animation is based on creating text, graphics or image and extending it with temporal appearance. The specification of the temporal appearance is usually done interactively using an editor (CPC, MPP). An animation can either be stored as one unit, or by separating temporal and non-temporal parts. The image and graphics elements are stored as already described, the storage of the temporal appearance is based on its type. The presentation of animation is rendering. The transfer of animation can be done off-line and during rendering. The latter case is used, when the temporal behavior is not known in advance, but then streaming must be used (CMS). Querying animation is structural query, different elements and relationships of the scenegraph can be queried for.

4.5.3 Resources

The extra resource animation might need is streaming, because the temporal appearance might be generated on-line by a remote device. However, a generic notion of streaming is necessary including local streaming for device location flexibility.

4.5.6 Video

4.5.6.1 Structure

The inherent structure of the video media-type is a temporal sequence of frames. There is a hierarchic structure that expresses the semantic of video. The top of the hierarchy is the video clip, it is split into scenes, which are again split into cuts. Cuts contain a certain amount of frames. There exist special key-frames that represent certain features of a frame sequence.

4.5.6.2 Appearance

The visual appearance of video is inherited from image appearance. Temporal appearance is the speed of the video. A more detailed temporal appearance specifies the qualities of video in terms of delay and jitter.

4.5.6.3 Activity

The creation of video is called capturing. The storage of video happens in streaming mode, in conjunction with storage. Presentation also facilitates streaming (CMS). Editing of the video is based on its structure. Video processing is comprised by compression, cut-detection etc. described by CBR and CPC domains. Querying can be based on content and structure. Content queries relate to key-frames, which are handled as images. Structural queries are based on the structure of video.

4.5.7 Sound

4.5.7.1 Structure

Sound is a temporal media-type. Its structure is similar to video, with the sound-clip as the top of the hierarchy, and with frames as the lowest level (CBR, CPC).

4.5.7.2 Appearance

Sound appearance is expressed as auditory appearance. The basic notion of auditory appearance is volume and speed. A more detailed description of appearance contains the auditory quality of sound in terms of delay and jitter.
4.5.7.3 Activities

The creation of sound is done through capturing. Since sound is a temporal media-type, its capturing and storage is done by streaming. Interactive or batched processing of sound is editing, it is described in the CPC domain. The presentation of sound is also done through streaming. Querying can be done on content (CBR) and on the structure of sound.

4.5.8 Music

4.5.8.1 Structure

The structure of music is built on the notions of the musical score. The musical score contains notes. Notes together form chords. Notes as well as chords have pitch (frequency) and duration. Frequency can be both absolute of expressed in semitones. Duration can also be expressed either in time or in beats. Several channels can contain scores that might be played by different musical instruments.

4.5.8.2 Appearance

Appearance of music contains several facets. Appearance contains the global music attribute speed. It is defined as beats per minute (BPM). Instrument is another attribute that defines how to play a particular channel. The description of an instrument can be as simple as specifying an instrument name, such as piano, but can also be as complex as specifying parameters of the instrument.

4.5.8.3 Activities

The creation of music is either capturing a real-time performance or off-line assembly. The music that has been assembled or captured can be manipulated. Manipulation of the structure comprises changing the duration and pitch of each note, transposing group of notes, reordering notes etc. Playing can also be performed on music.

4.5.8.4 Device Content

Musical device content is the sequence of events that describe music. A common standard is MIDI [79]. MIDI events can be streamed and stored. Typical events comprise status and data events. Status events specify channel, data events contain note on, note off, pitchbend, modulation etc. Instrument description can also be considered device content, these are either parameters of a generic sound generator or samples (single or multiple).

4.5.8.5 Resources

The most common musical device is the MIDI device that is capable of sending and receiving MIDI message streams.

4.5.9 Speech

4.5.9.1 Structure

The structure of speech is the sequence of phonemes that make up the utterance. Phonemes together form words and sentences. Coarser grain structures that can be created by the combination of sentences (e.g. parts) do not have a direct impact on speech.
4.5.9.2 Appearance

High level appearance is the global speed of speech, and the type of speaker (male, female, child, old). More complex appearance descriptions define the prosody using pitch, volume and length for each phoneme.

4.5.9.3 Activities

The creation of speech can be the result of capturing and recognition, such that sound is captured and recognized as a sequence of phonemes. Creation can also be offline by assembling phonemes to form a sequence.

4.5.9.4 Device Content

Device content must be capable of describing how speech is generated. This description depends on the generator device. It is either a high-level description containing phonemes and phoneme combinations along with their parameters, or a low-level description containing the combination of formants for voiced and noise for unvoiced sounds.

4.5.9.5 Resources

Speech devices can be twofold: formant synthesizers that produce phonemes by superimposing formants, or phoneme synthesizers that work on the level of phonemes.

4.6 Composite Media-Types

The media-types described in chapter 4 were monomedia-types. In order to keep abstractions transparent, we introduce composite media-types (also called multimedia-types) by composing existing media-types and add features to the composite to express the composition semantics. With other words, we build composite media-types using monomedia-types, rather than altering them. The differentiation and the typing of composite media-types is based on the semantics of the composite. In the followings, the most fundamental composite media-types are described.

4.6.1 Temporal Composition

Two media-types are composed to a temporal composite media-type if there is temporal relationship between them. From the MPP domain, there exist two basic relationships: sequential and parallel.

The structure of a temporal composite is empty, because there is no common structure involved into temporal composition.

The temporal appearance of a composite contains the description of the composition. Temporal composition can be specified at two levels. The high-level composition defines sequential and parallel temporal layout and relative delay of media-types to each other. The low-level composition describes the relationship between low-level elements of media-types (e.g. video and audio frames).

4.6.2 Spatial Composition

This composition is used to express the spatial relationship between media-types. The spatial relationship is expressed as a direction, distance etc. (MPP) of the visual appearance.
4.7. MEDIA-TYPE CONVERSIONS

4.6.3 Activity Composition

This composition is used to achieve composite activities that trigger several activities belonging to different media-types. This composition need not necessarily imply a temporal or spatial composition, but they can be used together. Activity composition may act either as a sequential workflow, or as parallel execution.

4.6.4 Collection Composition

The collection composition is the composition that is subject to queries that cannot be accomplished within one instance of a media-type. The query composition contains structure, which represents the feature space of the query. In case of declarative queries, this structure will be traversed. In case of comparative (similarity) queries, the source media-type of the collection is located and the references of its feature neighborhood are returned.

The collection composition does not contain appearance, but does contain activities concerning the query.

4.6.5 Hypermedia Composition

This kind of composition is used to relate media-types, or different instances of the same media-type to each other. It composes the media-types, but using its own structure, it contains the hypermedia references independently from the media-types. Its appearance describes how anchors appear within the media-type. Its activity describes which functionality of hypermedia can be used (MPP domain).

Hypermedia is often associated with interaction. Hypermedia composition in this work however is a generic notion, it is used as the generalization of annotations (DMW). For instance, a hypermedia composition can connect a video and a text media-type by referring to the text as describing the video (or a part of its structure).

4.7 Media-Type Conversions

Media-type conversions are needed for two reasons. The one reason is that we want to provide a certain appearance of activity on a media-type that is not available. The other reason is that we want to express the same information in a different media-type in order to exploit a different perception medium.

The first case comprises media-type conversions, such as graphics->image, text->graphics and animation->video, video->image conversion, and audio media-type conversions, such as music->sound, speech->sound.

The second case comprises text->speech, speech->text, image->text, video->animation, image->graphics.
Chapter 5

Designing Multimedia Components

«Mindenki másképp egyforma.»
/Mérő László/

5.1 The Goal of This Chapter

The design of multimedia components is a complex task. Components are a concept of deployment, however, in order to deploy components, they first need to be designed and implemented. This chapter describes the design of an object model. After having described the object model, components are defined based on the model.

5.2 Model Basics

In this chapter we recall the basic common abstractions described in chapter 4, and provide a model for media-types, including simple and compound media-type, along with their structure, appearance, activities, device content and resources.

This model will be an abstract model, that is, a model, that is not meant for implementation and instantiation. In order to implement and instantiate the model, several other issues need to be tackled that are not covered in this chapter, but in chapter 6.

For the notation of the model, the Unified Modeling Language, UML [39] is used. This is a standard notation adopted by the Object Management Group, it is flexible and builds on the object-oriented paradigm.

The model is called COMMOTION (Comprehensive Object Model for Multimedia Information Management), an overview is provided in figure 5.1. A detailed description of the model is discussed in the remainder of this chapter.

The model presented here is abstract in many aspects: (1) it is not bound to any programming language or technology, (2) it does not distinguish strictly between a type and its implementation within the model, and (3) it does not incorporate object localization in a distributed case. The elements of the model (which can be both types and classes) are named “Objects”.

Based on chapter [4.4.4], the model shall express two essential notions. The one is the notion of conceptual media, the other is the resource of conceptual media. The model shall express structure, appearance and activities as part of conceptual media, and devices and services as part of resource.

93
5.2.1 Commotion Object (CmtO)

This is the base object of the COMMOTION object model, almost all objects inherit from it. CmtO allows for composition, it realizes the composite pattern. Objects that inherit from CmtO are all able to use the functionality provided by CmtO hierarchies. These include the creation, insertion, deletion of objects. CmtO hierarchies can be explored in a variety of ways, special navigations and traversals are added in specialized objects.

The exceptions from inheritance are the Device Content Object, Service Objects, since they do not need the functionality provided by CmtO.

5.2.2 Media Object (MO)

The Media Object models conceptual media in COMMOTION. Media Object is aggregated from Data Objects, Appearance Objects, Activity Objects, Device Content Objects. Media Object can be thought of as a specialized bottleneck interface to a media-type. It specializes and restricts the management of the behavior and the state of media-types. This is intentional, because in this way, reuse through genericity of the underlying objects as well as semantic enforcement by typing can be assured.

A Media Object can represent a single perception medium being associated to a single media-type. Then, it is called a MonoMedia Object. If the MO is a MultiMedia Object, besides composing its own DOs, AppOs, ActOs, DCOs, it also composes other MOs, and also DOs, AppOs, ActOs, DCOs of other MOs. Despite most of the counterparts of MOs are also hierarchic, they cannot aggregate objects that belong to other MOs, only MOs can do this. The reason for this is the restriction of dependencies between different kind of CmtOs. An MO can depend on all other objects including other MOs, but no object except the MO can depend on other objects, except when they belong to the same MonoMedia Object.
Media Objects also contain event management. A Media Object can send and receive events to and from other Media Objects, respectively. The receiver MO can register itself with the sender directly, or through an event channel. The event sender or the event channel notify the receiver if the event is sent.

5.2.3 Data Object (DO)

This object corresponds to the structure of conceptual media. Data Objects are hierarchical, the top of the hierarchy is attached to the Media Object. The hierarchy management is inherited from CmtO. DOs take full advantage of it. The hierarchy is always single-rooted what concerns its attach point to a Media Object, however, different MO attach points in a DO hierarchy can have the meaning of multiple roots.

5.2.3.1 Structural Elements

The DO structure is composed of DOs that are the structural elements of the hierarchy. DOs have a type that reflects the structural element they represent. Structural elements depend on the media-type, but not on any device content format. In the following we list structural primitives and elements typical for different media-types. The composition of structural elements underly certain rules. Composition rules define which elements can be contained in an other, and also, in which order elements are composed.

5.2.3.2 Granularity of the Structure

One of the most complex problems COMMOTION solves is the seamlessness of the object model structure. As we saw in the domain analysis, differences in concept granularity appeared as a main bottleneck in combining concepts. In COMMOTION, the media-type as the main common and generic concept is not opaque. This means that it uses abstraction instead of encapsulation for hiding its counterparts, such as structure, appearance etc.

Since media-type stands for a single perception medium, every structure that requires more than a single perception medium is an MO composition, however, compositions within a single perception medium can also be DO compositions. Avoiding heterogeneity of the structure might be advantageous in cases, where MO compositions are of the same media-type. In this case, the MOs will retain their structure, but they can share this structure with a compound MO. This is shown in figure 5.2.

Changing the granularity can be accomplished dynamically in several ways. A leaf DO, in case it is not already the structural primitive for the media-type, can be refined to contain other DOs, making it a non-leaf.
Another way of refinement is the creation of a new hierarchy level between two existing ones. Abstraction, on the other hand means creating and attaching new MOs to a certain point in the DO hierarchy.

5.2.3.3 Attributes

The flexibility required from DOs is solved using different attributes. The attributes of a Data Object can be three kinds: internal, external and referential. Internal attributes are inherently part of the Data Object. External attributes are extracted from Device Content Objects and appear as redundant DO attributes. A referential attribute contains a reference to a DCO, plus an offset and a length that identifies a portion of a DCO. DOs can possess a reference to a DCO only if the identified portion contiguously fit into one DCO.

All three kinds of attributes can be either static meaning that they have been defined as part of the DOs type, or dynamic meaning that a separate property object manages them dynamically. Static attributes require the DO to be subtyped, whereas dynamic attribute management is applied on generic DOs.

The combination of internal, external and referential with static and dynamic would give eight different settings for DOs, but several considerations like performance, flexibility and weight of DOs constrain the combinations.

5.2.3.4 Typical Structure Kinds

We differentiate three basic kinds of structures reflecting the performance, flexibility and weight of nodes discussed above. Real DO structures might be either pure structures or a combinations of these.

Referential structures contain nodes with referential attributes as static attributes to DCOs. Nodes also contain parent/child management and a dynamic role that identifies the semantics of the reference the DO refers to. This kind of structure is used to obtain a reference to a DCO such that content can be directly accessed. This is an efficient structure for coarse-grain activities, because minimal effort is required for getting direct access on a contiguous piece of content. This structure however is not suitable for frequent insertions and deletions of fine-grain structural elements, which is typical for interactive editing. A good example of referential structures is the logical structure of a text document. Referential structures can be combined if the DCO portion they refer to is not contiguous. For instance, selecting an image region would create a region node containing a set of line segments that define the region.

Shadow structures contain nodes that extract content and redundantly contain it as external attributes. The reason for this is the frequent manipulation of fine-grain objects. For efficiency, attributes need to be static, which also types the DO. These structures are usually flat, having only specialized leaf nodes. The type of the DO is similar to the role of nodes in the referential structure. A good example of shadow structures is the physical structure of a text document, containing columns, rows, characters.

Index structures contain nodes that allow for quick access to a DO based on DO or DCO features. Index structures contain a single feature or multiple features and describe a collection of DOs or DCOs. Index nodes either refer to other index nodes (non-leaf) or the DOs (leaf) representing the node or the DCO to be indexed.

5.2.3.5 Structure Behavior

There is basic behavior provided with the structure. This behavior comprises insertion, deletion, traversal and navigation. Insertion and deletion is explicit in case of generic DO structures, however indices provide it implicitly. Traversal is used to visit all nodes of the structure in a given order, navigation is used to find a particular node based on rules.
5.2.4 Device Object (DevO)

The Device Object represents the notion of device in COMMOTION. Devices encapsulate functionality they can accomplish on information in a certain representation. DevOs have input and output formats to define representation of streaming information. The input and output formats define which information representation the device can receive and produce, respectively.

Formats can be hierarchic, lower level formats in the hierarchy are compatible with higher level formats. The lowest level of format is generic bitstream. The highest level can be as complex as an MPEG-2 bitstream. Only DevOs with a compatible format can be connected to each other through a stream.

There are two basic kind of devices: Media Device Objects (MDevO) and Connection Device Objects (CDevO). MDevOs either store, create, filter or present DCOs. CDevOs move CDOs between MDevOs. The format descriptions relate to MDevOs. CDevOs do not need to know about the formats of MDevOs.

Devices expose their capabilities and their behavior through device properties. Capabilities are read-only properties, they cannot be directly changed. The most important capability is set of formats supported by the device.

Devices expose their behavior for configuration through device properties. Some properties are generic for all devices, others are specific to a particular device. Generic properties pertain to time-dependent behavior, such as buffer management.

5.2.5 Device Content Object (DCO)

The DCO is the content of a DevO. Only MDevOs are associated DCOs. The DCO has a format which needs to be compatible with that of the device. It is important to note that the only notion that identifies DCOs are their format, since at this level, there is no notion of perception, appearance, activities etc. represented by media-types.

Depending on the format, DCOs can be divided into logical content units (LCU). Theoretically, these LCUs may also make up a hierarchy, but since they are handled inside the device (and there are no device hierarchies), the description of the format defines which LCUs the device can receive and send.

Formats can fall into several typical categories. The simplest format is the raw or device dependent format. Raw formats are usually produced by capturing devices, and consumed by presentation devices. This format is a sequence of one kind of LCU. This format needs extra information describing the content (see Appearance Object).

Another category contains two LCUs: header and payload. The header describes the content in the payload. The third category, tagged or chunk format contains tags/chunks consecutively, with tag/chunk identification, length and tag/chunk content. Tags/chunks might be hierarchic, any level can be assigned to LCUs, depending on the device.

5.2.6 Stream Object (SO)

The Stream Object connects several Device Objects. An SO deals with connection management, configuration, control, topology and the synchronization of the stream. The connection is a mechanism through which the one device sends information in a given format to the other device. The connection kind depends on the locality of the devices, remote devices require different connections than colocated ones.

Connection management deals with device connections and stream topology. Device connections are done through connecting MDevOs having compatible formats through CDevOs. Compatible formats can be matched based on device capabilities. Connection management initializes devices (lookup, instantiation etc.) and tells them which other devices they receive the stream from, and which other device they need to send it to.
CHAPTER 5. DESIGNING MULTIMEDIA COMPONENTS

Configuration management deals with how a certain device in the stream carries out the task of stream creation, filtering, presentation etc. The peculiarities of each device are configured through device properties that describe device behavior. Streams can also reconfigure devices during operation, in a dynamic way.

Stream control is responsible for starting and stopping streams. Streams can operate both in a continuous and in a one-shot mode. The continuous mode operates in a periodic manner, depending on the synchronization policies of the stream. The one-shot mode corresponds to one period. What needs to be done in one period, depends on the format.

In many cases, simple streams cannot carry device content, there is a need for more complex stream topologies, such as parallel, split or merged streams. If streams are parallel, it means that there are streams that can be controlled together by a compound stream. Parallel streams can be merged into one stream, the logic for merging resides inside an MDevO having several inputs and one output.

Streams can also be split. If a stream is split, it means that there is a device that has one input stream and more than one output streams. If there is a CDevO with this characteristics, it means that the output streams are multiplications of the input stream: this is called multicast. If a MDevO splits streams, the output stream was a compound stream, this is the inverse of merging.

Synchronization can be broken down to two parts: intra-stream and inter-stream synchronization. Intra-stream synchronization pertains to a single stream, inter-stream synchronization pertains to several streams. A single stream can be synchronized to a time-base.

In case of multiple streams, every single stream can be separately synchronized to a time-base, but in most cases, inter-stream synchronization is necessary. This means that such that these streams are synchronized to each other.

5.2.7 Appearance Object (AppO)

The Appearance Object describes how a Media Object appears to the user, this applies both to MonoMedia and MultiMedia Objects. The difference is that while an AppO for a MonoMedia Object describes the appearance of that MO, an AppO for MultiMedia Objects define only the part of the appearance that refers to how the MOs composed into the MultiMedia Object appear together. As described in a previous chapter, appearance has several facets. The description of appearance is based on the structure.

AppOs are hierarchically composed, since the definition of appearance is aligned with the structure. This means that an AppO always corresponds to a DO structural element. The description of the appearance is compact: an AppO defined in the appearance hierarchy for a particular DO is applied to every instance of the same DO towards to leaves of the DO structure, unless there is another AppO that redefines it.

AppO features fall into two categories: intrinsic and extrinsic. Intrinsic features are defined in the creation phase of the structure (or content), whereas extrinsic features are computed or set on demand in a given context. An example is the intrinsic size of an image as it has been captured, and the extrinsic size of the frame it has to fit while presenting it in a window on the screen.

There are three specializations of AppO: visual appearance, audio appearance, temporal appearance and interactive appearance. In the following, we elaborate on these.

5.2.7.1 Visual Appearance

Visual appearance (VAppO) describes general attributes of a media-type that concern its visibility. VAppO defines a frame, into which the visible part of the DO must fit. Frames can be as big as representing a page, and as small as representing a character. Frames either have predefined or calculated size. Since frames are hierarchically composed, the modification of the framesize may impact both the size and/or layout of its parent and child frames.
5.2. MODEL BASICS

VAppO is further specialized into media-type specific appearances. The appearance sub-types that correspond to the basic media-types are TextAppO, ImageAppO, VideoAppO, AnimationAppO and GraphicsAppO. Media-type specific appearances have impact on generic visual appearance and vice versa (e.g. fontsize may change framesize, and other way round).

5.2.7.2 Audio Appearance

Audio appearance (AAppO) describes general attributes of a media-type that concern its audibility. AAppO describes volume that applies to all audio media-types. AAppO is further specialized into media-type specific appearances. The appearance sub-types that correspond to the basic media-types are SpeechAppO, SoundAppO and MusicAppO.

5.2.7.3 Temporal Appearance

Temporal appearance (TAppO) describes general attributes of a media-type that concern its temporal behavior. We distinguish TAppO for single and compound media-types. For single media-types, TAppO, the generic feature is speed. For composite media-types, TAppO includes the parallel and sequential synchronization of MOs that belong to the composition.

5.2.7.4 Interactive Appearance

Interactive appearance (IAppO) describes general attributes of a media-type that concern its reaction to interaction. Interactive events, such as keyboard and mouse events that are sent to a certain MO are dispatched to the IAppO. The IAppO changes other appearances.

5.2.8 Activity Object (ActO)

Functionality in COMMOTION is split into two notions. We distinguish implicit functionality and explicit functionality. Implicit functionality is present inside objects. This functionality is rather generic, in the sense that many objects share it (e.g. tree traversing in CntO), and specific in the sense that they are inherent part of an objects behavior.

Explicit functionality is factored out to be embodied in separate objects. This functionality is generic in the sense that they are not inherent part of any object as its behavior, therefore any other object can use it, and specific in the sense that objects must explicitly use it, it does not automatically "come" with the object. There are many advantages of having separate objects for activities. Two trivial advantages are the simple provision of asynchronous behavior (queuing, pooling), and the easier separation for componentizing.

The classification of Activity Objects is not a straightforward task, since there are many criteria, and different classifications impact the model differently. Main classifications of activities can be done according to management functionality, media-types and environment. Since none of these can be a major classification, these features are contained in the ActO as attributes.

In the following, we do not describe all activities in detail, because they are either described at other places, or their description is obvious.

5.2.8.1 Presentation: Rendering and Interaction

In order for the user to view, browse, inspect media-types, instances of COMMOTION need to be rendered. Moreover, all functionality of the media-types, also complex workflows should be able to be triggered through interaction with the media-type. We expect the media-type to render itself. This corresponds to the Media Object calling its own presentation activity. Rendering of COMMOTION is a complex task, because of the following reasons:
the COMMOTION model to be rendered is usually compound

- the rendered result must reveal its structure, appearance and activities in an intuitive way

- direct manipulation of these features needs to be enabled

An obvious solution to this problem is to provide different views (renderings) to different features of the model. There would exist a view to the structure, appearance, activities etc. The problem of this solution is that all views would need to be described, such that the structure would have structure, appearance etc. in each particular view. A better solution is to factor these views as other Media Objects, creating an MVC pattern.

Rendering can be accomplished in two ways. In the one way, rendering means the provision of device content on a certain device that encapsulates the representation medium for a certain media-type. Since the presentation device may have any format and complexity, the rendering activity shall incorporate these device peculiarities. In the other way, there is no device explicitly assigned to presentation, therefore the rendering activity implicitly contains the device.

The rendering process has two parts. The first one prepares the Media Object instance structure for rendering, the other maps the prepared structure to device content. The preparation phase refines or abstracts the Media Object structure to fit a certain output device. Refinement means the creation of finer-grain structural elements and appearance descriptions that are needed for the mapping, whereas abstraction means the deleting of structural elements that cannot be mapped to device content of the target device. After preparation, the mapping process creates device content that is streamed to the device.

5.2.8.2 Externalizing and Internalizing DCOs

Usually, CmtOs are managed by the activities present in MOs, as opposed to DCOs that are managed by DevOs. There are few exceptions to this rule. One major exception is an activity pair, DCO externalizing and internalizing. An example of DCO internalization was already mentioned in the description of the presentation activity. Here we tackle this activity in general.

Externalizing a DCO means its conversion to DOs and AppOs. The most obvious purpose of externalization, as we have seen in the description of the presentation activity, is to serialize DOs and AppOs in order to make them available for a certain device for the purposes of streaming and processing. The need for this can be triggered by necessity: functionality is only available for DCOs on a device, e.g. decompression, and convenience: it is more performant to traverse the structure once and then apply several operations on the stream, then to traverse it each time an operation needs to be applied.

Internalizing a DCO means its creation from DOs and AppOs; it is the inverse process of externalizing. Obvious purposes of internalization are the same as externalization. Internalization is a necessity if a certain functionality is only available as activity. It is a convenience if traversing and navigation in DO and AppO structures is easier than handling streams.

The ActO that accomplishes internalizing and externalizing has to be aware of DCOs as well as DOs and AppOs. Depending on the format of the DCO and the media-type of DOs and AppOs, the results of these activities are different:

- extraction: kind of externalization, where LCUs of the DCO are converted to attributes and/or properties of the DO and AppO

- anchoring: kind of externalization, where structural elements are created and references from the structure to the DCO are set up

- serializing: kind of internalization, where DO and AppO attributes and properties are converted to a single DCO
Serializing and extraction can be used as import/export functionality in general. If a serialized format, like XML needs to be imported to COMMOTION, it is handled as a DCO, and an activity extracts CmtOs from it. On the other hand, export can be based on serializing CmtOs into DCOshaving formats like HTML.

5.3 Collaborations in COMMOTION

This chapter elaborates on the most characteristic and fundamental collaborations between COMMOTION objects.

5.3.1 Activities

In COMMOTION, ActOs are mainly used in two collaboration patterns: command and visitor. In the command collaboration (named after similarity to the “command” Gamma pattern), activities are accomplished on COMMOTION objects in an order defined by the workflow of commands. The workflow is responsible for triggering the execution and passing the target from command to command.

The visitor collaboration (named after the “visitor” Gamma pattern), the ActO acts as a callback that gets called when traversing or navigating in an object structure (typically DO, AppO or MO).

5.3.2 Embracing Model-View-Controller

COMMOTION can accommodate the MVC pattern at two abstraction levels. First, if arbitrary models need to be used, the pattern is augmented into a media model being the VC, and a business model being the M of MVC. This pattern augmentation is described in detail in chapter 6.

Second, COMMOTION itself can provide a model for accommodating the state of the view and/or the controller. The restriction is that the model must also be a COMMOTION object or structure. In this chapter we only deal with single-level MVC. The M(odel) is any COMMOTION object attached to MO that is a V(iew). The C(ontroller) is another MO attached to the view. Figure 5.3 depicts the MVC pattern within COMMOTION.

5.3.3 Hypermedia

The part describes how hypermedia appearance and functionality can be provided by COMMOTION. Hypermedia is handled in two parts: the logical model (links, anchors) and their appearance and interaction. The
The basic problem is that in COMMOTION a DO as the basic structural element cannot refer to another arbitrary DO element, because on the one hand it might be of another media-type, on the other hand, a bidirectional relationship between two DOs breaks the hierarchy rules. Having said that, a relationship that expressed hypermedia among arbitrary DOs would not prove as advantageous, because then hypermedia would not be distinguishable and separable from the original DO structure, moreover, the appearance of this extra feature could not be easily expressible, either.

The pattern proposed is more flexible. We define an additional Media Object representing the hypermedia link. Its structure consists of two or more DOs that are the anchors of the link. Its appearance describes how the anchors shall appear as a superposition to the MO's appearance and also how the links shall react to user interaction. Figure 5.4 shows the pattern.

5.4 The Architecture: Distributing the Model

An abstract elaboration on the architecture is needed, because we cannot describe components without the architecture. It is an abstract handing, the implementation of this abstract architecture is described in the next chapter.

The component architecture is based on the definition of components. Before we can define components, we have to deal with where objects are in the architecture and how they are supposed to be accessed.

For the definition of components, it is sufficient to give an initial abstract description of the architecture: The COMMOTION architecture is a seamless collaboration of COMMOTION objects that are distributed in an enterprise environment.

5.4.1 Object Location

There exists three kinds of object location in the COMMOTION architecture. We distinguish objects with strong, transparent, migratable and flexible location. Objects with strong location are bound to a certain location in a distributed environment. In COMMOTION, these are the Device Objects. The usage of objects with transparent location is independent of a certain location, it can be transparently handled. These are Service Objects in COMMOTION. Objects with migratable location do have a locality, but it can be migrated. In COMMOTION these are Device Content Objects. Objects with flexible location can either be handled as transparent or migratable, depending on the context.

The definition of same and different locality is not given in this chapter, since it is not required for the definition of the components.
5.4.2 Object Access

Object access means for an accessor to obtain a reference to an object and to send messages through this reference to the object. There are two dereferencing, four executive, and four complex kinds of object access patterns in COMMOTION. The dereferencing kinds refer to how an object reference turns into an object, the executive kinds refer to how to send a message to a certain object, the complex kinds refer to several objects that need to be accessed together consistently, combining different dereferencing and executive access kinds. The dereferencing and executive access patterns are orthogonal to each other.

5.4.2.1 Dereferencing Object Access Kinds

There are two dereferencing object access kinds, explicit and implicit. Explicit dereferencing is based on obtaining a generic object reference that must be explicitly dereferenced to get the object. Implicit dereferencing is based on automatic dereferencing at the time the object is sent a message through the reference.

5.4.2.2 Executing Object Access Kinds

The basic kinds of executing object access are direct, by-value, by-reference and by-streaming. Direct access means that the accessor and the accessed objects are collocated (have the same locality). If the object to be accessed in not collocated with the accessor, we refer to the object to be remote.

By-reference access means that the accessor and the accessed objects are not collocated (have different locality), therefore the object is accessed by the creation of a surrogate (proxy) that is collocated with the accessor, such that the surrogate dispatches the message to the remote object and also receives the result.

By-value means that the remote object is migrated to the accessor that can access the object directly. The difference to direct access is that the migrated object is a copy\(^1\) of the remote object. Object access by value can by differentiated upon the fact whether the whole object or only its state is copied.

By-streaming is a complex access kind. It is similar to access by-value in the sense that the object is migrated. However, when the object is streamed, there exists a temporal notion. There is no copy made, but the same object is “spread” around the distributed environment, different parts of the same object can be found at different localities at a certain snapshot in time.

In COMMOTION, by-reference and by-value refer to CmtO, by-streaming refers to DCO.

5.4.2.3 Complex Object Access Patterns

One of the main achievements of COMMOTION is the provision of complex access patterns. Since DOs contain the structure of the DCO, it is required synchronize the access of the one to the other. If several access kinds are combined and synchronized to each other, we talk about complex access patterns.

The complex patterns assume that DOs are accessed by-reference or by-value, whereas DCOs are accessed by-streaming. The two major complex access patterns are structure-driven and stream-driven. The structure-driven access pattern is based on the access of media-types through their structure, that is, the DO. This means that object access is driven by the structure (master), and DCO access is synchronized to it accordingly (slave). The streaming pattern is the inverse of this: object access is driven by the stream (master) and DO access is synchronized to it accordingly (slave).

The structure-driven access is used when object access is structure based and random. When a DO is accessed, the stream is in one-shot mode and the corresponding DCO is streamed to the location the DO accessor is. Stream policies can be used to define upon which activity or event the corresponding DCO shall be streamed.

\(^1\)The identity of this copy depends on many other circumstances, such as the transactional context which is discussed later, because it is not required in this context.
The streaming pattern is driven by the stream. The stream can be started, paused, stopped and positioned. DOs and AppOs are updated to the required depth based on LCUs in the active stream. Since a regular update process might require too much effort at a fine-grain level, the update can be broken down to the constant and peeking updates. Constant update always updates DOs and AppOs on a periodic basis, whereas peeking updates only update objects when they are peeked into.

5.5 The Architecture: Services

Services can be subdivided into generic and media services. Important generic services that need to support the model are persistence, transaction, naming, query and event service. Earlier in this work, we also mentioned streaming as a service in a general context. Streaming is explicitly present in the model as the Stream Object rather than being an explicit service. It relies on two issues: standard services that are being described in this chapter (e.g. persistent streams, querying on streams), and devices (MDevOs and CDevOs).

In previous chapters, we already elaborated on services that need to be present for media-types. Here, we elaborate on how these services are used in terms of the model.

5.5.1 Persistence

Persistence is the most fundamental service that is provided to COMMOTION. Persistence makes objects outlive the context in which they have been created. Persistence means saving the state of the object such that it can be restored in an other context. There are some assumptions made for the persistence service:

☐ all COMMOTION objects are subject to persistence, except DCOs: Although persistence can also be applied to DCOs, their management is explicit and taken care of by devices they reside on and streamed by.

☐ the persistence service only manages object state persistence, not behavior: Although object behavior, more precisely, its implementation can also be made persistent, the persistence service does not deal with it, behavior persistence is a component issue (see later).

☐ we consider object-based persistence, but neither session nor file persistence: Persistence is a broad field including different persistence mechanisms. Session persistence and file persistence are not suitable for COMMOTION in general. Session persistence is not suitable for controlling persistence granularity and sharing objects, because it operates memory-image based. File persistence creates opaque representations omitting identity, and uses the notion of files. Object-based persistence retains the identity of objects, it can handle persistence at object level, therefore, only this kind is used.

A central notion of object persistence is to identify objects and instances are persistent (not transient), when to store instances and when to retrieve them.

5.5.1.1 Persistence Indication for Objects and Their Instances

The persistence service must know about the state of the objects it should make persistent, since a generic run-time inquiry on an attribute basis would be too costly, moreover a meta-level interface would needed to be used. The state is known to the persistence service by registering types with their persistent state explicitly.

The creation of objects of a particular registered type can automatically involve its persistence. This mechanism would be rigid, since it does not allow for the coexistence of transient and persistent versions of the same type.
5.5. THE ARCHITECTURE: SERVICES

A better solution is to decide at the object creation phase by distinguishing a transient and a persistent factory. Only objects created by the persistent factory will be stored. These two factories can be merged into one generic factory of a type; a parameter can indicate persistence that is processed within the factory implementation.

Another solution is persistence by reachability. In case the instance is reachable from a root and its type is registered with the service, it will be stored. All roots of reachability must be explicitly registered with the service.

Another solution is explicit persistence. In this case, a store method must be called for each instance that needs to be stored. The method can be part of the object or that of a service object. In the latter case the instance reference must be passed along with the method call to the service object.

There exists a persistence refinement orthogonal to the persistence indication. Persistent state within an object can be a subset of its state. The persistence service does not know about transient state; it considers the persistent object to be complete based on the state it is aware of.

COMMOTION features some peculiarities that guide the decision for persistence indication. All COMMOTION objects inherited from CmiO are subject to persistence. Persistence by reachability, although being an elegant way of indicating persistence, cannot be used, because persistent parts of COMMOTION object hierarchies are extended with transient objects of the same type. Automatically persistent types cannot be used for the same reason. A combination of factory-based and explicit persistence seems to be a good solution, because it combines performance (factories) with flexibility (explicit). Moreover, persistence by reachability can be provided on top of these mechanisms.

5.5.1.2 Storage Time Indication for Instances

Storage time indication can be either immediate or transactional. Immediate storage means that every single object can issue a command to be stored immediately. Transactional storage means that the notion of transaction provides a context for storage. Once the context is set (begin transaction), every object creation is related to that context, such that when the context is closed (commit transaction), tentative changes become permanent.

The transactional context has not only advantages what concerns performance (store together and controlled rather than individually and uncontrolled), but also what concerns semantics: object storage is more structured by the explicit notion of context.

5.5.2 Transaction

The transaction service realizes the concept of a transaction in a distributed environment. The transaction is a unit of work that has the following (ACID) characteristics:

- A transaction is atomic: if interrupted by failure, all effects are undone (rolled back).
- A transaction produces consistent results: the effects of a transaction preserve invariant properties.
- A transaction is isolated: its intermediate states are not visible to other transactions. Transactions appear to execute serially, even if they are performed concurrently.
- A transaction is durable: the effects of a completed transaction are permanent; they are never lost.

Starting a transaction (being transaction) sets a context. Every operation that is carried out on objects pertains to this context. A transaction can be terminated in two ways: the transaction is either committed or aborted.
(rolled back). When a transaction is committed, all changes made in that context become permanent. When a transaction is aborted, all changes are undone.

COMMOTION is distributed both in terms of object locations and object usage meaning that objects in general can have an arbitrary location in an environment, and they also can be accessed from arbitrary locations. In order to synchronize concurrent and cooperative object accesses from multiple sources targeting multiple locations, COMMOTION makes use of a distributed transaction service.

The transactional behavior and therefore the usage of the transaction service heavily depends on what the particular objects represent and which operations are accomplished on them. More to the transaction service is handled in the chapter discussing containers.

5.5.3 Naming

Naming service is a service to locate "root" objects in a distributed environment, from which other objects can be accessed. Naming associates an arbitrary name with a symbol specific to the environment to gain a reference to the object. Naming is usually associated to persistent objects and services. Naming is often hierarchic, namespaces are used for navigation in the hierarchy.

The naming service in COMMOTION can be used to locate different kinds of instances. Examples include:

- **factories**: objects that are used to create other objects
- **documents**: Media Objects that represent logically coherent hierarchies of information
- **layouts**: typical configurations of Appearance Objects that can be applied to gain uniform look and feel
- **workflows**: configurations of several Activity Objects to carry out a complex task
- **streams**: certain stream configurations

5.5.4 Query

5.5.4.1 Navigational query

The query service can apply queries to the state of any object kind in COMMOTION. The query service does not include generic navigational queries, because these can be accomplished on the objects themselves. Often, generic navigational queries are not efficient.

Their inefficiency is due to the sheer quantity of objects in a collection, and the high costs of obtaining a reference and attributes to all objects through which we navigate. A query service obviates the need for obtaining remote references to each object, because it directs the query to a query engine at the location the objects are, which returns the results.

5.5.4.2 Set-based Query

The query service relies on the notion of collections and attributes. Collections is a set of objects that are referred to from a particular object (e.g., set of child objects of a parent objects is a collection). There is a special collection that contains all objects of a certain type. This collection is called the extent or alset (of a type). Attributes can be static, dynamic or computed. Static attributes are described by the type of the objects. Dynamic attributes require more care, their query rather coresponds to a meta-query (e.g. querying for the name and the type of an attribute, rather than its value). Computed attributes are also counted as state for querying.
5.6. COMPONENTIZING COMMOTION

Special navigational queries take advantage of the query service, in that they instruct the query service to perform a collection scan, indexed scan or collection matching. Collection scan is a simple mechanism that sequentially accesses all objects in a collection to find those which are queried for.

Indexed scan uses a selected attribute for that an index is generated. Indices always relate to a collection. This collection can be the child objects of a certain object or the extent of an object type. Based on the attribute, in COMMOTION, we distinguish several indices. Monodimensional index structures contain one attribute, multidimensional queries might contain a large number of indices (several hundred).

Set matching is an additional query mechanism that is based on a join algorithm. The join can be based on pointers (OIDs), and also on pointerwise unrelated collections.

5.5.4.3 Ad-hoc Query

Ad hoc queries are cleartext expressions that are evaluated to get query results. Ad hoc queries are formulated as strings and passed to the query service. The query service will use set-based query mechanisms described above to execute the query. Since the ad-hoc query is not explicit, there is room for optimizations.

5.5.4.4 Pattern Matching

The above query mechanisms are applicable to all objects that inherit from CmtO. For DCOs we apply a different query approach. Pattern matching queries a DCO for the occurrence of a certain pattern. Pattern matching can be combined with navigational and set-based queries. A typical example is navigating among DOs in order to locate a DO that refers to a DCO portion that is the subject of pattern matching.

5.5.5 Event

The event service provides the sending of messages between objects. As opposed to the event service, in case of direct message sending, three conditions always hold:

- the sender always selects the receiver (by reference)
- there is always a single receiver
- there is no message queue

If one or more of these conditions does not hold, that is, the messages are not sent to a particular receiver, there are several receivers, and there needs to be a queue that holds messages if the receiver is currently not available, the event service must be utilized.

The event service features a supplier (also known as publisher), a consumer (also known as subscriber, and an event channel (dispatcher). There are two event handling modes at both sides: push and pull mode. The event supplier pushes a message to an existing message channel, or creates a channel to push to. The event consumer registers with (subscribes to) a certain event channel that will push events to it on availability. If the event consumer does not register with the channel, it can still enquiry the channel for events and pull them on availability.

5.6 Componentizing COMMOTION

In chapter 2, we already provided a fundamental definition of components. In this chapter, we restrict this notion and define a more concrete one (still abstract in the sense of implementation). The components will be defined based on COMMOTION.

COMMOTION components are based on five notions:
1. Data Object, Appearance Object, Activity Object, Stream Object are **media components** that constitute and extend a Media Object.

2. Media Object is a **media framework** (substrate component) that can be extended in several dimensions by Data Object, Appearance Object, Activity Object and Stream Object.

3. Compound (Multi)Media Objects constitute a **media architecture** that is built of Media Object frameworks.

4. Media Container is a localized **execution framework** that provides service and device components in order for the above frameworks and architecture to be executed.

5. Media Container **distributed media architecture** is a coherent set of Media Containers that are coordinated in order to constitute a distributed media environment.

An overview of the five notions can be inspected in figure 5.5.

### 5.6.1 Media Components and Framework

The smallest reusable asset is the Data Object, the Appearance Object, Activity Object and Stream Object. These objects are plugged into a Media Object framework. Pluggability is based on the media-type supported. The media-type is defined by the Media Object, all components’ media-type must match with the Media Object’s media-type.
5.6. COMPONENTIZING COMMOTION

5.6.2 Media Architecture

Several media frameworks constitute a media architecture. The media architecture is local to a certain container. The media container defines how Media Objects are communicating in a certain container.

5.6.3 Media Container and Devices

The container provides an abstraction over the services and devices available at a certain location in a distributed environment in order to execute media components in a media architecture. In particular, containers provide the followings for devices:

- abstraction over configurability and usage of CDevOs
- abstraction over configurability and usage of MDevOs
- seamless use of services

The Media container features the use of CDevOs. CDevOs are used for transferring DCOs between MDevOs, they cannot be considered components from the media framework and architecture point of view. The main reason for this is that they often use core operating system or kernel features (or themselves are realized as such a feature) and therefore are not easily portable to other environments. They are assumed to be provided with the container itself. Of course, CDevOs can also be implemented as components (e.g. kernel modules), but this feature is not exploited by the architecture. MDevOs on the other hand are components of the container architecture, which can be plugged into the container, and several of which can be connected to form a stream.

5.6.4 Media Container and Services

Service embedding into containers serves two main purposes:

- optimizes for local and remote execution
- automates service usage

Services embedded in a container can be optimized for local as well as remote execution, without the user of the service noticing the different between the realizations, therefore, seamless service usage is assured.

There are some services, such as the persistence and the transaction service that are not necessarily used explicitly, objects might want to make use of these services implicitly. The following few chapters discuss the embedding of these services in terms of seamless optimization and automated use.

5.6.4.1 Query Embedding

A generic query service has to be able to query both local and remote, as well as persistent and transient objects. In COMMOTION, this is of particular relevance, because Media Object instances often contain many objects which are transient. The query service has a local implementation that can query objects that are currently instantiated in the container and objects that are stored in the realization of the persistence service locally. The container will assist the query service in finding other containers with query service implementations to query them and summarize the results.

In case of having client and server containers, there might be client containers without any query service implementation: these connect to server containers where the objects reside and execute the query there.

An inverse of the above situation is when the server container has no query service implementation, but the client does. In this case, server objects must be loaded into the client in order to execute the query.
5.6.4.2 Persistence Embedding

Component instances might not want to indicate when to store them. This can happen for two reasons: they either do not know or do not want to know. When objects are created by persistent factories or later indicate their wish to persist they get registered with the persistence service. The persistence service will store registered objects when it is explicitly prompted by the object or implicitly by the transaction service.

Objects are created by factories, which unambiguously define which container makes the object persistent. If objects are used in other containers than created (e.g. accessed by reference), client containers can provide a lightweight version of persistence: caching. Caching might or might not be aligned with the transactional context. If they are, transaction commits shall invalidate the cache. However, there might be cases, where the objects residing in the other container, do not change or their change is not relevant for the client, because they will never be communicated back to their source. In this case, the client cache might never be invalidated, the caching mechanism might even be extended by lightweight persistence.

5.6.4.3 Transaction Embedding

Since transactions are unit of work, the boundaries of the work needs to be specified. Transaction-aware object usage means that the user of objects sets a transaction context before accessing and manipulating objects and thus, carries out every task in that context. Transaction-unaware object usage means that the user of the object does not set any context, therefore the context needs to be set automatically. In the transaction-unaware case, there can be applied several policies, such as method-scoped, object-scoped, session-scoped, time-scoped etc.

transactions.

There is a fundamental difference between transactions for objects local to the container and remote objects. These differences are made seamless by the embedding the transaction service into the container. In case of local objects, there is no need to manage distributed transactions. The transaction service shall recognize the scope of objects and instruct the persistence service accordingly.

When COMMOTION objects are accessed by reference, ACID behavior can be straightforwardly maintained, since objects are not accessed directly. There are some cases when the value of the object is shipped (or the referential access is cached) without the notion of a traditional transaction. These transactions are called optimistic transactions. When the state of this objects is needed to migrated back to its source, the server might refuse the update or provide a different object version.

5.6.4.4 Event Embedding

Events are sent by locating or creating an event channel through the event service. The event service implementation can be in the container, but it can also be dispatched to an independent event service. However, channels that can only be subscribed to by subscribers local to the container does not need to be dispatched to any other event service.

5.6.5 Distributed Media Architecture

The distributed media architecture is comprised by all the media containers in a distributed architecture. We can distinguish three main architecture topologies:

- single container
- client/server container architecture
- non-dedicated distributed container architecture
5.6. COMPONENTIZING COMMOTION

5.6.5.1 Single Container

A minimal setup may contain only one container. In this case the container must support at least persistence service and devices for presentation. This standalone version must interface with independent systems to import information from and export information to. As described earlier, the import/export functionality is provided as activities. The persistence service stores the imported information as COMMOTION. There are devices that present COMMOTION within the container.

5.6.5.2 Client-Server Media Architecture

A minimal distributed setup involves two kinds of containers. The one is a heavier weight container, the server container, and a lighter weight container, the client container. There exist several instances of client containers, but typically only one server container. There might exist several server containers, however, they exist independently, such that clients can either only utilize one of them at a time (session), or provide ad-hoc solutions for concurrent usage without the servers cooperating or even knowing about each other.

The server container contains transaction and persistence service, and optionally other services. It also contains devices to stream content. There can be two different clients: presentational and non-presentational.

Presentational client containers contain devices for presenting the COMMOTION model for the user. They do not have any service implementations of persistence, transaction and query, these services are dispatched to the server. They contain devices for sending and receiving streams, and access COMMOTION from the server either by-reference or by-value. They also contain devices and activities for presenting COMMOTION in a certain environment.

5.6.5.3 Non-dedicated Distributed Media Architecture

This architecture is an arbitrary composition of server and client containers. There are several issues that are raised in such a generic architecture. The major issues are the sharing of COMMOTION between containers. This involves distributed transactions over several containers.

There is no restriction of which services and devices are provided by a container in the generic distributed media architecture, each container can provide any combinations and rely on services provided by other containers.
Chapter 6

Realization and Integration

"There's always time to be late."

6.1 The Goal of This Chapter

The goal of this chapter is to present the realization and integration of multimedia components built on COMMOTION in an enterprise environment. The realization and integration shall reflect the embedding of abstractions of all stakeholders of the system (chapters 4.2.2, 4.3.3, 4.4.6). The reflection of the abstractions lead to a common architectural foundation, and an integration method that comprises a common syntax and semantics for the integration. This chapter describes the architecture and the method in detail.

Commonly, realization and integration are used independently, in different contexts: realization is the process of creating an operational software system using models, specifications and tools, whereas integration refers to enabling the interoperability between two or more existing software systems. The

In accordance with the fundamental definitions given in chapter 2.2, we unify realization and integration, and place it into the context of domains. The terminology of domains has been used in the analysis chapter (chapter3), but in later chapters it was abandoned, because of the seamlessness of modeling. Now, with the integration into heterogeneous environments, we again need to use domain terminology. As defined in 2.2, realization and integration break down to domain mapping and reuse. The distinction between realization and integration is a function of abstraction and granularity levels of the existing system and that of the interoperability.

Firstly, we revisit synergy fundamentals between the business and the multimedia systems.

These levels are discussed in detail in this chapter.

6.2 Synergy Fundamentals Revisited

In this chapter we recall the assumptions made earlier (chapters 4.2.2, 4.3.3, 4.4.6) what concern the enterprise environment. Our guidelines for the integration did not change. From the user's point of view, a business system shall use COMMOTION to convey business information to the user, and to complement and enrich the management of information that is not subject to the business system. From the system architect's point of view, the required way of integration is to strive for reusing resources (devices, services) between the business system and COMMOTION. The conceptual designer's viewpoint is based on keeping the business and COMMOTION models separate and enable seamless interoperability between them.

The user's, the system architect's and the conceptual designer's abstractions on integration lead to three distinct issues:
a common architectural foundation is the most fundamental issue for the integration of the multimedia and the business system

in order for the two systems to interoperate, an integration method must be established that defines a common syntax

the integration method must also define common semantics that is expressed using the common syntax

In order to realize integration, the most fundamental issue, the common architectural foundation must be established. In the next chapter, we examine existing enterprise architectures and their suitability for integration.

6.3 Levels of Integration

In the title, we used integration and realization as synonyms.

6.4 Integration into Existing Enterprise Architectures

In this chapter we describe possibilities to integrate the COMMOTION model into existing, mainstream enterprise architectures. Existing architectures for business systems already contain multimedia management functionality to some extent and also suggest ways for extensions. Due to this, incompatibilities in their architecture and in the COMMOTION architecture might hinder seamless integration, and thus, the revisited goals of the stakeholders might not be fully satisfied.

We examine three cases, which are the script-based, active control-based and mixed architectures. These architectures dictate their way for integration and also limit the quality and extent of it.

6.4.1 Script-based Architectures

The main feature of script based integration is simplicity. The architecture is a classical "three-tier" architecture comprising a data-tier, an application tier and a client/UI tier. A typical architecture is depicted in figure 6.1.
6.4. INTEGRATION INTO EXISTING ENTERPRISE ARCHITECTURES

The integration of multimedia information management can be accomplished in two ways: in the script generation phase (middle-tier) or in the plug-in (client-tier).

Attaching a COMMOTION container with components to the application server can be solved by an activity that exports a COMMOTION model to HTML that can be woven with the business data in the script generator, and the HTTP server part of the application server can be used to transfer it to the client, which is a browser application. The exporting process is necessary, since there is no notion of distributed object model into which COMMOTION could be integrated.

The main deficiencies of this approach are twofold. The one deficiency is the inadequacy of HTML to capture COMMOTION. Generic languages, such as XML-based scripts could be created to support advanced issues (e.g. SMIL), which would give more room for exporting COMMOTION, however, the coexistence of different technologies causes a confusing heterogeneity to the user. A good example is the navigation facility of the browser environment versus navigational interactivity provided within the content.

The other deficiency is the bidirectional communication of information, which is crucial in a full interactive information management session. Communication is severely constrained by the web-server/paradigm, that is, the underlying HTTP protocol and the architecture of web-servers and clients. Moreover, this paradigm only features document shipping, without the possibility to provide and control transactional context, which also is a fundamental enabler of distributed multimedia information management. Details of this approach is provided by the author in [75] and [76].

Another way of integration is to export the COMMOTION model to an opaque type and let it be handled by a client plug-in. With respect to the integration of COMMOTION, client plug-ins are not a perspective, because although they provide an alleviation to the limitations of HTML/XML, they are unable to solve the above mentioned problems. Having said that, time-dependent media-types, such as audio and video can only be provided as plug-ins in the client environment.

The architecture depicted can be generalized by introducing a distributed object model between the data-tier and the application server. This generalization does not improve the architecture substantially, since the distributed object model does not reach the client.

6.4.2 Active Control-based Architectures

Active control-based architectures present a different approach than the script-based architecture: active control provides a customized view on information provided by the middle-tier. The view, as opposed to scripts is defined at the client side. Figure 6.2 gives an overview of a typical architecture.

Similarly to the script-based approach, the integration of COMMOTION can be accomplished at the application server side as well as in the client side. The integration into the application server would mean its extension by a media container to provide COMMOTION objects that are accessible from the client. The advantage of this approach is that in case the transactional context and persistence is provided by the application server, it can also be applied to COMMOTION objects such that client interaction with COMMOTION objects can seamlessly be provided. This approach requires that active controls that realize a view on COMMOTION are also provided in the client. The active controls can run as a standalone application, or in an execution environment in a browser client. There can also be done a client-only integration. In this case the application server is not reused.

Although this approach solves some of the problems that arose in the script-based architecture, there are few unsolved problems: Document shipping is not featured by this architecture, object access is referential due to the transactional context. Similarly to the script-based architecture, in case of providing the user with several active controls, separately for media and business, the user might easily be confused by the inconsistence of independent controls. Active controls are not first class citizens in a distributed architecture, they are local views on a distributed model. This constrains their execution (e.g. there is no container provided for active control components) and configurability (their configuration cannot be automatically made persistent).
6.4.3 Mixed Architectures and Interoperability

There are different ways to combine advantages of both architectures. Although the script-based approach would provide uniform rendering for both the business and the media model, the rendering engine is a black-box, whose functionality cannot be extended.

The other way is to provide active controls not only for referential access, but also for document shipping and time-dependent media streaming. This approach is promising, since functionality extensions can be provided on an active control basis, along with the possibility to configure them independently. This architecture would require changes in the application server concerning transactions, persistence and streaming coordination. Even if alleviating these, some problems still persist. The top half of figure 6.3 shows an abstract architecture that illustrates the problems, active controls are depicted as views.

The one problem is that there is a coexistence of unrelated business and media controls that are presented as concurrent views to the user. This problem is present in all architectures (script-based, active control-based and mixed).

The other problem is centered around the interoperability of the business and the media model. We distinguish two kinds of interoperability: business and media interoperability. Business interoperability is the interoperability within the same or different media models. Business interoperability is accomplished within the business model, it has no relevance to the media model and the architecture.

Media interoperability however, represents the second problem. Media interoperability in general means the interoperability between models that feature media elements. Business views also contain rudimentary media elements, such as simple GUI management. The interoperability of media elements in business views is a lesser problem, because these are mostly orchestrated by the user, such as cut and paste, drag and drop etc. Media interoperability between business and media views represents a major problem, because complex interoperability cannot be accomplished through heterogeneous views. The following example illustrates this problem:

**Example:** The police department's information system is contains a business model that is based on a person's database featuring all important personal data. The database is queryable for every
Figure 6.3: Concurrent business and media views and their interoperability
attribute featured by the model. The user interface comprises two business views, the one is a form interface (BV1), the other displays the textual record describing committed crimes of the person active in the form interface (BV2). BV2 is interactive in the sense that names of spots in the crime description are hypermedia anchors showing the spot on the map of MV2. The user interface also contains two views, the one (MV1) shows the photograph of the active person. MV1 is interactive in the sense that the images it shows can also be used as a base for similarity search, moreover they can also be altered to perform a search based on montage synthesis. The other media view (MV2) is a map that shows spots related to the person active in other views. The map is interactive: the spots clicked reveal persons related to the spot on views BV1, BV2 and MV1.

It is obvious that BV1 and BV2, and MV1 and MV2 are views that are produced by active controls attached to the business model and the media model, respectively. These views must interoperate in order to reflect changes and update each other. As mentioned above, in case the views are of the same model, such as BV1 and BV2, interoperability can be accomplished within the business model. For instance, if a query for a person is issued in BV1, a BV2 update is pushed from the model automatically, the view does not need to perform any proactive action, but to be prepared to receive an event to update itself.

Since the media model is separated from the business model, their interoperability can only be accomplished among MVs and BVs. For example, if someone performs a similarity query on an image in MV1, the image result is associated with a symbol that can be queried for in BV1 in order to obtain a full attribute set of the person. If someone clicks the map in MV2, a spatial query is performed to figure out, which registered location is the nearest. The location is associated a symbol that can be queried for in BV2 to obtain descriptions of crimes took place at that spot.

View interoperability causes several deficiencies. Clients become too heavy, since they must contain interoperability logic. Clients also become specific, since views contain interoperability logic with respect to certain other views. As a consequence of heavy clients, their scalability and reusability is severely limited.

Two solutions arise to alleviate these deficiencies: a common model and model-level interoperability. The main advantage of a common business-media model would be the reduction of complexity. The disadvantages would be the limitations in scalability and genericity. Model-level interoperability seems to be the suitable way for integration. The following few chapters elaborate on this issue.

### 6.5 Abstract Architecture

We describe a common abstract architecture that is the base for the integration that alleviates the problems described previously. The key of integration is the augmentation of the mediator pattern. The mediator promotes loose coupling by keeping objects from referring to each other explicitly. In the current context, it:

- mediates between the business and the media model
- bidirectionally maps the business model and media model
- frees the client from upholding both views, it lets the client to only understand the media model

The use of the mediator pattern is shown in the bottom half of figure 6.3. It delivers the following advantages:

- it enables the variation of models independently
- in enables direct collaboration between the media model and the business model without deferring collaboration to their views
- it simplifies the client, since it only needs to know about the media model
- it can provide uniform views for both business and media models
6.6 The Active Mapping Process

The mapping process between the business system and the COMMOTION implementation is accomplished by an active mapping process which is executed in the mediator component. The mapping is active, because it actively maintains synchronization of the two parts between which it mediates. Active mapping is built on three notions:

- **relationship management**: Relation management connects business and media objects such that objects remain immutable. A business object can either be connected to a Media Object instance or template. Connection to a template means that the Media Object is the conveyor of the business object to the user, the content of the template will be filled in by model instantiation. Connection to an instance means that the Media Object is an enrichment of the business object, the Media Objects contains information the business object cannot.

- **model instantiation**: Model instantiation can refer to the instantiation of business objects as well as COMMOTION objects. The instantiation of business objects is accomplished by the mediator, when the instantiation is interactive and done by the user, such as by filling in a form. The instantiation of COMMOTION objects relates to the configuration of a COMMOTION template when it is used as a conveyor for business objects.

- **event management**: Event management is responsible for managing the dynamic behavior of the models: if instances of either model change, which impact instances of the other model, events are dispatched by the mediator to the corresponding instance.

6.7 An Integration Architecture

A typical architecture for the integration of the business system and multimedia information management is shown in figure 6.4. The architecture is traditional in the sense that it can be realized using mainstream technologies for architectural components without any need to reengineer the COMMOTION model and its architecture. These traditional architectural components are DBMSs, media subsystems, user interfaces etc. The architecture features six fundamental architectural components and two kinds of inter-component communication infrastructures.

The fundamental architectural component kinds are as follows:

1. **business server**: The business server is responsible for providing the realization of business resources and logic in terms of a consistent system of accessible objects possessing behavior (logic) and state (resources). Business resources may originate from proprietary and heterogeneous sources, the business server provides a consistent model through which the objects that encapsulate the logic and the resources can be accessed. The objects are typically realized as business components falling into categories like service, session, process and entity components. The business server is also responsible for offering transactional context and secure access for the objects it provides.

2. **media repository**: The media repository implements a lightweight COMMOTION architecture. This comprises a media container that provides persistence and access for CmtOs, but does not deal with (1) DCOs and their streaming, and (2) presentation functionality. The lightweight container is a better fit for implementation using “traditional elements”, such as DBMSs. DCOs have to be externalized into DOS and AppOs in order to be managed by the media repository. Due to size and temporal constraints, this cannot be done for all media-types.

3. **streaming server**: The streaming server complements the media repository in that it provides a media container that supports DCOs and streaming. The container provides devices that store and stream
Figure 6.4: A typical integration architecture
DCOs. The streaming server might omit complex management of DOs, AppOs and MOs, the provided management support might be rudimentary.

4. *media client:* The media client is either a medium-weight or lightweight implementation of the COMMOTION model. The focus of the implementation is on the provision of presentation and interaction activities. The media client implements a media container that supports streaming but no persistence, caching might be supported though.

5. *media-enhanced application server:* The application server is the architectural component that realizes the mapping between the business model and the media model. The application server acts as a *business client* of the business server and as a *media server* for the media client.

6. *independent services:* Independent services are services that can be implemented as independent architectural components. They can be used by media containers and thus provided to COMMOTION components. The most important independent services are naming, event, relationship and distributed transaction services.

The fundamental communication infrastructures between these architectural components are as follows:

1. *distributed object bus:* The distributed object bus is a complex communication infrastructure. It is the basic means to communicate objects between architectural components both by value and by reference. It is used for communication between components in different containers, as well as between the media and the business model.

2. *media streaming:* Media streaming is an ancillary communication infrastructure between architectural components that need to stream COMMOTION objects.
Chapter 7

Validation, Conclusions, Outlook

"I quite agree with you," said the Duchess; "and the moral of that is—'Be what you would seem to be'—or, if you'd like it put more simply—'Never imagine yourself not to be otherwise than what it might appear to others that what you were or might have been was not otherwise than what you had been would have appeared to them to be otherwise.'

(Lewis Carroll: Alice in Wonderland)

7.1 The Goal of This Chapter

In this last chapter, we validate the results of this work in terms of the initial statements, so that those statements form a frame for the whole work. For the validation we recall the scenario and explain how the project can be accomplished with the results of this work. We provide a validating overview of the project using the perspectives of the stakeholders. After that, we make a general conclusion of the work and put it into into a short and long-term future perspective.

7.2 Validation Rules

The order of validation is different from the order of stakeholders we used throughout the work: in the introductory project scenario we put the user to the first place, because we started the scenario from her perspective, and put the conceptual designer to the last place, because we realized the importance of the conceptual designer's role during the project.

During domain analysis, we changed this order, because the first issue in each domain was the description of domain concepts, and the last issue were realization possibilities through the role of the system architect.

In the this chapter we use yet another order for tackling the perspectives of each stakeholder. Since we validate the design and realization of a system, namely COMMOTION, we need to put the conceptual designer to the first place in order to assure the validity of the design first. Secondly, the user's perspective must be validated, since COMMOTION must be a valid foundation for all the metaphors required by the user. Thirdly, the system architect's perspective must be tackled, because the conceptual design must have a valid realization in a particular environment.

7.3 Validation from the Conceptual Designer's Perspective

Reiterating the requirements in chapter 1.4, the conceptual designer needs common assets that:
1. are based on an abstract model, rather than a language, an API or exchange format
2. are generic enough to be a solid and consistent foundation and abstraction for all metaphores and architectures and their components, respectively
3. are specific enough to be straightforwardly reusable for all multimedia enhanced domains of information management
4. provide semantics support for the integration of multimedia and business systems

The work presented here satisfies the requirements stated. The reasoning is broken down in the following chapters.

7.3.1 Abstract model and its Reifications

COMMOTION is an abstract model exploiting the object-oriented paradigm. It is not based on any particular language, API or exchange format. Despite this, a particular reification of COMMOTION can be bound to any language, can be accessed using several APIs and can also be serialized to be exchanged in different ways.

The abstract model has been presented in UML, which is a generic purpose model, therefore it is free from language and implementation issues. The language bindings we have defined are either C++ or Java. C++ (and perhaps also C) are meant as language bindings for the server and other heavy-weight components, whereas Java is meant as a language binding to be facilitated for lighter-weight clients. The language binding has obviously no implication on the functionality and the semantics of the model.

The API, and the programming model that is suggested by the API are the next important issues in the reification of COMMOTION. As we have seen in chapter 6, the API for accessing COMMOTION does not depend on the selected architecture.

7.3.2 Model Genericity

As shown in this work, the COMMOTION model provides both a generic foundation for metaphors, and an abstraction for architectural components.

The generic foundation for metaphors is fulfilled by the notion of Media Objects that incorporate all relevant aspects of multimedia information management, focusing on appearance and activities. Media Objects achieve this generic foundation consistently by componentization and the dynamic and compound notion of mediatype.

The abstraction of architectural components is fulfilled by the notion of resources, namely services and devices. These two resource notions can incorporate simple, local as well as complex, distributed services. In this way, the implementation

7.3.3 Domain Reuse

Since the model has been created as a common abstraction over the analyzed domains, it contains their major concepts, therefore these domains can be straightforwardly reproduced using a certain architecture and COMMOTION.

7.3.4 Integration Support

COMMOTION provides semantics for the integration with business systems, since it contains all the concepts that are needed to represent business content with it, and it also able to manage content, which is beyond the scope of business.
7.4 Validation from the User’s Perspective

We recall the requirements from the user’s perspective we posed in chapter 1.2. The requirements were separated into generic metaphor issues and specific foundation metaphors used in multimedia information management. The following two chapters elaborate on the validation of these two issues.

7.4.1 Generic Issues

As stated in chapter 1.2, metaphors the user interacts with shall possess some crucial features. COMMOTION proved to be a valid foundation to all the features of metaphors. This is described in the followings.

Similarity to real-life artifacts is not directly within the scope of COMMOTION, but the appearance and the activities that define such artifacts are major counterparts of COMMOTION, therefore the design of a suitable artifact is not compromised but reasonably supported by COMMOTION.

COMMOTION is a solid foundation for applying recurring schemas in metaphors. The reason for this being that metaphors can be composed of many components, rather than being monolithic. Fine-grain granularity of reuse contributes to design complex metaphors with recurring schemas.

The free determination of abstraction levels is one of the main features of COMMOTION, since it builds on abstracting and refining the information model, rather than encapsulating information. This flexibility is enabled by hierarchies of both the Media Object and the components of the Media Object (Data, Appearance, Activities etc.). This hierarchy is seamless, therefore it is optimally suitable for changing the abstraction levels. The only break in the seamlessness is at the lowest Media Object level, but this does not compromise abstraction.

Consistence between metaphors is a crucial issue for the user. COMMOTION supports consistence by two means: it maintains a single source of information all metaphors are based on (namely COMMOTION), and handles events through which Media Objects can notify each other. In this way, all metaphors contribute to the development of a consistent mental model in the user.

Integration with existing business systems is of great added value for the user, because she can use existing business systems along with the multimedia system. The integration described in this work is valid and comprehensive; it has a double impact on the business system. It both provides a multimedia front-end for the business system, and it extends it with functionality which is beyond its scope.

7.4.2 Specific Metaphor Issues

Besides the validation of generic user issues we tackled in the previous chapter, there exist specific metaphor issues described in chapter 1.2. The four metaphor kinds that have been described can all be realized with the Media Object using its components, mainly the Activity and the Appearance Objects.

7.5 Validation from the System Architect’s Perspective

By reiterating the issues stated in chapter 1.3, we state that the system architect’s perspective is focused around the realization of COMMOTION on an architecture using architectural components that either exists or are to be developed.

The validation of the system architect’s perspective can be split into general issues that deal with the quality of the architecture and components, and into specific issues that deal with the reuse of specific architectural components by the framework.
7.5.1 General Issues

As described in chapter 1.3 the basic guidelines that concern the overall design of the architecture and the implementation of COMMOTION upon that architecture can be summarized in the level of reuse, the scalability and the maintainability of the architecture.

The dichotomy of genericity and specificity for the implementation reuse of architectural components is solved in this work by abstracting them as devices and services. On the one hand, service abstractions are generic and therefore more let room for mapping any specific component into a service, regardless of its location, implementation, quality etc. On the other hand, device abstractions are specific, and require from the underlying mechanism to precisely provide details about the mechanism to be abstracted. Therefore, using these two mechanisms simultaneously allows for mapping both simple and highly specialized and complex, generic architectural components.

Scalability is another key issue in the quality of the architecture. The work described here provides scalability by two means. On the one hand, the model itself is scalable by its component-oriented realization, which means that each constituent component can be substituted by light-weight as well as heavy-weight implementations, as long as component contracts remain satisfied. On the other hand, the containers are scalable through the devices, services it provides, and their implementations. Typical profiles for container scalability are client and server containers. Scalability of this architecture is symmetric, what concerns down and up scalability.

The maintainability of the architecture gets a key role in the deployment phase of a system. Design-time maintainability is realized in the system described in this work by designing the components and the containers to be suitable for a certain task in a certain environment. This is the most flexible way, but any change needs redesign. Configuration-time maintainability means configuration prior to deployment. This can also be done both for components (e.g. through attributes) and containers (through execution switches, component interconnection topologies etc.). Run-time maintainability is the most flexible way, but the degrees of freedom are limited. Run-time maintainability plays an important role in numerous places in the system, one characteristic example is the active mapping between the business and the multimedia system, which can be maintained completely dynamically.

7.5.2 Specific Issues

This work describes the validity of how COMMOTION can be broken down to, and realized in components. The architecture that has been described in this work is flexible enough to incorporate all the component kinds and let them interoperate in a distributed environment. Containers are the main abstraction in which components execute, in that they provide devices and services for component execution. The previous chapter validated the generic architectural qualities of the work, this chapter focuses on details what concern the concrete architectural components to be reused. The most important architectural component kinds to be (re)used are focused around the fields of persistence, transaction management, access, user interfaces. We also mentioned overall solutions that try to capture the most of these fields.

The most outstanding result of the work that can be validated is the explicit transparence of the architecture what concerns the incorporation of the architectural components. All crucial mechanisms are either described in terms of devices, or services, whereby services themselves are composed of devices. This means that there the opaqueeness of the architecture is only influenced by the architectural component to be reused (e.g. a particular device), and not by architectural mechanisms themselves.

Persistence is the most crucial field of services for a multimedia system. The work described here provides persistence in two ways, with explicit transparence. It provides object-based persistence, in which the state of a component instance is made persistent through a service. Persistence is also provided within the framework using a special configuration of streaming, where the one end of the stream is a persistent device such as
a harddisk. With streaming, caching and transfer of the data as well as other mechanisms can be modelled explicitly.

Transactions are also provided by the containers in that components execute. A particular contain can provide local transactions for the data it provides persistence to, and can also participate in distributed transactions along with other containers.

The architecture described in this work supports all major kinds of data access mechanisms. As we have seen, access kinds and access kind implementations are two important architectural issues. The architecture allows for three kinds of access mechanisms, such as reference-based, value-based and streaming-based. Implementations of these mechanisms depend on the locality of components and containers: it reaches from direct pointer passing to the use of distributed object technology mechanisms, such as CORBA.

User interfaces and their typical MVC mechanisms can be incorporated into COMMOTION. The system architect’s concerns are the embedding graphical and multimedia subsystems into the implementation of COMMOTION. This embedding can be validly transformed into the services and devices of the container architecture.

7.6 Conclusions

This work has tackled a wide range of issues in multimedia information management starting from fundamental definitions, arriving at the integration with business systems. The work has presented contributions in several fields. These are summarized in the followings.

The first major contribution of this work has been the definition of a terminology in which the role of domains, models, architectures, frameworks and components have been clarified. It has presented a joint and consistent view on the above issues that could be used as a solid basis for the whole work.

The second major contribution of this work has been the analysis of multimedia information management. After clarifying fundamental issues of media and information management, it has presented a novel approach of analysing multimedia enhanced domains. The novelty of the approach has been that the analysis of domains was accomplished from the viewpoints of several stakeholders.

The third major contribution of this work has been the definition of fundamental common abstractions on multimedia information management. The novelty of the contribution has been the incorporation of all relevant stakeholders of multimedia information management into the definition of the abstractions.

The fourth major contribution of this work has been the establishment of a model, COMMOTION. The novelty of the COMMOTION model has been comprehensiveness, and the suitability for abstraction and refinement in several dimensions of extensibility. These features uniquely enabled the efficient componentization of COMMOTION.

The fifth major contribution of this work has been the componentization of COMMOTION and the establishment of a sound architecture. This has comprised the definition of components and their communication. The novelty of componentization has been the uniformization of the notion of components and architectures: there have not been separate architectures and communication mechanisms for server (heavy-weight) and client (light-weight) components.

The sixth major contribution of this work has been the integration of the multimedia information management framework with business systems. The novelty of the integration has been threefold: it has been non-intrusive (it has allowed for the multimedia and the business systems retain their independence), flexible (it has provided for a dynamic, run-time mapping process) and uniformity (the client-side of the mapping only needs to know about one model).

The six major contributions have together formed a complex consistent platform for multimedia information management, which consequently follows a single methodology from the foundations to deployment.
7.7 Outlook

The work that has been described in seven chapters above is not a pure theoretical work. While it does not provide a full implementation of the framework, it contains two kinds of implementation, which has not been included with the written part of the work. The reason for this being that the implementations either have not been finished or had an indirect impact only of the theoretical decisions made in the work.

The one kind of implementations are small programs that helped gain insight into the mechanisms of concrete storage systems (e.g. ODBMSs), middlewares (e.g. CORBA), APIs (e.g. JFC), operating systems (e.g. Unix IPC).

The other kind of implementations are directed to the future. The theoretical foundations and results achieved in this work are now being used by a small team led by the author of this work to develop a production-quality framework for multimedia information management. The development of the framework to be implemented has been preceeded by several strategic decisions.

The operating system chosen is the Linux [43] operating system, because it is a popular, widely accepted, open-source operating system with a tremendous amount of software tools available. Linux, as other traditional monolithic operating systems is not optimally suitable to satisfy requirements posed by multimedia. On the other hand, specialized operating systems and kernels such as the Be OS [14], SUMO microkernel [72] etc. are closed source, their hardware platform availability is severely limited, and have failed to gain wide acceptance in software communities. Linux provides a huge potential for changing some of its crucial components to be suitable for multimedia, therefore it is a strategic choice to bring these ideas closer to a wider range of people.

The middleware technology chosen is CORBA, because it is a freely available industry standard. It is the goal of the implementation project to alleviate the problems that arise from applying CORBA at coarse as well as fine grain granularities.

The storage system chosen does not reuse any commercially or freely available product. While the provision of containers is possible over most of the available systems, the implementation project decided to develop a storage system from scratch. The main reasons for this being efficiency, scalability and availability.

The overall goal of the implementation project is twofold. The first goal is to provide a framework for multimedia information management to the open-source community. the second goal is to develop value-added commercial products that build on the framework and are targeted towards a certain application domain.
Bibliography


Nederlandse samenvatting

Inleiding

Het werk begint met een inleidend hoofdstuk waarin de probleemstellingen worden geformuleerd, gebaseerd op een praktijk-senario. Het scenario speelt zich af op een politiebureau dat een multimedia informatie-management-systeem wil implementeren dat informatie acquireert, opslaat, toont, bevaagt en transporteert m.b.v. een variëteit van waarneembare kanalen, zoals plaatjes, video's, teksten etc. De politie wijst deze taak toe aan een softwarebedrijf. Het bedrijf zet consultants in, welke zorg dragen voor het gebruikersperspectief op het toekomstige systeem en voorzien in een formele beschrijving ervan. Daarnaast zet het systeemarchitecten in die het gewenste systeem in elkaar zetten met beschikbare componenten. Snel blijkt dat de geproduceerde output van de consultants en de gewenste input voor de systeemarchitecten niet op elkaar aansluiten, zodat het project op het punt staat te mislukken. Het bedrijf realiseert zich dat het slechts vergeten is conceptuele ontwerpers te betrekken bij het project, die de rol zouden hebben van het hebben van een abstracte zienswijze op het toekomstige systeem, onafhankelijk van de gebruikerswensen en de realisatiearchitectuur. Om hun gezichtspunt te kunnen uitdrukken hebben conceptuele ontwerpers een krachtige modelleeromgeving nodig die zich bevindt op het juiste abstractieniveau om de gebruikerswensen te kunnen verbinden met de architecturale componenten. De projectleden definiëren gezamenlijk de kritieke punten in het project, die we gebruiken als de initiële probleemstelling voor dissertatie:

- de gebruiker van een multimedia informatie-management-systeem wordt geconfronteerd met te veel metaforen die vaak conflictiserende zienswijzen op multimedia blootleggen
- multimedia informaticemanagement-systeem zijn sterk gebonden aan specifieke architecturen en gebruikersbehoeften, en daarom ontbreken vaak een architectuur- en gebruikersafhankelijke semantiek, met als gevolg een mager hergebruik tussen complexe systemen
- abstracte modellen voor formele beschrijving van een systeem zijn gewoonlijk of te specifiek, een bepaald aspect van het systeem benadrukkend, of te generiek voor hergebruik op hogere niveau's
- architecturale ondersteuning voor de inpassing van multimedia in modale bedrijfsomgevingen is problematisch vanwege de integratie van gebruikersmetaforen met middleware en back-end componenten van het bedrijf
- semantische ondersteuning voor het verweven van bedrijfsinformatie-management met multimedia informatie-management is ook problematisch, omdat het gewoonlijk plaatsvindt op het data- of view-niveau, terwijl de mogelijkheid buiten beschouwing wordt gelaten van uitwisseling van informatie m.b.v. een gemeenschappelijk, goed onderbouwd model

Fundamentele principes van softwaresystemen

Omdat elk domein zijn eigen terminologie heeft definiëren we, om conflictiserende terminologie te voorkomen, de meeste fundamentele termen die zijn gebruikt in dit werk. Het hoofdstuk begint met de definitie van
modellen, model contexten, meta-niveaus en domeinen. Omdat modelleren op oneindige meta-niveaus te generiek is, en domeinstructuren te specifiek zijn, is het nodig een generieke domeinarchitectuur te definieren waarin actoren de ankerpunten bepalen. Actoren worden de belanghebbenden bij het systeem, zij benadrukken verschillende aspecten, en hebben verschillende belangen en eisen.

Naast domeinen definiëert dit hoofdstuk ook het object-paradigma en componenten, welke zullen worden gebruikt in dit werk.

**Analyse van multimedia informatiemanagement**

Dit hoofdstuk selecteert een aantal multimedia domeinen en voorziet de lezer van een degelijke analyse ervan. Voorafgaand aan deze analyse worden verschillende media- en informatiemanagement-definities aangeleverd. De focus van de definities is gericht op kunstmatige, digitale en elektronische media welke worden onderverdeeld in presentatie-, opslag-, transmissie-, representatie- en perceptie-media. Monomedia en multimedia hebben betrekking op perceptiemedia, multimedia informatiemanagement heeft betrekking op het feit dat informatie wordt beheerd op een multimediale wijze, dit in tegenstelling tot bedrijfsinformatiemanagement.

Domeinen worden geselecteerd op basis van hun relatie tot multimedia informatiemanagement. Wij selecteren multimedia georiënteerde domeinen zoals (1) inhoudverwerking en -samenstelling, (2) inhoudgebaseerde indexering en retrieval, (3) documentmanagement en workflow, (4) continue mediaopslag en streaming, (5) medialisatie (een generalisatie van visualisatie voor alle waarneembare media), (6) gebruikersinterfaces en (7) multimedia publicatie en presentatie. De analyse van de domeinen reflecteert de domeinarchitectuur zoals opgezet in het vorige hoofdstuk: elk multimedia georiënteerde domein wordt beschreven vanuit het gezichtspunt van drie belanghebbenden: de conceptueel ontwerper, de gebruiker en de systeemarchitect.

Na analyse wordt een conclusie gepresenteerd. De bevindingen van de conclusie kunnen worden beschouwd als de initiële probleemstellingen van de dissertatie gezien vanuit een andere hoek. Het combineren van multimedia georiënteerde domeinen voor het bouwen van complexe toepassingen is een absolute noodzakelijkheid, omdat geen van de domeinen voorziet in uitputtende functionaliteit voor multimedia. Om in staat te zijn multimedia-toepassingen te bouwen die zich uitstrekken over meerdere domeinen, hebben wij drie mogelijkheden:

- neem een bestaand domein en breid dit uit met alle mogelijke architecturele en conceptuele oplossingen
- laat elk domein intact en lever verbindingen tussen de domeinen
- introduceer een nieuw, uniform en generiek domein hetwelk alle concepten van de domeinen uitputtend bevat

Na een degelijke bestudering van de verschillende mogelijkheden hebben wij besloten om de derde aanpak te volgen en een nieuw, generiek domein te ontwikkelen voor multimedia informatiemanagement.

**Gemeenschappelijke abstracties in multimedia informatiemanagement**

In dit hoofdstuk worden de concepten van de belanghebbenden nog eens bekeken en worden ze geunificeerd naar een nieuw domein. De gebruiker stelt belang in de inhoud die hem bereikt d.m.v. metaforen. Inhoud is een concept dat de gebruiker hanteert als een algemene abstractie voor verschillende media en de informatie die deze bevatten. Echter, de gebruiker is niet in staat om inhoud waar te nemen, hij/zij kan dit alleen doen d.m.v. metaforen.

De systeemarchitect begint bij media en informatie. Hij abstrahieert media naar apparaten en informatie naar apparaat-inhoud. Apparaten communiceren met elkaar door hun apparaat-inhoud uit te wisselen. Apparaten
kunnen verder worden geabstraheerd tot services, welke lokatie-onafhankelijk zijn (in tegenstelling tot apparaaten), en welke meerdere apparaten kunnen omvatten en meer complexe functionaliteit kunnen bieden.

De conceptueel ontwerper moet apparaat-inhoud verbinden met metaforen. Hij definiert de concepten structuur, verschijning en activiteit om een gemeenschappelijk begrippenkader te bieden voor de systeemarchitect en de gebruiker. Deze drie concepten van de conceptueel ontwerper worden algemeen aangeduid met ‘conceptuele media’. Conceptuele media kunnen worden getypeerd, hiermee de notie van media-types met zich meebrengend. Omdat conceptuele media een samengesteld concept vormt hangt de typering af van de onderdelen. Daarom worden media-types als dynamisch beschouwd, gedefinieerd door de structuur, verschijning en activiteit van de abstracte notie van inhoud.

Het ontwerpen van multimedia componenten

Dit hoofdstuk introduceert een model dat formeel alle fundamentele concepten beschrijft die in het vorige hoofdstuk zijn gedefinieerd. Het model is COMMOTION genoemd (Comprehensive Object Model for Multimedia Information Management). De fundamentele objecten van COMMOTION zijn het media-object, het data-object, het verschijnings-object (appearance), het activiteit-object, het apparaat-object (device), het apparaatinhoud-object (device content) en het stream-object.


- Het data-object: dit object modelleert de structuur van conceptuele media. Het is hiërarchisch, haar knooppunten refereren naar structurele elementen, echter, onafhankelijk van de vorm van apparaatinhoud. Het kan zowel in haar knooppunten inhoud internaliseren als deze externaliseren door gebruik te maken van apparaatinhoud-objecten.

- Het verschijnings-object (appearance): dit object modelleert de verschijning van conceptuele media in termen van visuele, auditieve, tijdelijke en interactieve verschijningsvormen.


- Het activiteit-object: Dit object staat voor opgedeelde functionaliteit. Samenstellingen van dit object vormen de wijze waarop dynamisch multimedia beheer-functionaliteit kan worden geassembleerd.

- Het apparaatinhoud-object (device content): Dit object staat voor de inhoud van een bepaald apparaat. Inhoud in een gegeven vorm wordt gecommuniceerd met andere apparaten door streams.

- Het stream-object: Dit object regelt de configuratie, besturing, topologie en synchronisatie van apparaatinhoud stromen.

De kracht van COMMOTION is meervoudig. In de eerste plaats is het uitputtend. Het biedt compleetheid door all gemeenschappelijke concepten van alle domeinen te modelleren vanuit het perspectief van alle belanghebbenden. In de tweede plaats is het flexibel door de samenstelling van conceptuele media vanuit hun onderdelen, welke onafhankelijk kunnen worden samengesteld, uitgewisseld en gemodificeerd. In de derde plaats ondersteunt het veel niveaus van abstractie-granulariteiten. Alle informatie die door COMMOTION wordt beschreven en beheerd kan worden beschouwd op het tijnsle en grootste niveau m.b.v. dezelfde concepten voor het erop toepassen van activiteiten (opslag, wijziging, manipulatie, presentatie etc.).
De realisatie van COMMOTION behoeft een bepaalde architectuur. In het algemeen beschrijft de architectuur de wijze waarop COMMOTION objecten kunnen worden gedistribueerd, hoe zij communiceren met elkaar, hoe de status van objecten kan persisteren, hoe de transactionele inhoud van objectcommunicatie wordt beheerd en welke generieke services ter beschikking staan van een object.

In het bijzonder beschrijft de architectuur hoe COMMOTION objecten worden 'gecomponentiseerd', dat is, in welke eenheden worden zij ingezet. De componentarchitectuur waarmee COMMOTION kan worden verweven onderscheidt vijf verschillende noties. Het data-object, het verschijnings-object, het activiteits-object en het stream-object zijn media-componenten. Het media-object is een media framework (substraat) dat is samengesteld uit mediacomponenten. Samengestelde media-objecten vormen een media-architectuur die is gebouwd uit media-frameworks. Een media-container is een lokaal executie-framework voor mediaarchitecturen, en als laatste, een gedistribueerde media-architectuur is een coherente verzameling van media-containers die wordt gecoördineerd om een gedistribueerde media-omgeving te vormen.

**Realisatie en integratie**

De realisatie en integratie van COMMOTION is de laatste belangrijke stap in de ontwikkeling van een multimedia informatie-management-systeem. We kunnen realisatie niet scheiden van integratie omdat de realisatie-omgeving van het systeem een gemeenschappelijke infrastructuur moet delen met de realisatie van het andere systeem waarmee we ons systeem willen integreer. In het bijzonder wordt integratie met een bedrijfssysteem in detail behandeld. De dissertatie definieert integratie als de definitie van een gemeenschappelijke integratie-architectuur voor een multimedia-systeem en een bedrijfssysteem.

Gebaseerd op het voorgaande onderscheiden we vier verschillende integratie soorten: script-gearbeide, actieve besturing-gearbeide, en gemengde architecturen. De dissertatie weidt uit over deze drie architecturen, beschrijft hun tekortkomingen en stelt een vierde integratie-architectuur voor: de actieve 'mapping' gebaseerd architectuur. De uniciteit van deze architectuur is tweevoudig.

In de eerste plaats is het multimedia-systeem geen gelijktijdig systeem t.o.v. het bedrijfssysteem, beiden staan ter beschikking van de gebruiker, maar het multimedia-systeem heeft een dubbele impact: het is een uniforme, coherente front-end voor één of meerdere bedrijfssystemen, en het complementeert ook het bedrijfssysteem met functionaliteit dat het niet biedt.

In de tweede plaats kunnen de twee systemen onafhankelijk worden beheerd, gemodificeerd en vervangen, meer dan dat, hun onderlinge communicatie, het actieve 'mapping'-proces kan dynamisch in run-time worden geconfigureerd. Deze eigenschappen geven de integratie-architectuur een grote flexibiliteit en schaalbaarheid.

**Validatie, conclusies en vooruitzicht**

De validatie doorloopt ondermaal de probleemstellingen die aan het begin van dit werk zijn geformuleerd. Het vat de validiteit van de resultaten samen vanuit het perspectief van de gebruiker, de systeem-architect en de conceptueel ontwerper, met de bedoeling een totaaloverzicht op het geheel te schetsen.

De conclusies vatten de belangrijkste bijdragen van dit werk samen in zes punten. Deze zijn:

- de definitie van een coherente terminologie die een gezamenlijke en consistente zienswijze op het onderwerp reflecteert
- de analyse van multimedia informatie-management vanuit het perspectief van alle belanghebbenden
- de definitie van gemeenschappelijk fundamentele multimedia-concepten dat alle belanghebbenden omvat
□ de ontwikkeling van een model, COMMOTION, dat uitputtend de hierboven gedefinieerde concepten bevat

□ de 'componentisering' van COMMOTION tezamen met de totstandkoming van een goede architectuur

□ de integratie van multimedia informatiemanagement met bedrijfssystemen

Het vooruitzicht concentreert zich op de implementatie van hetgeen is bereikt. Het benadrukt het belang van standaarden en open technologieën omdat deze zouden leiden tot een snelle disseminatie en gebruik in gedistribueerde software-ontwikkelingsgemeenschappen.

Het vooruitzicht noemt het Linux besturingssysteem als de beste kandidaat voor een efficiënte, geheel nieuwe implementatieomgeving. Het noemt CORBA als de beste open standaard voor heterogene gedistribueerde object-infrastructures en noemt Java als een wijdverspreide, platform-onafhankelijke, mobiele taal.
About the Author

Gábor Szentiványi was born in Budapest, Hungary, on the 6th of November, 1970. In 1989 he finished secondary school studies and attended the University of Technology Budapest (BME). During his studies, he was involved in several multimedia projects in Germany, at the University of Karlsruhe. He prepared his MSC at Digital Equipment Corporation (DEC). After graduating, he worked as a research assistant in multimedia at the GMD (German National Research Centre for Information Technology), in St. Augustin. In 1995, he joined the Delft University of Technology to work as a research assistant under the supervision of Prof. Dr. Ing.-habil Waltraud Gerhardt. The core topic of the research team, database systems substantially influenced his view in the multimedia field; this is reflected in the current thesis.

Gábor Szentiványi has been involved in several joint European projects that focused on information management. His database management and multimedia background helped him be a core member of the projects, and share his expertise with domain experts of several business fields. Besides his project activities, he attended several international conferences, where he was awarded for his novel work in the field of multimedia information management.

At present, his main interests are the commercial applicability and technology transfer of fundamental research results in multimedia information management.