# Erweiterbarkeit multimedialer Dokumentensysteme zur dynamischen Anpassung an anwendungsspezifische Anforderungen

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von Jürgen Hauser aus Kornwestheim

| | |
|---|---|
| Hauptberichter: | Prof. Dr. Dr. K. Rothermel |
| Mitberichter: | Prof. Dr. T. Ertl |
| Tag der mündlichen Prüfung: | 04.02.2004 |

Institut für Parallele und Verteilte Systeme der Universität Stuttgart

2004

# Danksagung

An erster Stelle möchte ich Herrn Prof. Dr. Dr. Kurt Rothermel für die Betreuung und wertvollen Diskussionen meiner Arbeit danken. Herrn Prof. Dr. Thomas Ertl danke ich für die zügige, unkomplizierte und hilfreiche Unterstützung bei meinem Promotionsvorhaben.

Der Deutschen Forschungsgemeinschaft danke ich für die Finanzierung meiner Forschungsarbeit.

Bei meinen ehemaligen Kollegen bedanke ich mich für die gute Zusammenarbeit. Hervorzuheben ist Dr. Stefan Wirag, der mir als Diskussionspartner eine wertvolle Hilfe bei der Durchführung des Projekts war. Bei Herrn Frank Scholze von der Universitätsbibliothek bedanke ich mich für die hervorragende Zusammenarbeit und Unterstützung.

Meiner Frau Silke danke ich für ihr Verständnis dafür, dass ich nicht immer Zeit für sie und meine Tochter Sara Lea hatte, und das Korrekturlesen meiner Arbeit.

Mit ihren Diplomarbeiten haben Jing Tian, Alexander Frey und Rod Groß mir bei der Implementierung eines Prototypen geholfen, der zur Überprüfung der in dieser Arbeit beschriebenen Modelle und Konzepte diente. Ebenso trugen Uwe Albrecht, Christian Czech und Kanyin Cai als wissenschaftliche Hilfskräfte zur Implementierung bei. Dafür möchte ich mich an dieser Stelle bedanken. Ebenso bedanke ich mich bei Lisa Freydank für das Korrekturlesen dieser Arbeit.

Kornwestheim, 07. März 2004

# Inhaltsverzeichnis

Danksagung . . . . . 3
Inhaltsverzeichnis . . . . . 5
Abkürzungsverzeichnis . . . . . 9
Kurzfassung . . . . . 11
Extensibility of Multimedia Document Systems for Dynamic Adaptation of Application-specific Requirements . . . . . 13

**1 EINLEITUNG . . . . . 27**
1.1 Anwendungsspezifische Multimedia-Präsentationen . . . . . 29
1.2 MAVA - Multimedia Document Versatile Architecture . . . . . 30
1.3 Aufbau der Abhandlung . . . . . 33

**2 SPEZIFIKATION VON MULTIMEDIA-DOKUMENTEN . . . . . 35**
2.1 Multimedia-Dokumente . . . . . 35
2.2 Anforderungsanalyse für ein Dokumentenmodell . . . . . 37
2.3 Existierende Ansätze . . . . . 40
2.3.1 Generische Ansätze . . . . . 40
2.3.2 Anwendungsspezifische Ansätze . . . . . 46
2.4 Zusammenfassende Bewertung existierender Ansätze . . . . . 50

**3 EIN ERWEITERBARES META-DOKUMENTENMODELL . . . . . 55**
3.1 Das Meta-Dokumentenmodell . . . . . 55
3.1.1 Medienelemente . . . . . 56
3.1.2 Operatoren . . . . . 56
3.2 Erweiterungen des Meta-Dokumentenmodells . . . . . 58
3.2.1 Container . . . . . 59
3.2.2 Effektoperatoren . . . . . 60
3.3 Anwendungsspezifische Instanzen des Meta-Dokumentenmodells . . . . . 63
3.4 Klassifikation anwendungsspezifischer Instanzen . . . . . 64
3.5 Einordnung und Zusammenfassung . . . . . 67

**4 ARCHITEKTUR ERWEITERBARER MULTIMEDIA-PRÄSENTATIONSSYSTEME . . . . . 69**

4.1 Beziehung zwischen Managern und Medienelementen . . . . . 69

4.2 Kommunikation zwischen Medienelementen und Managern . . . . . 74

4.2.1 Struktur von Dokumenten . . . . . 75

4.2.2 Ereignismengen von Managern und Medienelementen . . . . . 77

4.3 Syntaktische Konsistenz . . . . . 80

4.4 Darstellbarkeit von Dokumenten . . . . . 82

4.5 Architektur des Präsentationssystems . . . . . 83

4.6 Implementierung von Managern . . . . . 85

4.6.1 Interne Repräsentationen von Anwendungskonzepten . . . . . 87

4.7 Beispiel einer Erweiterung . . . . . 88

4.7.1 Interne Repräsentation der Beispielanwendung . . . . . 89

4.7.2 Erweiterung des Konzepts für computergestütztes Training . . . . . 92

4.7.3 Implementierung des Managers für computergestütztes Training . . 94

4.8 Zusammenfassung . . . . . 97

**5 VERMITTLUNG MULTIMEDIALER DOKUMENTE . . . . . 99**

5.1 Ein Dokumententransferformat . . . . . 100

5.1.1 Die Auszeichnungssprache XML . . . . . 100

5.1.2 Definition des Transferformats . . . . . 103

5.1.3 Aufbau des Objektmodells . . . . . 108

5.2 Speicherung von Dokumenten auf Internet-Servern . . . . . 112

5.2.1 Logische Sicht auf Dokumente . . . . . 113

5.2.2 Der Dokumentenladevorgang . . . . . 114

5.2.3 Dynamisches Nachladen von Erweiterungen . . . . . 115

5.3 Zusammenfassung . . . . . 117

**6 ARCHITEKTUR EINES ERWEITERBAREN AUTORENWERKZEUGS . 119**

6.1 Grundfunktionalität des Autorenwerkzeugs . . . . . 119

6.1.1 Graphische Visualisierung des Dokumentenmodells . . . . . 119

6.1.2 Interaktion während des Spezifikationsprozesses . . . . . 121

6.2 Autorenunterstützung für die Spezifikation . . . . . 123
6.2.1 Navigationsunterstützung durch Visualisierung der Baumstruktur 124
6.2.2 Filterung der Visualisierung von Dokumenten . . . . . 125
6.2.3 Vorlagen zur Wiederverwendung von Teilen von Dokumenten . . 126
6.2.4 Konzeptspezifische Sichten auf Dokumente . . . . . 129
6.3 Das Autorenwerkzeug . . . . . 131
6.4 Dynamische Konfiguration . . . . . 136
6.5 Bewertung und Vergleich . . . . . 141
6.6 Zusammenfassung . . . . . 143

**7 ZUSAMMENFASSUNG UND AUSBLICK . . . . . 145**

**A BEISPIELSPEZIFIKATION . . . . . 149**

**B BEISPIELSPEZIFIKATION IM TRANSFERFORMAT . . . . . 153**

**C LITERATURVERZEICHNIS . . . . . 157**

# Abkürzungsverzeichnis

| | |
|---|---|
| ANSI | = American National Standards Institute |
| DOM | = Document Object Model<br>Referenz [App+98] |
| DTD | = Document Type Definition<br>Referenz [BrS98] |
| GPS | = Global Positioning System<br>Referenz [HLC92] |
| HTML | = Hypertext Markup Language<br>Referenz [RHJ99, Pem+00] |
| ID | = Identifier |
| IDREF | = Identifier Reference |
| JAR | = Java Archive<br>Referenz [CaW01] |
| JPEG | = Joint Picture Expert Group<br>Referenz [HYS88] |
| MAVA | = Multimedia Document Versatile Architecture<br>Referenz [Hau99, HaR00, Hau00, HaR01, Hau01, HaT01] |
| MHEG | = Multimedia and Hypermedia Expert Group<br>Referenz [Mheg95, Mheg96] |
| MIME | = Multipurpose Internet Mail Exchange<br>Referenz [FrB96] |
| MPEG | = Moving Pictures Expert Group<br>Referenz [ISO93] |
| PDA | = Persönlicher Digitaler Assistent (engl. personal digital assistent) |
| PNG | = Portable Network Graphics<br>Referenz [Png95] |
| SMIL | = Synchronized Multimedia Integration Language<br>Referenz [Hos01] |
| URI | = Uniform Resource Identifier<br>Referenz [BFM98] |

URL = Uniform Resource Locator
Referenz [BFM98]

W3C = World Wide Web Consortium
Referenz [W3C]

WYSIWYG = What You See Is What You Get

XML = Extensible Markup Language
Referenz [BrS98]

# Kurzfassung

Multimedia-Präsentationen stellen hohe technische und programmiersprachliche Anforderungen an ihren Ersteller. Grundlage multimedialer Präsentationen sind so genannte Multimedia-Dokumente. Durch die Verwendung von Multimedia-Dokumenten kann der Erstellungsprozess definiert und strukturiert werden.

Multimedia-Präsentationen sind durch ihre Zeitabhängigkeit und die Interaktionsmöglichkeiten, die den Verlauf der Präsentation beeinflussen, gekennzeichnet. Damit unterscheiden sie sich grundlegend von textuellen oder statischen Dokumenten. Die Weitergabe von statischen Dokumenten erfolgt nach einer Ausgabe über den Drucker im Allgemeinen in Papierform. Multimedia-Dokumente werden dem Benutzer immer auf einem Rechner präsentiert.

Die Zeitabhängigkeit von Multimedia-Präsentationen wurde bereits in vielen wissenschaftlichen Publikationen behandelt. Die vorliegende Abhandlung widmet sich daher hauptsächlich den Interaktionsmöglichkeiten. Beim näheren Betrachten der Interaktion ist zu erkennen, dass sie vom jeweiligen Anwendungsgebiet, in dem das Multimedia-Dokument eingesetzt werden soll, abhängt. Neben der Interaktion selbst ist die Reaktion der Präsentation auf Interaktionen von Bedeutung. Letztendlich unterscheiden sich die Anwendungsgebiete in der Interaktionsform und den Reaktionen der Präsentation.

Diese Unterschiede sollten in der Spezifikationssprache für Multimedia-Dokumente sichtbar sein. In der Literatur gibt es zwei unterschiedliche Ansätze für Spezifikationssprachen. Der erste Ansatz sind generische Sprachen. Sie führen zu einem hohen Aufwand bei der Erstellung, da sie anwendungsunabhängig sind und damit den Autor nicht unterstützen. Anwendungsspezifische Spezifikationssprachen legen sich auf ein Anwendungsgebiet fest. Neue Anwendungsgebiete erfordern eine Neuentwicklung und bedeuten eine neue Benutzungsschnittstelle für den Autor. Viele unterschiedliche Systeme und Speicherformate erhöhen den Verwaltungsaufwand und erschweren die Weiterverarbeitung der Dokumente, beispielsweise in bibliographischen Nachweissystemen.

In dieser Abhandlung wird ein erweiterbarer Ansatz vorgestellt. Für den erweiterbaren Ansatz wird ein Meta-Dokumentenmodell entwickelt, das die Eigenschaften der Spezifikationsspra-

chen zusammenfasst, die unabhängig vom Anwendungsgebiet sind. Für konkrete Anwendungsgebiete werden Spezifikationssprachen als Instanz dieses Meta-Dokumentenmodells zur Verfügung gestellt. Die Erweiterbarkeit des Ansatzes bedeutet, dass sowohl in einem Autorenwerkzeug als auch im zugehörigen Präsentationssystem die Unterstützung für ein neues Anwendungsgebiet nachträglich hinzugefügt werden kann.

Die Implementierung einer Spezifikationssprache, als Erweiterung des Präsentationssystems, soll mit einem möglichst geringem Aufwand verbunden sein. Dazu wird eine modulare Architektur für ein erweiterbares Präsentationssystem vorgeschlagen. Für die Präsentation eines Dokuments müssen alle benötigten Erweiterungen dynamisch zur Laufzeit in die Architektur eingefügt werden. Idealerweise erfolgt dies ohne Zutun des Autors oder Benutzers automatisch durch das Präsentationssystem. Damit beschreibt das Transferformat für Dokumente, das ebenfalls die Erweiterbarkeit unterstützen muss, nicht nur die Struktur der Dokumente, sondern auch die Konfiguration des Präsentationssystems zur Präsentationszeit.

Neben der Entwicklung des Meta-Dokumentenmodells und des erweiterbaren Präsentationssystems wird ein ebenfalls dynamisch erweiterbares Autorenwerkzeug konzipiert. Die Erweiterbarkeit des Ansatzes soll im Autorenwerkzeug minimale Auswirkungen auf die Benutzungsschnittstelle haben. Aus diesem Grund greift die Benutzungsschnittstelle zur Erstellung von Multimedia-Dokumenten nur auf Eigenschaften des Meta-Dokumentenmodells zurück. Neben der Erweiterbarkeit werden Konzepte zur Vereinfachung des Spezifikationsprozesses untersucht.

# Extensibility of Multimedia Document Systems for Dynamic Adaptation of Application-specific Requirements

Multimedia presentations can be found in various areas. HTML pages can be considered as simple multimedia presentations and there are a lot of CD-ROMS that familiarize users with multimedia presentations. In the future even more multimedia presentations will be used because the increase of computing power of cell phones, for example, enables multimedia applications on them. In other areas like computer-based training more and more multimedia capabilities will be used. This leads to an increased demand of tools and experts that are capable of creating multimedia presentations.

A multimedia presentation is specified by a multimedia document. The tool that is used to create multimedia documents is called authoring tool. A multimedia presentation is characterized by the combination of different media types like audios, videos, images or simple text and the temporal order of their presentation. The temporal order of the presentation can be influenced by user interactions.

This thesis focuses on the view of a potential author who wants to create multimedia documents. Every multimedia document has an application area for its use. Examples for application areas are an interactive tourist guide or a computer-based training. These application areas have to be reflected in the authoring tool the author uses and in the underlying document model. In this thesis a so-called extensible approach is presented. That means that both the presentation system and the authoring system can be extended by functionality that is specific for a particular application area. Therefore a meta document model is introduced. Application areas are represented by instances of this meta document model.

The described models, concepts and the prototype implementation were carried out in a research project called MAVA (*M*ultimedi*A* document *V*ersatile *A*rchitecture) [Hau99, HaR00, Hau00, HaR01, Hau01, HaT01]. The project was located at the Institute of Parallel and Distributed Systems (IPVS) of the University of Stuttgart. It was funded by the German Research Foundation (DFG) in the strategic research initiative "V3D2" ("Distributed Processing and Delivery of Digital Documents") [V3D2].

In the next section, the specification of multimedia documents is covered. After discussing related approaches an extensible meta document model is introduced. Based on the meta document model the architecture of an extensible multimedia presentation system is presented. Storing and disseminating multimedia documents based on the meta document model is discussed afterwards. After this, properties of the architecture of an extensible authoring tool are presented. A summary concludes this abstract.

## Specification of Multimedia Documents

Application areas for multimedia documents differ mainly in two points. The first point is that each application area has specific interactions during the presentation. In a tourist guide the interaction is the change of the physical location of the user, determined by a GPS receiver for example. In a computer-based training the interaction is answering a question. Besides the interaction also the reaction of the presentation plays an important role. The reaction in a computer-based training is the evaluation of the answer and the explanation of the correct solution if the given answer was wrong. A tourist guide will present information of another sight when the location of the user changes and if he is near enough to a sight.

Based on the described differences of application areas the following requirements for a multimedia document model can be derived:

- *High abstraction level*

  A high abstraction level means that an application area is supported by application-specific abstractions for interactions and reactions. Besides that it has to be ensured that the document model can be used for different application areas.

- *Ad-hoc extensibility*

  During development of a document system it is impossible to consider different kinds of application areas. Therefore it has to be possible to integrate an extension for a new application area in both, the presentation system and the authoring tool. This happens automatically before the presentation of a document.

- *Closeness of the document model*

  This means that the author uses only the document models itself. A change of the authoring paradigm is not necessary (e.g. using scripting languages).

- *Reuse of extensions*

  Once an application area is integrated, it is subject to reuse by other authors. An extension is contained in a module and these modules can be exchanged. The support provided by a module has to be visible in the document model.

The abstraction level and the extensibility are taken as criteria to classify existing approaches. Based on these criteria these approaches are divided into two classes. The first class consists of generic approaches and the second one of application-specific approaches.

- *Generic approaches*

  Generic approaches provide no explicit support for an application area. Application specific relations have to be programmed using a script language. Hence, they can be used for different application areas. Examples for this class include Macromedia MHEG-5 [Mheg95], MHEG-6 [Mheg97], KOAMI [JRT00] from Inria Alpes, or the event-based approach from Vazirgiannis [VaS96].

- *Application-specific approaches*

  Application-specific approaches provide a document model or specification language that is specific for the application area of the approach. Application-specific approaches can be used only for one particular application area and are not extensible to be used in other application areas. Multimedia systems designed for research purposes provide fixed languages [Lig90, HRB93, JLR98, Wir99a, Hos01, SDL+96]. The quality of the support for their application areas has to be considered in each example. Commercial examples for this class include Macromedia Authorware Professional [Aut01] for computer-based trainings and Microsoft PowerPoint [Bes01] for slide-based presentations.

The advantages and disadvantages based on the introduced requirements are summarized in table 1. Because for generic approaches the document model itself is not extensible, generic approaches get a ’0’ as mark.

| | Generic Approaches | Application-specific Approaches | Extensible Approaches |
|---|---|---|---|
| High abstraction level | - | + | + |
| Ad-hoc extensibility | 0 | - | + |
| Closeness | - | + | + |
| Reuse of extensions | - | - | + |

*Table 1 : Comparison of the approaches*

To fulfill the proposed requirements an adequate document model, an extensible presentation system and an extensible authoring tool have to be designed. This is the topic of this thesis.

## An Extensible Meta Document Model

In this section an extensible meta document model is defined that fulfils the proposed requirements of the previous section. Instances of that meta document model are specification languages that are used by authors to create their multimedia document. Different instances are also used in the examples of this thesis.

For the document model an operator-based approach was used. In an operator-based approach, operators are used to specify relations between media items. This means that a document specification is a graph, whose nodes are media items and operators and edges are connections between operators and media items. Operators and media items are visualized graphically and can be interactively manipulated in an authoring tool. This approach is basis of all application specific extensions which minimizes the effort when a document for a different application area has to be created.

Figure 1 shows a clipping of an example specification. The example uses two media items, indicated by rounded rectangles and two operators, indicated by rectangles. The first media item represents an image and the second one an audio clip. The *setPosition* operator defines the location of the image during presentation time, namely 20 pixels below and right of the upper left corner of the presentation area. The second operator defines the temporal layout of the presentation. The image is displayed three seconds before the audio clip is played-out and five seconds after the audio clip finished also the image disappears. See [WaR94] for a detailed specification of temporal interval operators. The last operator also requires the definition which media item is showed first and which one determines the complete duration. Therefore, the notions source media and destination media items are introduced. If a media item is connected with an arrow from the media item to the operator it is the source media item. A media item that is connected with an arrow from the operator to the media item is the destination media. This distinction is also useful for defining which media item is presented in front of another media item. Because it makes a difference if a text is displayed in front of a picture or behind a picture. In the latter case it is invisible.

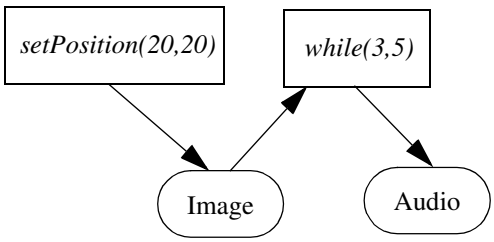


**Figure 1 :** A simple specification

Two extensions of the base model include containers and effect operators. A container can be used to group parts of the presentation. They also have an application specific meaning, for example a container that abstracts from a question or a container that represents a sight in the tourist guide.

Figure 2 shows a specification that uses three containers and two operators. The question container abstracts from a question in a computer-based training and the other two containers represent the reaction depending on the answer. The question container is connected with a *correct* operator with a container that represents a plaudit. Because of that connection, the plaudit is pre-

sented only if the answer to the question was correct. According to that, the question container is also connected with a *false* operator to a container that presents an explanation of the correct answer. The question itself has to be specified too, but we refrain from it here.

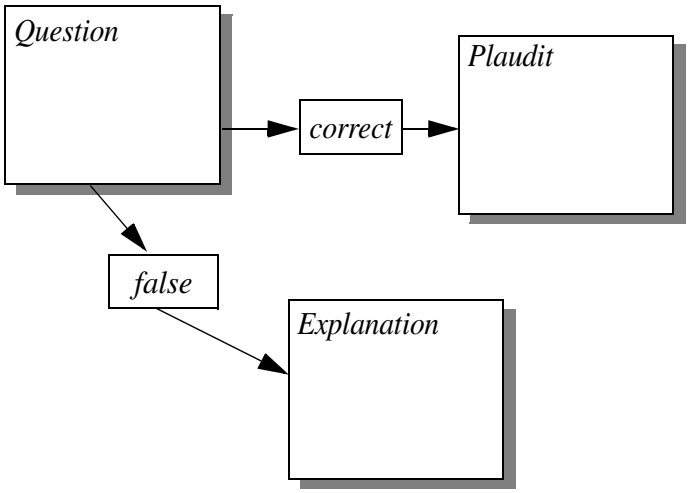


**Figure 2 :** Example of a specification with containers

For the definition and usage of effect operators see reference [HaR01].

## Architecture of an Extensible Multimedia Presentation System

Main topic for the architecture of an extensible multimedia presentation system is how document elements are realized. This thesis focuses on the implementation of operators. Therefore so-called managers are introduced. A manager is responsible for the implementation of all its operators during presentation time.

For the implementation of operators, two different types of events have to be supported by media items. The first event type is the so-called control event and the second event type is the notification event. Control events are used to affect the behavior of media items. Notification events are used by media items to inform managers that something particular has happened (e.g. the mouse button was pressed). Hence, a manager is a program that consumes notification events and generates control events to realize the desired behavior of operators.

Based on the two event types, the notion of a syntactical correct document can be defined. A syntactical correct document has no connections between operators and media items where a

media item does not support the control events or does not provide the required notification events. This ensures that during presentation time a manager receives the notification events it requires and all media items support the control events, i.e. it can realize the operators.

Figure 3 shows the architecture of the extensible multimedia presentation system proposed in this thesis. It includes the following components.

- *Base system*

  The base system provides the basic functionality for the presentation system like loading document, loading extensions, starting and stopping the presentation. It also provides a graphical user interface component in which the presentation takes place.

- *Document loader*

  The document loader parses the document and creates an in-memory representation of the document. The loading of program code is delegated to the class manager.

- *Class manager*

  The class manager manages classes and extensions that are required for the presentation of the document. It downloads extensions and caches them in the local file system for later reuse.

The described components build the framework wherein functional extensions can be integrated. The required extensions are loaded prior to the beginning of the presentation and integrated into the presentation system. All components can exit several times.

The following components are used to represent the document model.

- *Operators*

  Operators have a descriptive functionality for relations between media items. Therefore they contain references to media items.

- *Media objects*

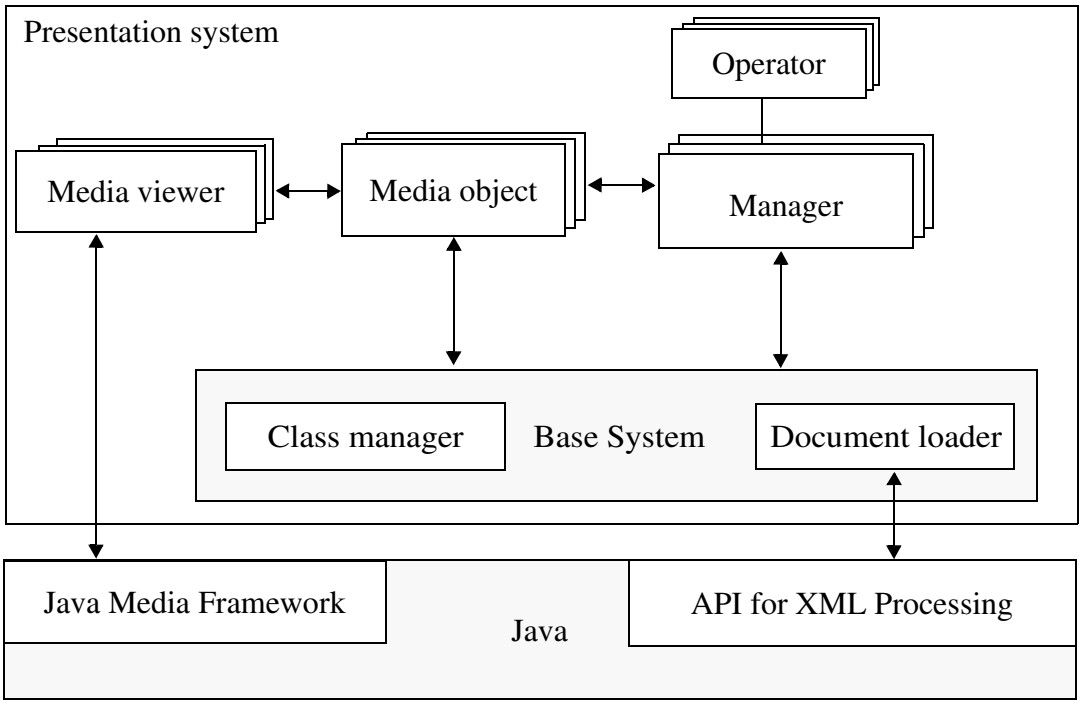


**Figure 3 :** Architecture of an extensible presentation system

Media objects represent media items of the meta document model. They realize the logic of media items during presentation. A check box, for example, has to manage its state (i.e. whether it is checked or not).

The following components are used during the presentation of a document.

- *Manager*

  A manager is responsible for the implementation of its operators during presentation.

- *Media viewer*

  A media viewer is responsible for the play-out of media data during presentation.

The prototype implementation was realized using Java as programming language. Besides the standard libraries, the Java Media Framework (JMF) [Sun01c] and the Java API for XML Processing (JAXP) [Sun01a] were used.

## Dissemination of Multimedia Documents

Multimedia documents specified on the basis of the introduced meta document model have to be stored on servers and transmitted over the Internet to users who want to watch the presentation.

As described in the meta document model section, a document specification consists mainly of media items, operators and the connections between both of them. Because of the connections a document specification is a graph.

To be stored, a document specification is mapped onto XML [BrS98]. The resulting XML file of the document is called transfer format. Two main points have to be considered for this mapping. Firstly, the mapping onto XML of the graph property of the document specification and secondly the linking between the transfer format and the implementation of operators and media items have to be considered. The latter is evaluated to realize the ad-hoc extensibility.

Because an XML document has a tree-based structure the mapping of a document specification has to be defined as follows. First, the containment relation between a container and its media items and operators is considered. This containment relation can be directly mapped on the tree-like structure of XML. The XML tags of a media item (e.g.<image>) or an operator are enclosed by the start and end tag of the corresponding XML tag of the container (e.g. <question>). This is shown in the next example.

```
<question>
   <image />
      ...
</question>
```

The graph structure of media items and the relations between them (i.e. connections between operators and media items) has to be mapped on XML using attributes of the XML tags. Therefore two attributes are used, namely ID and IDREF. The name of the attributes corresponds to their XML type. IDREF is a reference to an XML element that has an ID attribute with a value that corresponds to the value of the IDREF attribute, as seen in the next example.

```
<image ID = "i0001">
```

```
<setPosition>
    <destination IDREF="i0001">
</setPosition>
```

A document has to provide the information which extensions are required for its presentation. Therefore a link between the document elements (operators, media items and containers) and their implementation is required. In the prototype implementation the document elements are implemented using Java. Hence, each document element has an attribute called "class" that is used to reference its implementation. The class attribute is defined as a fixed attribute in the document type definition (DTD) of the document. The following line is an example therefore.

```
<!ATTLIST image class #FIXED "mavax.baseelements.Image">
```

This means that the media item "image" is implemented by the Java class "mavax.baseelements.Image".

All classes of one extension are packed into a so called JAR (Java Archive). The tag <requiredExtension> is used in a document to provide the information which extensions are required for the presentation.

```
<requiredExtension name="Base Media Items"
        package ="basemediaitems.jar"
        ns ="de.uni-stuttgart.informatik.mava" />
```

This means that for the extension "Base Media Items" the jar file is "basemediaitems.jar". To make the name of the extension unique the `ns` attribute (short for name space) is used.

For a complex example see appendix B and appendix C of this thesis. Appendix B shows several screenshots of a document specification. In appendix C the corresponding transfer format including the DTD (document type definition) is presented.

## Architecture of an Extensible Multimedia Authoring System

In this section an authoring system is introduced that can be used to specify documents for the proposed approach. Figure 4 shows a screenshot of the authoring system that is described next.

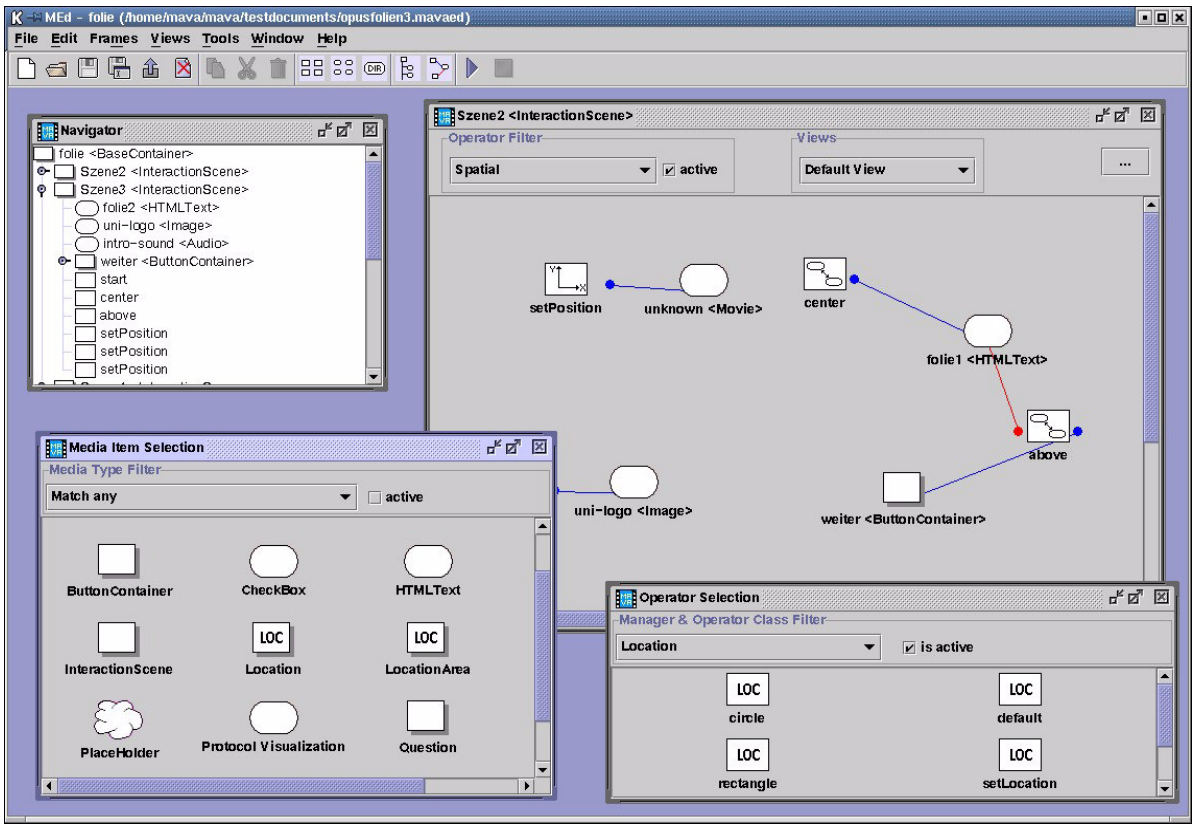


**Figure 4 :** Screenshot of the authoring system

To reduce the complexity of the document authoring process the authoring system has to provide only a few interactions that have to be known and used. The basic functionality includes three different tasks that have to be executed by an author. The first task is adding document elements, like media items or operators, to the document. The second one is defining connections between operators and media items. And the third task is to specify properties of media items and operators. The first two tasks can be performed by using interactive manipulation as defined by [Shn83]. This includes a graphical visualization of the document model and the usage of drag and drop to add new document elements. The third task is performed in a dialog, where parameters can be entered using the keyboard. So called media inspectors can be used to automatically determine the value of parameters for media items.

Besides the basic functionality the authoring system provides support for the author as well. This support includes the visualization of the document structure, i.e. the containment relation between containers and media items and operators. This visualization can be used for navigating

through the document structure. A filtering mechanism is used to cope with an increased number of operators and media items. This filtering mechanism allows to mask out document elements that do not belong to a particular application area. As third concept, a template mechanism is introduced that allows reusing parts of a document specification [Hat01]. As last support, so called application specific views can be added to the authoring system. An application specific view provides a specialized view of a container, that view may also be editable. This figure shows a graphical visualization of operators that are used to specify the location of media item in the presentation area. In this view the parameters of the corresponding operators can be changed by moving the media items in the visualization.

Like the presentation system, the authoring system has to be dynamically extensible for new application areas as well. For this reason, there is a central component in the authoring system called document elements and views repository (DEVR). That repository is responsible for managing the information which extension is integrated and which operators and media items the authoring system provides. It also provides the information which application specific views have been added to the authoring system.

There are two kinds of extensions for the authoring system. The distinction can be made by the fact whether an extensions needs program code or not. Not all document elements for an application area need program code. The properties of each document element are provided in an XML descriptor. The properties include the name of the document element, its attributes and what type of a document element it is. Depending on the type, further information like the number of source and destination elements is provided.

Other extensions to the authoring system, like application specific views or media inspectors, need program code to be realized. In this case the descriptor also includes references to a Java archive and the main Java class implementing the extension.

## Summary

During authoring the application area of a multimedia document plays an import role. Every document is created for a particular application area. Hence, the authoring tool has to support the author in that application area. And there is not only one application area; there are several different application areas. In this thesis two application areas, namely interactive tourist guides

and computer-based training were used as examples. Based on the fact that there are several application areas an extensible approach was proposed. New application areas can be easily integrated into the authoring and presentation system.

The support for an application area was achieved by defining application specific instances of the meta document model. An application specific instance provides a set of language elements that can be used by the author. The most important language element is the operator. The presented approach makes the assumption that every aspect of a multimedia document can be specified by using operators.

The ad-hoc extensibility of the approach means that the implementation of operators is integrated into to presentation system right before the presentation starts. This integration requires no manual user actions. The required information is provided in the document itself. It is stored in a proprietary format because none of the existing formats supports the requirements proposed in this thesis. But the transfer format for documents is based on the open standard XML.

Future work for this approach includes the complete coverage of an application area and an evaluation of the required effort therefore. The prototype implementation gave a first hint that extensions can be programmed with little effort and have less than 2000 lines of code.

The presented approach makes a contribution to the simplification of multimedia authoring by allowing to integrate application-specific support for authors and a defined way how new application areas can be realized and integrated.

# 1 EINLEITUNG

Multimedia-Präsentationen sind bereits in vielen unterschiedlichen Bereichen anzutreffen. Zum einen können einzelne Seiten des World Wide Webs (WWW) als einfache multimediale Präsentationen angesehen werden, zum anderen machen eine Vielzahl von Multimedia-CDs dem Benutzer Multimedia-Präsentationen vertraut. Zukünftig ist mit einer weiteren Verbreitung von multimedialen Präsentationen zu rechnen, so nimmt z.B. die Leistungsfähigkeit von Mobiltelefonen laufend zu, was zukünftig die Ausführung multimedialer Anwendungen erlaubt. Aber auch andere Bereiche werden zunehmend multimedial, so werden immer mehr Medien im Anwendungsbereich computergestütztes Lernen integriert, beispielsweise Animationen oder Filme. Der Benutzer wird es demnach in Zukunft noch häufiger mit interaktiven Multimedia-Präsentationen zu tun bekommen als es bisher der Fall ist. Der weiteren Verbreitung von Multimedia-Präsentationen steht eine erhöhte Nachfrage nach Werkzeugen für ihre Erstellung und nach Personen mit der Fähigkeit sie zu erstellen gegenüber.

Eine Multimedia-Präsentation ist die kombinierte Darstellung unterschiedlicher Medien in einem vorab festgelegten Zusammenhang. In den meisten Fällen nimmt der Betrachter interaktiv an der Präsentation teil, d.h. er kann ihren Verlauf verändern. Dies steht im Gegensatz zum passiven Konsum des Fernsehprogramms. Interaktives Fernsehen, bei dem der Benutzer in das laufende Programm eingreifen oder interaktiv Zusatzinformationen anfordern kann, kann ebenfalls als eine multimediale Präsentation verstanden werden. Die zur Darstellung der Multimedia-Präsentation verwendete Anwendung wird Präsentationssystem genannt.

Im Allgemeinen basieren Multimedia-Präsentationen auf so genannten Multimedia-Dokumenten. Bei der Produktion von Multimedia-Präsentationen werden durch einen Autor Multimedia-Dokumente erstellt. Das Anwendungssystem, das er dafür verwendet, wird als Autorenwerkzeug bezeichnet. Es ist aber auch möglich, Multimedia-Präsentationen in einer Programmiersprache zu programmieren.

In dieser Abhandlung wird besonderes Augenmerk auf die Sicht des potentiellen Autors gelegt. Diese Sicht ist sinnvoll, da beim Konsum einer Präsentation der Aufwand und die Komplexität nicht mehr sichtbar sind. Für den Autor ist der benötigte Zeitaufwand und die Komplexität der Erstellung einer Multimedia-Präsentation von Bedeutung. Die Erstellung zu vereinfachen und

den Zeitaufwand zu reduzieren, wie dies bei der Erstellung von textuellen Dokumenten bereits geschehen ist, ist unabdingbar für eine weitere Verbreitung von Multimedia-Präsentationen und deren Erstellung durch technisch wenig versierte Autoren.

Der triviale Ansatz, eine Multimedia-Präsentation von Grund auf neu zu programmieren, trägt nicht zur Vereinfachung der Dokumentenerstellung bei. Dies liegt zum einen daran, dass der Autor, häufig ein Grafiker oder Designer, diese Fähigkeiten höchstwahrscheinlich nicht besitzt und dass diese Vorgehensweise sehr zeitintensiv ist. Der Autor möchte mit möglichst einfachen Mitteln eine Präsentation erstellen. Dafür werden Modelle und Konzepte benötigt, die sich auf einem höheren Abstraktionsniveau befinden als eine Programmiersprache.

Diese Forderung lässt sich sehr gut mit der Erstellung von Briefen und Büchern vergleichen. Den ersten Versuchen, professionell gestaltete Textdokumente auf Rechnern zu erstellen, lagen sehr einfache Benutzungsschnittstellen zu Grunde. In den Text wurden Formatierungsanweisungen durch spezielle Symbole eingefügt. Die Vorgehensweise ist vergleichbar mit heutigen Auszeichnungssprachen. Das letztendliche Layout des Dokuments war erst nach dem Ausdruck in seiner endgültigen Form sichtbar.

Auf diese Art und Weise werden heute nur noch selten Dokumente erstellt. Heutzutage werden Dokumente direkt auf dem Bildschirm im endgültigen Layout erstellt, diese Vorgehensweise wird mit dem englischen Bergriff Desktop Publishing bezeichnet. In diesem Zusammenhang wird vom What-You-See-Is-What-You-Get-Prinzip (kurz WYSIWYG) gesprochen, da der Ausdruck dem Entwurf am Bildschirm entspricht.

Das gleiche Erscheinungsbild eines Dokuments kann mit einem Texteditor und der Druckeransteuerungssprache PostScript [AdS99] erreicht werden. Dazu muss “nur” der PostScript-Code direkt eingegeben werden, was einen viel höheren Zeitaufwand bedeutet und wesentlich komplizierter ist. Der tatsächliche Erstellungsaufwand ist dem fertigen Ausdruck nicht anzusehen. Für den Benutzer ist es nicht wichtig, in welcher Form der Drucker angesteuert wird. Er möchte eine intuitive Anwendung, die ihm das Schreiben eines Briefes ermöglicht.

An diesem Punkt stellt sich die Frage, wo das Erstellen von Multimedia-Dokumenten im Vergleich zum vorherigen Beispiel eingeordnet werden kann. Es ist durchaus möglich, ein Text-Dokument in der Seitenbeschreibungssprache PostScript von Hand einzugeben. Dies kann mit

der Programmierung eines Multimedia-Dokuments in einer Skript- oder Programmiersprache verglichen werden. Einfacher und schneller geht die Erstellung, wenn eine Multimedia-Präsentation basierend auf einem höheren Dokumentenmodell erstellt wird, das von der Programmierung abstrahiert. Eine weitere Frage ist, inwieweit sich das WYSIWYG-Prinzip auf Multimedia-Dokumente übertragen lässt.

Um diese Fragen im Laufe der Abhandlung zu beantworten, wird in diesem Kapitel ein Überblick über die Erstellung von Multimedia-Dokumenten in unterschiedlichen Anwendungsgebieten gegeben. Im Laufe dieser Abhandlung wird gezeigt, wie diese Anforderungen auch für technisch wenig versierte Autoren erfüllt werden können.

## 1.1 Anwendungsspezifische Multimedia-Präsentationen

Anhand der zwei folgenden Beispiele soll verdeutlicht werden, dass es in unterschiedlichen Anwendungsgebieten auch unterschiedliche Anforderungen bezüglich der Erstellung einer Präsentation gibt. Diesen unterschiedlichen Anforderungen muss sowohl ein multimediales Präsentationssystem als auch ein Autorenwerkzeug gerecht werden.

Als erstes Beispiel wird ein Konzept für computergestütztes Training vorgestellt. Ein wichtiger Bestandteil eines solchen Trainings ist die Überprüfung des in einer Lektion vermittelten Wissens. Dazu werden durch Interaktion mit der Präsentation Fragen durch den Benutzer beantwortet. Ein Beispiel hierfür ist eine Multiple-Choice-Frage, bei der der Benutzer zwischen mehreren Antworten wählen kann. Anschließend erfolgt eine Reaktion auf die Eingabe durch die Präsentation. In Abhängigkeit davon, ob die Frage richtig oder falsch beantwortet wurde, wird die Präsentation an unterschiedlichen Stellen fortgesetzt. Im Fall einer falschen Antwort kann die richtige Lösung erklärt werden, um den Lerneffekt für den Benutzer zu erhöhen.

Als zweites Beispiel kann ein multimedialer Reiseführer herangezogen werden, der einem Touristen Informationen in Abhängigkeit seiner aktuellen Position präsentiert. Durch die Ermittlung der aktuellen Position des Touristen, beispielsweise mit Hilfe eines GPS-Empfängers (Global Positioning System) [HLC92], kann dem Touristen das Suchen und Blättern in einem Reiseführer, der “nur” in Buchform vorliegt, erspart werden. Zudem bekommt er die Informationen über eine Sehenswürdigkeit multimedial aufbereitet präsentiert.

Die beiden oben genannten Beispiele unterscheiden sich in der Art, wie der Benutzer mit der Präsentation interagiert und in welcher Art und Weise die Präsentation darauf reagiert. Im ersten Beispiel war die Interaktion das Beantworten einer Frage und im zweiten Beispiel die physische Veränderungen der aktuellen ortsbezogenen Position. Die Reaktionen der Präsentationen unterscheiden sich ebenfalls. Im ersten Beispiel wird abhängig davon, ob die Frage richtig beantwortet wurde oder nicht, unterschiedlich in der Präsentation fortgefahren. Der Reiseführer aus dem zweiten Beispiel zeigt nur Informationen, die zu Sehenswürdigkeiten am aktuellen Aufenthaltsort passen.

Für die effiziente Erstellung von Multimedia-Dokumenten werden Sprachkonstrukte benötigt, die spezifisch für das jeweilige Anwendungsgebiet sind. Dabei unterscheiden sich die Sprachkonstrukte je nach Anwendungsgebiet beträchtlich. Für jedes der vorangegangenen Szenarien wird eine eigene Spezifikationssprache benötigt. Für die Erstellung der Multimedia-Dokumente müssen die jeweiligen Sprachkonstrukte dem Autorenwerkzeug hinzugefügt werden, und für die Präsentation muss die Implementierung der Sprachkonstrukte dem Präsentationssystem hinzugefügt werden können.

Die Möglichkeit des Hinzufügens von (Sprach-)Erweiterungen wird als Erweiterbarkeit bezeichnet, d.h. ein erweiterbares Multimedia-Präsentationssystem kann nachträglich um die Unterstützung spezieller Anwendungsgebiete erweitert werden. Die Erweiterbarkeit ist eine wichtige Eigenschaft, denn bei der Konzeption und Entwicklung eines Multimedia-Präsentationssystems sind nicht alle Anwendungsgebiete bereits im voraus bekannt. Das gleiche gilt für ein Multimedia-Autorenwerkzeug.

Zusammenfassend lässt sich sagen, dass Multimedia-Dokumente in einem anwendungsspezifischen Kontext erstellt werden. Bei der Modellierung von Multimedia-Dokumenten ist demnach zu betrachten, ob und in welcher Art und Weise unterschiedliche Anwendungsgebiete unterstützt werden.

## 1.2 MAVA - Multimedia Document Versatile Architecture

Die in dieser Abhandlung beschriebenen Modelle, Konzepte und deren prototypische Implementierung wurden im Rahmen des DFG-Projekts MAVA (*M*ultimedi*a* Document *V*ersatile

*A*rchitecture) [Hau99, HaR00, Hau00, HaR01, Hau01, HaT01] am Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR) der Universität Stuttgart entwickelt. Das MAVA-Projekt war Teil des DFG-Schwerpunktprogramms “Verteilte Verarbeitung und Vermittlung Digitaler Dokumente (V3D2)” [V3D2], das in über 20 Projekten die Forschung im Umfeld digitaler Bibliotheken vorangetrieben hat. Aspekte digitaler Bibliotheken wurden durch die Kooperation mit der Universitätsbibliothek Stuttgart untersucht und veröffentlicht [HSA99, HaS01].

Es wurde ein Meta-Dokumentenmodell entwickelt, auf dem Spezifikationssprachen für Anwendungsgebiete basieren. Für jedes Anwendungsgebiet wird eine Spezifikationssprache verwendet, die eine Instanz des Meta-Dokumentenmodells ist.

An dieses Meta-Dokumentenmodell werden Anforderungen bezüglich dessen Eigenschaften gestellt. Eine Eigenschaft ist zum Beispiel die Unterstützung eines hohen Abstraktionsniveaus, um anwendungsspezifische Beziehungen zwischen Medienelementen abbilden zu können. Eine weitere Anforderung ist die Wiederverwendbarkeit von Erweiterungen für spezielle Anwendungsgebiete. Basierend auf diesem Meta-Dokumentenmodell sind Spezifikationssprachen für alle Anwendungsgebiete möglich.

Im MAVA-Projekt wurde als Meta-Dokumentenmodell ein auf Operatoren basierendes Modell gewählt. Dabei handelt es sich um eine Verallgemeinerung des Ansatzes aus dem TIEMPO-Projekt [Wir99a, Wir99b], welches ebenfalls am Institut für Parallele und Verteilte Höchstleistungsrechner durchgeführt wurde. In TIEMPO gibt es zehn Operatoren, die das zugrunde liegende temporale Intervallmodel zur Beschreibung des zeitlichen Ablaufs von Multimedia-Präsentationen repräsentieren. Ziel des TIEMPO-Projekts war die Spezifikation adaptiver Multimedia-Dokumente. In MAVA werden die Operatoren für die Beschreibung aller Beziehungen zwischen Medienelementen herangezogen, insbesondere der anwendungsspezifischen Beziehungen.

Im Verlauf des MAVA-Projekts wurde ein erweiterbares Präsentationssystem und das entsprechende erweiterbare Autorenwerkzeug entwickelt und implementiert. Die betrachteten Fragestellungen umfassen die Umsetzung des Meta-Dokumentenmodells und den Aufbau der Architekturen des erweiterbaren Präsentationssystems und des erweiterbaren Autorenwerkzeugs. Um Dokumente zwischen dem Autorenwerkzeug und dem Präsentationssystem und letztendlich

übers Internet zur übertragen, wurde ein Transferformat entwickelt, dass ebenfalls die Erweiterbarkeit unterstützt. Bei der Entwicklung eines Autorenwerkzeugs wurden zusätzliche Konzepte zur weiteren Vereinfachung der Spezifikation untersucht.

Für die Implementierung wurde Java als Programmiersprache [Arg96, Fla00, CaW01] und die Auszeichnungssprache XML (E*x*tensible *M*arkup *L*anguage) [BrS98] zur Speicherung von Datenobjekten (Dokumente und Konfigurationen des Editors und des Präsentationssystems) herangezogen.

Die Erfüllung spezieller Anforderungen führte zur Wahl von Java und XML. Für die Realisierung der Erweiterbarkeit wurde auf die Plattformunabhängigkeit und den dynamischen Klassenlader von Java zurückgegriffen. Die Möglichkeit, mittels XML eigene Auszeichnungssprachen zu definieren, eignet sich hervorragend, die unterschiedlichen Spezifikationssprachen der einzelnen Anwendungsgebiete auf ein gemeinsames Transferformat abzubilden.

Die Plattformunabhängigkeit der Implementierung sichert die Verfügbarkeit des Autorenwerkzeugs und des Präsentationssystems für eine Vielzahl von Plattformen. Die Funktionsfähigkeit des Prototypen konnte im Rahmen des MAVA-Projekts unter Linux, Sun Solaris und den unterschiedlichen Microsoft Windows-Varianten demonstriert werden. Die erstellten Dokumenten konnten ohne Veränderungen auf den verschiedenen Plattformen präsentiert werden.

An dieser Stelle soll nicht verschwiegen werden, dass trotz der Plattformunabhängigkeit von Java eine Java-Laufzeitumgebung für die einzelnen Plattformen vorhanden sein muss. Zusätzlich traten kleinere Inkompatibilitäten der Java-Laufzeitumgebungen der unterschiedlichen Plattformen auf, die bewältigt werden mussten.

Waren Bedenken bezüglich der Ausführungsgeschwindigkeit zur Anfangszeit von Java noch vorhanden, so hat die Weiterentwicklung der Java-Laufzeitumgebung, nicht zu letzt durch Just-In-Time-Compiler, große Fortschritte gemacht. Plattformabhängige “Performance-Packs” steigern die Leistung von Java-Programmen weiter. Abschließend sei noch angemerkt, dass in Multimedia-Dokumenten gewöhnlich keine größere Anzahl von Video- und Audiomedien gleichzeitig angezeigt werden.

Die Verwendung des offenen Standards XML für die Speicherung von Dokumenten ermöglicht deren Weiterverarbeitung durch andere Werkzeuge. Durch die Kooperation mit der Universitätsbibliothek wurde aufgezeigt, wie durch die Einbettung von Meta-Daten, basierend auf dem DL-Meta-Standard, die Weiterverarbeitung in einem Nachweissystem für digitale Bibliotheken ermöglicht wird [HSA99, HaS01].

## 1.3 Aufbau der Abhandlung

Kapitel 2 befasst sich mit der Beschreibung der Anforderungen an die Erstellung anwendungsspezifischer Multimedia-Dokumente. Dafür wird zuerst der Modellierungsprozess von Multimedia-Dokumenten dargestellt. Nach der Beschreibung der Anforderungen werden existierende Ansätze im Detail untersucht und bezüglich der Anforderungen bewertet.

In Kapitel 3 wird basierend auf den Untersuchungsergebnissen aus dem vorangegangenen Kapitel ein Meta-Dokumentenmodell entwickelt. Dieses Meta-Dokumentenmodell bildet die Grundlage für ein erweiterbares Multimedia-Präsentationssystem und ein erweiterbares Autorenwerkzeug. Beide werden in den folgenden Kapiteln der Abhandlung konzipiert und umgesetzt.

Das Kapitel 4 beschreibt die Architektur eines erweiterbaren Präsentationssystems, das auf dem Meta-Dokumentenmodell basiert. In einer Diskussion werden Alternativen für die Implementierung von Spezifikationssprachen bewertet. Eine Eigenschaft des erweiterbaren Ansatzes ist die ad-hoc Erweiterbarkeit. Darunter wird das automatische Nachladen der Erweiterungen verstanden, die zur Präsentation des aktuellen Dokuments benötigt werden.

Kapitel 5 beschreibt die Vermittlung von Multimedia-Dokumenten, die auf dem in Kapitel 3 eingeführten Meta-Dokumentenmodell basieren. Die Vermittlung beschreibt die Übertragung und Bereitstellung von Dokumenten auf einem Server. Dazu muss ein Transferformat definiert werden, das die Erweiterbarkeit des Meta-Dokumentenmodells unterstützt. Die Speicherung von Dokumenten auf Internet-Servern wird ebenfalls betrachtet, da für die Präsentation eines Dokuments zusätzlich die Erweiterungen für das Präsentationssystem benötigt werden.

In Kapitel 6 wird die Architektur eines erweiterbaren Autorenwerkzeugs erläutert. Es werden die Benutzungsschnittstelle und die notwendigen Interaktionsschritte betrachtet, die für die Erstellung eines Multimedia-Dokuments benötigt werden. Zusätzlich werden Konzepte eingeführt, die die notwendige Zeit und Komplexität der Dokumenten-Erstellung weiter reduzieren. Anhand der Architektur des Autorenwerkzeugs wird die Umsetzung der Erweiterbarkeit diskutiert.

In Kapitel 7 werden die Ergebnisse dieser Abhandlung zusammengefasst. In einem Ausblick wird angedeutet, in welche Richtung die Forschung bezüglich der Erstellung multimedialer Dokumente sich weiterentwickeln kann.

# 2 SPEZIFIKATION VON MULTIMEDIA-DOKUMENTEN

Grundlage der effizienten Erstellung von Multimedia-Präsentationen sind Multimedia-Dokumente. Bei der Spezifikation eines Multimedia-Dokuments durch einen Autor werden unterschiedliche Medien miteinander in Beziehung gesetzt. Die Spezifikation der Beziehungen erfolgt dabei basierend auf einem Dokumentenmodell, das den strukturellen Aufbau eines Multimedia-Dokuments beschreibt.

In diesem Kapitel wird in die Modellierung von Multimedia-Dokumenten eingeführt. Außerdem werden die Anforderungen an die Unterstützung des Autors bei der Erstellung von Multimedia-Dokumenten in unterschiedlichen Anwendungsgebieten erörtert. Anschließend wird überprüft, inwieweit existierende Ansätze diese Anforderungen erfüllen.

## 2.1 Multimedia-Dokumente

Die Präsentation von Dokumenten dient dazu, einem Benutzer Informationen zu vermitteln. Hierbei können zwei verschiedene Dokumententypen unterschieden werden:

- *Statische Dokumente*

  Statische Dokumente besitzen keine zeitlichen Abhängigkeiten während ihrer Darstellung. Sie werden häufig am Rechner erstellt und anschließend in gedruckter Form weitergegeben.

- *Multimedia-Dokumente*

  Multimedia-Dokumente sind Dokumente, deren Darstellung von der Präsentationszeit abhängt. Dementsprechend werden sie vorwiegend durch einen Rechner präsentiert.

Die in Dokumenten verwendeten Medien können in zwei Klassen eingeteilt werden [Ste99].

- *Diskrete Medien*

  Das sind Medien, die bei Ihrer Darstellung zeitunabhängig sind. Das bedeutet, dass die Darstellung ihrer Mediendaten sich bei fortschreitender Zeit nicht verändert. Ein Beispiel für Medien dieser Klasse sind Bilder. Die Mediendaten eines Bilds können in unterschiedlichen Formaten gespeichert werden, beispielsweise JPEG [HYS88] oder Portable Network Graphics (PNG) [Png95].

- *Kontinuierliche Medien*

  Dies sind zeitabhängige Medien, deren Darstellung sich mit dem Verlauf der Zeit ändert. Welcher Teil der Mediendaten dargestellt wird, hängt vom konkreten Zeitpunkt ab. Beispiel für Medien dieser Klasse sind Video- und Audiosequenzen. Die Mediendaten von Videosequenzen liegen in unterschiedlichen Formaten vor, häufig wird das MPEG-2-Format [ISO93, Gal91] gewählt. Audiosequenzen liegen häufig im MP3-Format [Bra99] vor.

Statische Dokumente unterscheiden sich von Multimedia-Dokumenten dadurch, dass sie keine kontinuierlichen Medien verwenden, alle Medien sofort angezeigt werden und sich ihre räumliche Anordnung nicht im Verlauf der Präsentationszeit verändert.

Multimedia-Dokumente können entweder interaktiv oder nicht-interaktiv sein. In dieser Abhandlung werden speziell die interaktiven Multimedia-Dokumente betrachtet. Interaktiv bedeutet, dass der Benutzer während der Präsentation in deren Verlauf eingreifen und ihn verändern kann. In [Wir02] werden die unterschiedlichen Arten von Interaktionen in die folgenden Klassen eingeteilt: Räumliche Interaktionen, gestalterische Interaktionen, zeitliche Interaktionen und strukturverändernde Interaktionen.

Das Erstellen eines Multimedia-Dokuments wird häufig mit dem englischen Begriff Authoring bezeichnet. Die dazu verwendete Anwendung wird Autorenwerkzeug genannt. Das Abspielen eines Multimedia-Dokuments wird Multimedia-Präsentation genannt und durch ein Präsentationssystem durchgeführt. Unter einem Dokumentensystem wird in dieser Abhandlung das Autorenwerkzeug und das zugehörige Präsentationssystem verstanden.

Aus Sicht von Multimedia-Dokumenten sind Medien atomare Einheiten, die Mediendaten eines bestimmten Formats repräsentieren. Auf Grund dieser Abstraktion von Medienformaten und Mediendaten erfolgt die Bearbeitung der Mediendaten meist nicht mittels Autorenwerkzeug sondern in speziellen Anwendungen, beispielsweise muss ein Video-Medium für seine Verwendung in einer Multimedia-Präsentation fertig geschnitten vorliegen.

Neben den Medien selbst werden Beziehungen zwischen den Medienelementen dazu verwendet, den Ablauf einer Präsentation zu beschreiben. Diese Beziehungen beschreiben den strukturellen Aufbau eines Multimedia-Dokuments. Sie werden in den nächsten Abschnitten genauer betrachtet.

## 2.2 Anforderungsanalyse für ein Dokumentenmodell

In allen Dokumentenmodellen werden die Mediendaten und Eigenschaften des zugehörigen Medientyps durch so genannte Medienelemente zusammengefasst und repräsentiert. Große Unterschiede gibt es bei der Beschreibung der Beziehungen zwischen Medienelementen. An dieser Stelle zeigt sich, wie gut die Unterstützung unterschiedlicher Anwendungsgebiete durch das Dokumentenmodell tatsächlich ist.

Die Hauptanforderung an ein Dokumentenmodell ist die Unterstützung der Spezifikation von Multimedia-Dokumenten in unterschiedlichen Anwendungsgebieten, wie etwa in den in der Einleitung vorgestellten Beispielanwendungen. Dazu wird zunächst untersucht, wie sich Anwendungsgebiete bei der Spezifikation von Multimedia-Dokumenten unterscheiden.

Wichtig bei der Betrachtung von Anwendungsgebieten ist, dass von Teilspezifikationen abstrahiert werden kann. Ein Beispiel für solch eine Abstraktion ist eine Frage in einem computergestützten Training. Diese kann aus Texten, Bildern, Texteingabefeldern oder Kontrollkästchen und den erforderlichen Beziehungen aufgebaut werden. Mit Hilfe von Kontrollkästchen lassen sich beispielsweise Multiple-Choice-Frage spezifizieren.

Basierend auf diesen anwendungsspezifischen Abstraktionen lassen sich Beziehungen erkennen, die ebenfalls typisch für das Anwendungsgebiet sind. Um diese Frage zu erörtern, werden zunächst Interaktionen näher betrachtet.

Wie eine Interaktion durchgeführt wird und welche Semantik sie besitzt, ist vom Anwendungsgebiet abhängig. In vielen Fällen wird die Interaktion mit der Maus oder der Tastatur durchgeführt. Das muss aber nicht immer so sein, beispielsweise kann die Interaktion bei interaktivem Fernsehen durch eine externe Fernbedienung erfolgen oder bei einem Multimedia-Reiseführer durch die Lauf- oder Fahrbewegung des Benutzers, die die GPS-Position verändert. Die Interaktionen des Benutzers während der Präsentation eines computergestützten Trainings werden beispielsweise durch die Verwendung der Maus zur Auswahl von Kontrollkästchen einer Multiple-Choice-Frage durchgeführt. Die Semantik der Interaktion ist in diesem letzten Fall die Beantwortung einer Frage.

Neben der Interaktion ist die Reaktion auf eine Interaktion der Multimedia-Präsentation vom Anwendungsgebiet abhängig. So hängt der Verlauf eines computergestützten Trainings davon ab, ob die gestellten Fragen richtig oder falsch beantwortet werden. Im Falle des Multimedia-Reiseführers entspricht eine Positionsänderung des Benutzers einem Sprung an eine andere Stelle in der Präsentation.

Damit lassen sich Anwendungsgebiete für Multimedia-Dokumente durch die Ausprägung der folgenden zwei Punkte unterscheiden und klassifizieren.

- *Interaktionsmöglichkeiten*

  Interaktionsmöglichkeiten sind die Möglichkeiten, die der Benutzer hat, die Präsentation eines Multimedia-Dokuments durch Interaktionen zu beeinflussen. Dabei müssen die Art, wie die Interaktion durch den Benutzer ausgeführt wird (beispielsweise das Drücken der Maustaste oder das Ändern der eigenen räumlichen Position) und die Bedeutung der Interaktion in dem jeweiligen Anwendungsgebiet (beispielsweise die Beantwortung einer Frage oder Bewegung auf einer Landkarte) beachtet werden.

- *Reaktionen der Präsentation auf Interaktion*

  Nach einer erfolgten Interaktion unterscheiden sich die Reaktionen einer Präsentation entsprechend dem Anwendungsgebiet. Dabei umfasst die Reaktion die Auswertung der Interaktion (beispielsweise die Auswertung einer Antwort auf eine Frage in einem computergestützten Training) und eine davon abhängige Entscheidung, wie in der Präsentation fort-

gefahren werden soll (beispielsweise die Erklärung der richtigen Lösung bei einer falschen Antwort auf eine Frage).

Nachfolgend werden die Anforderungen an ein Dokumentenmodell beschrieben. Diese ergeben sich aus der Forderung der Unterstützung von unterschiedlichen Anwendungsgebieten.

- *Hohes Abstraktionsniveau*

  Die wichtigste Anforderung an ein Dokumentenmodell ist die Unterstützung der Eigenschaften unterschiedlicher Anwendungsgebiete. Diese Eigenschaften umfassen anwendungsspezifische Abstraktionen und Beziehungen. Gleichzeitig ist die Unabhängigkeit des Dokumentenmodells von speziellen Anwendungsgebieten zu wahren, denn es sollen Multimedia-Dokumente in unterschiedlichen Anwendungsgebieten mit dem selben Dokumentenmodell spezifiziert werden können.

- *Ad-hoc Erweiterbarkeit*

  Zur Entwicklungszeit des Dokumentenmodells und des Dokumentensystems sind und können noch nicht alle möglichen Anwendungsgebiete erschlossen und integriert werden. Durch die Erweiterbarkeit wird die Möglichkeit gegeben, bei Bedarf ein neues Anwendungsgebiet zu integrieren. Dabei werden benötigte Erweiterungen erst kurz vor ihrer Verwendung dem Dokumentensystem hinzugefügt. Dies soll als ad-hoc Erweiterbarkeit bezeichnet werden.

- *Abgeschlossenheit*

  Unter Abgeschlossenheit des Dokumentenmodells wird verstanden, dass der Autor sein Dokument nur auf diesem Modell basierend erstellt. Ist diese Eigenschaft gegeben, muss beispielsweise nicht während der Spezifikation des Dokuments vom Dokumentenmodell zur Skriptprogrammierung gewechselt werden.

- *Wiederverwendbarkeit von Erweiterungen*

  Eine einmal erschlossene und integrierte anwendungsspezifische Unterstützung soll durch andere Autoren wiederverwendet werden können. Dazu müssen anwendungsspezifische Erweiterungen in einer Art Modul zusammengefasst werden können. Diese Module müs-

sen austauschbar sein. Auf Ebene des Dokumentenmodells müssen die durch das Modul zur Verfügung gestellten anwendungsspezifischen Konzepte sichtbar werden.

Nachdem die Anforderungen an das Dokumentenmodell beschrieben wurden, werden im nächsten Abschnitt existierende Ansätze untersucht. Eine Bewertung der existierenden Ansätze wird anhand der Erfüllung der vier vorher genannten Anforderungen vorgenommen.

## 2.3 Existierende Ansätze

Werden die Anforderungen an das Abstraktionsniveau und die Erweiterbarkeit als Kriterien für eine Klassifikation von Ansätzen gewählt, dann lassen sich existierende Ansätze für die Modellierung von Multimedia-Dokumenten in zwei Klassen einteilen.

- *Generische Ansätze*
  Generische Ansätze bieten keine explizite Unterstützung für ein bestimmtes Anwendungsgebiet. Anwendungsspezifische Beziehungen müssen basierend auf einer Skriptsprache programmiert werden. Sie können damit für unterschiedliche Anwendungsgebiete verwendet werden.

- *Anwendungsspezifische Ansätze*
  Anwendungsspezifische Ansätze sind dadurch charakterisiert, dass sie ein Dokumentenmodell und Sprachelemente für ein Anwendungsgebiet fest vorgeben. Damit sind sie auf die Verwendung für ein Anwendungsgebiet spezialisiert und nicht für neue Anwendungsgebiete erweiterbar.

Im Folgenden werden beide Klassen näher beschrieben und die Bewertung durchgeführt.

### 2.3.1 Generische Ansätze

Ein generischer Ansatz bietet ein Spezifikationsmodell, das für kein bestimmtes Anwendungsgebiete spezialisiert ist. Beispielsweise bietet der Macromedia Director [Dir01] oder Macromedia Flash [Fla99] eine Zeitachse an. Zur Realisierung anwendungsspezifischer Funktionalität wird zusätzlich eine Skriptsprache zur Verfügung gestellt. Die benötigte anwendungsspezifische Funktionalität muss in dieser Skriptsprache programmiert werden. Damit ermöglichen

Ansätze dieser Klasse die Erstellung von Multimedia-Dokumenten aus den unterschiedlichsten Anwendungsgebieten.

Abbildung 2-1 stellt die Beziehung zwischen einem generischen System und Dokumenten aus Anwendungsgebieten graphisch dar. Die Dokumente aus den beiden Anwendungsbeispielen basieren dabei auf dem selben generischen Multimedia-Dokumentensystem. Die anwendungsspezifische Funktionalität ist als Skriptprogramme in den Dokumenten selbst enthalten.

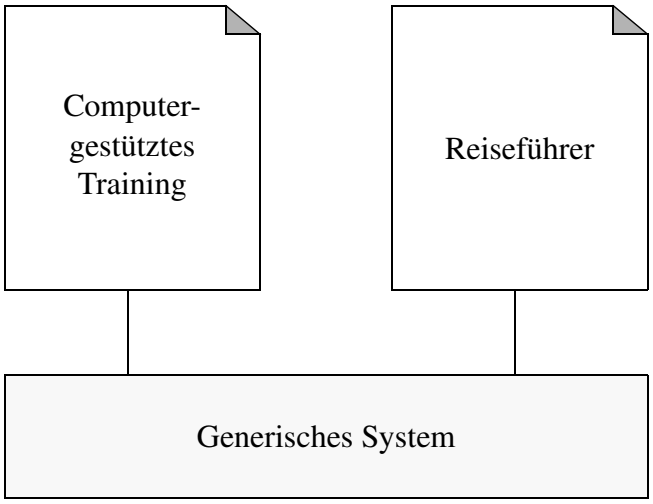


**Abb. 2-1 :** Architektur generischer Ansätze

*2.3.1.1 Beispiele*

In diese Klasse von Ansätzen fallen beispielweise das kommerzielle Autoren- und Präsentationssystem Director von der Firma Macromedia, alle Ansätze, die auf MHEG-5 [Mheg95] und MHEG-6 [Mheg97] basieren, der Ansatz KAOMI [JRT00] von Inria Alpes und der ereignisbasierte Ansatz von Vazirgiannis [VaS96].

An dieser Stelle sei noch ScriptX erwähnt. Dieser kommerzielle Ansatz der Firma Kaleida Labs hat sich von vornherein darauf konzentriert, eine Skriptsprache mit dazugehörigen Bibliotheken zu entwickeln, die für die Programmierung multimedialer Präsentationen besonders gut geeignet ist. Der ScriptX-Ansatz hat es aber nie bis zu Marktreife geschafft und seine Weiterentwicklung wurde mittlerweile eingestellt.

Macromedia Director bietet für die zeitliche Anordnung von Medienelementen eine Zeitachse an. Die räumliche Anordnung erfolgt durch eine absolute Positionierung innerhalb der Präsentationsfläche, im Director „stage“ (Bühne) genannt. Interaktionen und sonstige Funktionalitäten werden in der Skriptsprache des Macromedia Directors, genannt Lingo, programmiert.

Um den Aufwand bei der Programmierung zu reduzieren, wurden im Macromedia Director so genannte Verhaltensweisen (engl. behaviors) eingeführt. Einzelnen Medienelemente können eine oder mehrere Verhaltensweisen hinzugefügt werden. Verhaltensweisen ermöglichen somit die Wiederverwendung von Skriptprogrammen. Beispielsweise kann die Reaktion auf das Auswählen einer Schaltfläche bei der Präsentation als Verhaltensweise immer wieder verwendet werden und muss somit nicht jedes Mal neu programmiert werden. Aus Sicht eines Programmierers stellen die Verhaltenweisen eine Programmbibliothek dar.

Weitere Beispiele für diese Klasse sind alle Ansätze, die auf der Verwendung von MHEG-5 [Mheg95] mit einer Skriptsprache basieren. Bei MHEG-5 kann, genauso wie beim Director, das zeitliche und räumliche Verhalten in dem dafür vorgesehenen Modell spezifiziert werden. Für die Spezifikation von anwendungsspezifischer Funktionalität wird durch MHEG-6 definiert, wie Java als Skriptsprache in MHEG-5-Präsentationssysteme zu integrieren ist und welche JAVA-APIs bereitzustellen sind. Anwendungsspezifische Funktionalität muss vom Autor also in Java programmiert werden. Bei diesen Ansätzen können Java-Programmbibliotheken, so genannte Java-Archive, zur Wiederverwendung von Java-Programmcode verwendet werden.

Mit KAOMI wird in [JRT00] vorgeschlagen, verschiedene temporale und räumliche Konzepte auf ein generisches Modell abzubilden und Multimedia-Dokumente basierend auf diesem generischen Modell zu spezifizieren. Das generische Modell basiert auf der Beschreibung von Randbedingungen, durch deren Erfüllung das temporale und räumliche Layout eines Multimedia-Dokuments bestimmt wird. Für das temporale Layout sind die Randbedingungen die temporalen Intervall-Operatoren nach Allen [All83]. Dieser Ansatz beschränkt sich auf temporale und räumliche Konzepte und betrachtet nicht die Unterstützung beliebiger Anwendungsgebiete für Multimedia-Dokumente.

In [VaS96] schlagen Vazirgiannis et al. vor, Multimedia-Dokumente basierend auf Ereignissen und deren Verarbeitung zu spezifizieren. Durch das Hinzufügen neuer Ereignisse ist dieser

Ansatz generisch. Die Ereignisverarbeitung muss aber für neue Ereignisse ausprogrammiert werden. Die genaue Umsetzung wird nicht beschrieben. Zudem ist die von Vazirgiannis vorgeschlagene Ereignisverarbeitung mit dem Link-Action-Mechanismus von MHEG vergleichbar.

Um den Autor bei der Spezifikation zu unterstützen, wird bei generischen Ansätzen auf die Wiederverwendung von Programmcode gesetzt. Dies ist bei den beschriebenen Vertretern zu sehen. Die Wiederverwendung von Programmcode in Bibliotheksform ist ein bewährtes Prinzip der Programmierung. Vergleicht man dies mit der Modellierung von Multimedia-Dokumenten, so ist das Modell die Programmiersprache und die Programmierung die Spezifikation. Durch Bibliothekskonzepte wird die Programmierung vereinfacht, aber kein Modell für Multimedia-Dokumente eingeführt.

#### *2.3.1.2 Bewertung*

Der Vorteil einer Dokumentenspezifikation, die auf einem Modell mit hohem Abstraktionsniveau basiert, soll am Vergleich der Auswertung einer Frage in einem computergestützten Training gezeigt werden.

Für die Präsentation einer Frage wurde ein bestimmter Abschnitt auf einer Zeitachse gewählt. Im Zeitachsenmodell ist nicht zu erkennen, dass es sich dabei um eine Frage handelt. Bei der Spezifikation der Frage hat der Autor für jedes Kontrollkästchen festgelegt, ob es für die richtige Lösung selektiert sein muss oder nicht. Dies geschah durch die Festlegung des Wertes der Variable `element.solution` für jedes Kontrollkästchen. Die Semantik der Variablen wird nur durch die Auswertungsfunktion bestimmt.

In Abbildung 2-2 ist zu sehen, wie in Pseudo-Code die Auswertung einer Frage spezifiziert werden kann. Die dargestellte Auswertungsfunktion soll bestimmen, ob eine Multiple-Choice-Frage richtig beantwortet wurde und in Abhängigkeit davon, falls die Antwort richtig war, an der Position `t0` der Zeitachse die Präsentation fortsetzten oder im Fall einer falschen Antwort an Position `t1` fortfahren.

Die Auswertungsfunktion (`Evaluate_Question`) prüft in einer Schleife für jedes Element, ob es den vom Autor geforderten Zustand besitzt. Der Autor muss während dem Editieren zusätzlich noch festlegen, wann die Auswertungsfunktion aufgerufen werden soll. Falls er keine

Und-Verknüpfung der richtigen Lösung (alle Kontrollkästchen müssen wie vorgegeben ausgewählt sein) haben möchte, muss eine andere Auswertungsfunktion (die beispielsweise eine Oder-Verknüpfung realisiert, bei der es reicht, wenn eines der geforderten Kontrollkästchen ausgewählt ist) verwendet, beziehungsweise programmiert werden.

```
Evaluate_Question(t0, t1)
{
   boolean answer = true;
   cbt-elements = get-related-cbt-elements();
   for all element in cbt-elements do {
      if (element.state != element.solution) {
         answer = false;
      }
   }
   if (answer) {
      goto t0;
   } else {
      goto t1;
   }
}
```

**Abb. 2-2 :** Skript zum Auswerten einer Frage

Die von Autoren programmierten Funktionen sind in den meisten Fällen Speziallösungen für genau den einen Fall, in dem sie benötigt werden. Das erschwert ihre Wiederverwendung in anderen Dokumenten. Versucht man, die Funktionalität allgemeiner zu beschreiben, liegt es nahe, ein zweites Modell für spezielle Konzepte des Anwendungsgebiets einzuführen. Es wird damit ein Dokumentenmodell in der Skriptsprache des Ursprungssystems entwickelt, das durch die graphisch interaktive Benutzungsschnittstelle des Ursprungssystems nicht unterstützt wird. Letztendlich wird die Skriptsprache als Implementierungssprache eines neuen Autorenwerkzeugs verwendet und das ursprüngliche Modell und Autorenwerkzeug nicht mehr verwendet.

Ein Spezifikationsmodell auf höherem Abstraktionsniveau ermöglicht dem Autor, die Frage und die Reaktion deskriptiv zu spezifizieren (Abbildung 2-3). Er legt dabei nur fest, welches die korrekte Lösung ist und in welchem Teil der Präsentation entsprechend einer richtigen oder falschen Antwort fortgefahren werden soll. Um die Auswertung und der Zeitpunkt muss er sich nicht kümmern, dies wird automatisch während der Präsentation des Dokuments vom Präsentationssystem bestimmt. Über einen Parameter kann der Autor entscheiden, ob bei der Auswertung eine Und- oder eine Oder-Verknüpfung der möglichen Lösungen verwendet werden soll.

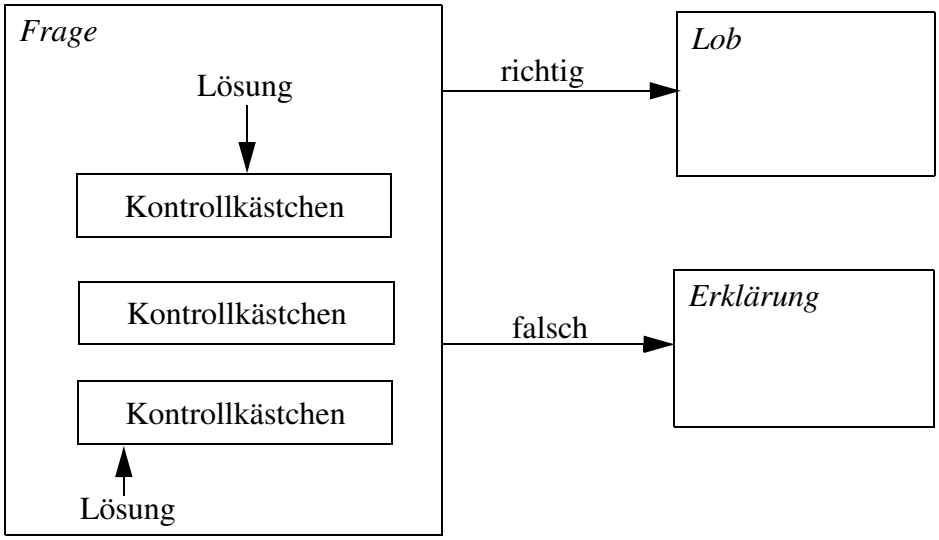


**Abb. 2-3 :** Deskriptive Beschreibung einer Frage und der Reaktion

Durch die Programmierung von Skripten kann neue anwendungsspezifische Funktionalität hinzugefügt und somit Multimedia-Präsentationen für unterschiedliche Anwendungsgebiete erstellt werden. Dadurch wird aber keine Unterstützung in Form von Abstraktionen für bestimmte Anwendungsgebiete geboten. Zusätzlich werden vom Autor Programmierkenntnisse gefordert. Durch die Kombination eines einfachen Spezifikationsmodells und komplexer Programmierung entsteht der Nachteil des hohen Aufwands bei der Erstellung eines Dokuments.

In der Praxis wird heutzutage die Erstellung von Multimedia-Dokumenten oft von Programmierern erledigt, was z.B. an der Vielfalt von Büchern über die Skriptsprache Lingo [Cal96, NyM00, Klo00] und der weiten Verbreitung des Macromedia Directors und der Nachfrage nach Lingo-Programmierern von Design-Agenturen ersichtlich wird.

Gerade die Einfachheit der Dokumenterstellung ist für die Praxis ein entscheidendes Argument, da man nicht davon ausgehen sollte, dass Multimedia-Dokumente von Programmierern erstellt werden. Deshalb ist es wichtig, ein Modell für die Spezifikation anzubieten, das hohe Abstraktionen in Anwendungsgebieten ermöglicht.

Ein weiterer wichtiger Punkt ist die Wiederverwendung von einmal entwickelten Skripten. Dafür wurden unterschiedliche Konzepte (beispielsweise Verhaltensweisen im Macromedia Director) zur Realisierung von Skriptbibliotheken eingeführt. Dies erfordert vom Autor zusätz-

lich Disziplin bei der Erstellung der Skripte. Für eine Wiederverwendung müssen die Skripte so programmiert werden, dass sie leicht in anderen Zusammenhänge wiederverwendbar sind. Meistens sind Skripte jedoch Speziallösungen für einen konkreten Fall und dazu noch im Dokument selbst gespeichert, was ihre Wiederverwendung erschwert.

Somit wird die Forderung nach Erweiterbarkeit nur eingeschränkt erfüllt. Die Anforderung besagt, dass ein Modell auf hohem Abstraktionsniveau erweiterbar sein soll. Bei generischen Ansätzen gibt es dieses Modell nicht. Es lässt sich aber anwendungsspezifische Funktionalität durch Skriptprogramme hinzufügen. Die Vermischung von Dokumentenspezifikation und programmierter anwendungsspezifischer Funktionalität hat zwei Nachteile: Sie erschwert deren Wiederverwendung und die Spezifikation findet nicht in einem abgeschlossenen Modell statt. Durch eine klare Trennung können diese beiden Nachteile vermieden werden.

### 2.3.2 Anwendungsspezifische Ansätze

Ein anwendungsspezifischer Ansatz stellt sowohl ein Spezifikationsmodell als auch ein Autorenwerkzeug zur Verfügung, die beide auf ein bestimmtes Anwendungsgebiet spezialisiert sind. Während dem Erstellen eines Dokuments kann nur auf die vorgegebenen Sprachelemente zurückgegriffen werden. In solch einem System gibt es keine Möglichkeit, die Spezifikationssprache durch den Benutzer zu verändern oder für andere Anwendungsgebiete zu erweitern.

Abbildung 2-4 stellt den Zusammenhang zwischen unterschiedlichen Anwendungsgebieten und den entsprechenden Dokumentensystemen dar. Die anwendungsspezifische Funktionalität ist in den Dokumentensystemen enthalten und nicht in den Dokumenten, wie es bei den generischen Ansätzen der Fall ist. Aus diesem Grund sind die Dokumente kleiner abgebildet als in Abbildung 2-1.

#### *2.3.2.1 Beispiele*

Multimedia-Systeme aus der Forschung [LiG90, HRB93, JLR98, Wir99a, Hos01, SDL+96, BuZ92, SKD96] bieten fest vorgegebene Spezifikationssprachen. Diese legen die Anwendungsgebiete fest, für die die Systeme geeignet sind. Es ist im Einzelfall zu bewerten, wie gut die Unterstützung für das jeweilige Anwendungsgebiet ist.

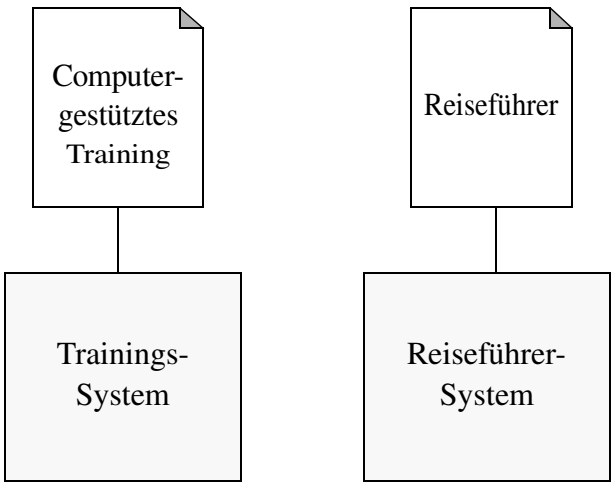


**Abb. 2-4 :** Architektur anwendungsspezifischer Ansätze

In diese Klasse von Ansätzen fallen als kommerzielle Beispiele unter anderen Macromedia Authorware Professional [Aut01] und Microsoft PowerPoint [Bes01]. Macromedia Authorware Professional ist speziell für computergestützte Lerneinheiten gedacht. Mit Microsoft PowerPoint [Bes01] können Vorträge durch multimedial aufbereitete Folien unterstützt werden.

Der vom World Wide Web Consortium (W3C) [W3C] propagierte Standard SMIL (Synchronized Multimedia Integration Language) [Hos01] ist für das Anwendungsgebiet „interaktive fernsehähnliche Präsentationen“ gedacht. SMIL-Dokumente werden in einer auf XML-basierenden Sprache erstellt. Sie stellt zur Spezifikation des temporalen Verlaufs einer Multimedia-Präsentation zwei unterschiedliche Beziehungen zur Verfügung: eine sequentielle Beziehung (`<seq>`) und eine Parallelitätsbeziehung (`<par>`). Anhand der Schachtelung der zeitlichen Beziehungen in der Dokumentspezifikation wird der zeitliche Ablauf der Präsentation für den Autor sichtbar.

Die Verwendung von XML im Zusammenhang mit Textdokumenten bedeutet die Auszeichnung des strukturellen Aufbaus, beispielsweise von Überschriften und Absätzen. Dies führt zu einer baumartigen Struktur. Vom W3C wird vorgeschlagen, bei der Verarbeitung von XML-Dokumenten das DOM (Document Object Model) [App+98] zu verwenden. Das DOM stellt eine objektorientierte Darstellung eines XML-Dokuments als Baum zur Verfügung. Es werden Attribute und Methoden definiert, die die Verarbeitung des DOMs ermöglichen.

Textdokumente besitzen im Allgemeinen eine Baumstruktur, weshalb XML und das DOM sich sehr gut zur ihrer Verarbeitung eignen. Aus Sicht eines möglichst allgemeinen Dokumentenmodells für Multimedia-Dokumente ist die Einschränkung, dass es sich bei dem DOM um einen Baum handelt, ein Nachteil. Die Verwendung des DOM als Dokumentenmodell bedeutet, dass allen anwendungsspezifischen Spezifikationssprachen eine Baumstruktur, wie im Beispiel von SMIL, zu Grunde liegen muss.

Die Struktur multimedialer Dokumente kann nicht immer auf einen Baum abgebildet werden und verursacht demnach zusätzlichen Aufwand bei der Dokumentenverarbeitung. Beispielsweise wird bei der Spezifikation eines Dokuments mittels Intervalloperatoren [All83, WaR94] ein temporaler Beziehungsgraph [Wir99a, Wir99b] erstellt. Es wird also ein Graph als Dokumentenmodell benötigt.

Im Channel-Konzept, auf dem der CMIF-Ansatz (CWI Multimedia Interchange Format) [HRB93, RJM+93] basiert, wird ein Dokumentenmodell speziell für die Beschreibung von temporalen Beziehungen definiert. Das Dokumentenmodell des Channel-Ansatzes besteht aus so genannten Kanälen. Innerhalb von Kanälen wird ein sequentieller Zeitverlauf angenommen. Zusätzlich können mehrere Kanäle gleichzeitig präsentiert werden. Damit lässt sich modellieren, dass Medienelemente hintereinander oder gleichzeitig präsentiert werden. Interaktionen erlauben das Verändern des aktuellen Ausspielzeitpunkts. Dieses Modell ist speziell für die Beschreibung temporaler Beziehungen ausgelegt, womit es nicht die Anforderung der Unterstützung spezieller Konzepte aus unterschiedlichen Anwendungsgebieten erfüllt.

Temporale Petri-Netze [LiG90] bestehen aus Stellen und Transitionen. Die Verweildauer einer Marke auf einer Stelle eines temporalen Petri-Netzes entspricht der Präsentationsdauer eines der Stelle zugeordneten Medienelements. Das Schalten einer Transition entspricht dem Beenden und Starten der Präsentation von zugeordneten Medienelementen. Interaktions-Konzepte oder räumliche Anordnungen von Medienelementen können mit den Petri-Netzten nur sehr umständlich [KFB99] beziehungsweise gar nicht dargestellt werden. Auch dieses Modell ist speziell für die Beschreibung temporaler Beziehungen ausgelegt.

*2.3.2.2 Bewertung*

Anwendungsspezifische Ansätze bieten eine fest vorgegebene Spezifikationssprache und spezielle Dokumentenmodelle. Damit haben diese den Vorteil, dass das Autorenwerkzeug, das Präsentationssystem und das Dokumentenformat an das jeweilige Anwendungsgebiet optimal angepasst wurden. Somit werden die Forderungen an ein hohes Abstraktionsniveau und nach einem abgeschlossenen Spezifikationsmodell erfüllt.

Ist die Spezifikationssprache oder das Dokumentenmodell für ein Anwendungsgebiet nicht ausreichend, so kann ein Dokument in dem entsprechenden Anwendungsgebiet nicht spezifiziert werden. So können beispielsweise SMIL und alle anderen in diesem Abschnitt vorgestellten Ansätze nicht dazu verwendet werden, die beiden Beispiele aus dem Einleitungs-Kapitel zu verwirklichen. Damit wird die Forderung nach Erweiterbarkeit für neue Anwendungsgebiete nicht erfüllt.

Anwendungsspezifische Ansätze haben aus Sicht eines Autors den Nachteil, dass für jedes Anwendungsgebiet ein anderes System verwendet werden muss. Jedes Dokumentensystem bietet ein eigenes Autorenwerkzeug, das eine neue Benutzungsschnittstelle und ein anderes Spezifikationsmodell verwendet. Dies bedeutet, dass sich der Autor in das neue System einarbeiten muss.

Existiert ein Multimedia-Dokumentensystem für ein Anwendungsgebiet noch nicht, muss es zuerst entwickelt werden, was einen möglicherweise hohen Aufwand mit sich bringt. Bei der Entwicklung kann ein Dokumentenmodell gewählt werden, das sich für das jeweilige Anwendungsgebiet besonders eignet. Beispielsweise kann für das Anwendungsgebiet computergestütztes Training ein Flussdiagramm verwendet werden, das den Verlauf durch eine Trainingseinheit gut darstellt. Für den Reisführer wäre das Modell eher schlecht geeignet, da in ihm normalerweise keine vorab festgelegten Routen gefragt sind. In diesem Fall würde die Wahl auf ein anderes geeignetes Modell fallen. Damit lassen sich nur sehr schwer Teile von anderen Autorenwerkzeugen bei der Entwicklung eines neuen Autorenwerkzeugs wiederverwenden.

Viele unterschiedliche proprietäre Dokumentenspeicherformate führen zu Problemen bei ihrer Weiterverarbeitung. Beispielsweise ermöglicht ein proprietäres Speicherformat keine Integration und Auswertung von Meta-Daten durch Nachweissysteme für digitale Bibliotheken. Dazu

muss das Nachweissystem alle Dateiformate kennen und die Formate müssen offengelegt sein. Ein offenes und erweiterbares Format würde an dieser Stelle keine Probleme bereiten.

Die Forderungen nach der ad-hoc Erweiterbarkeit und der Wiederverwendbarkeit von Erweiterungen werden nicht erfüllt, da bei diesen Ansätzen keine Erweiterungen für neue Anwendungsgebiete möglich sind.

## 2.4 Zusammenfassende Bewertung existierender Ansätze

Multimedia-Präsentationen werden immer für bestimmte Anwendungsgebiete erstellt. Betrachtet man die Unterstützung, die ein Autor erhält, wenn er mehr als “nur” den zeitlichen und räumlichen Aufbau eines Multimedia-Dokuments spezifizieren muss, so ist zu erkennen, dass existierende Ansätze in zwei Klassen eingeteilt werden können: die generischen und die anwendungsspezifischen Ansätze. Wie in den vorangegangenen Unterkapiteln beschrieben, erfüllt keiner der beiden Ansätze alle aufgestellten Forderungen.

Ein Multimedia-System, das alle vier aufgestellten Anforderungen erfüllt, soll im folgenden ein erweiterbares Multimedia-System genannt werden. Durch dieses wird der zu Anfang des Kapitels vorgenommenen Klassifikation von Multimediasystemen eine weitere Klasse hinzugefügt: Die erweiterbaren Ansätze.

Abbildung 2-5 stellt die Beziehung zwischen Dokumenten und dem erweiterbaren Ansatz graphisch dar.

Verglichen mit den generischen Ansätzen bietet der erweiterbare Ansatz folgende Vorteile. Für die Spezifikation wird ein Dokumentenmodell verwendet, das Abstraktionen und Beziehungen auf einem hohen Abstraktionsniveau bietet. Dieses ist zur Unterstützung neuer Anwendungsgebiete erweiterbar. Für die Erschließung eines neuen Anwendungsbereichs muss lediglich die entsprechende Erweiterung entwickelt werden. Für die Erstellung der Erweiterung wird ein Experte benötigt, da die Erweiterung programmiert werden muss. Die klare Trennung zwischen Spezifikation eines Dokumentes und Entwicklung einer Erweiterung ermöglicht die Wiederverwendung von Erweiterungen. Für die (Wieder-)Verwendung der Erweiterung werden keine spe-

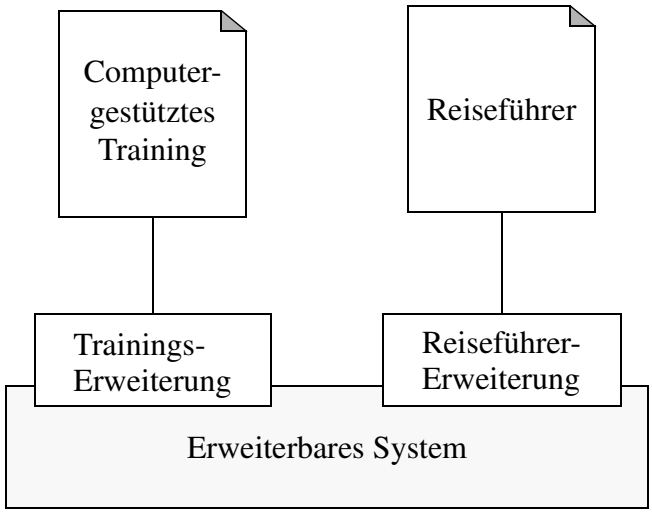


**Abb. 2-5 :** Architektur erweiterbarer Ansätze

ziellen Kenntnisse benötigt. Dies geschieht automatisch durch die ad-hoc Erweiterbarkeit des Ansatzes.

Da anwendungsspezifische Ansätze nicht erweiterbar sind, muss für ein anderes Anwendungsgebiet auch ein anderes Dokumentensystem verwendet werden. Dabei ist nicht auszuschließen, dass dieses andere Dokumentensystem ein neues Autorenwerkzeug mit neuer Benutzungsschnittstelle, ein neues Dokumentenformat und ein neues Präsentationswerkzeug besitzt. Dies bedeutet einen erhöhten Einarbeitungsaufwand für den Autor. Dieser lässt sich durch einen erweiterbaren Ansatz reduzieren. Eigenschaften der Vorgehensweise und der Benutzungsschnittstelle, die unabhängig vom Anwendungsgebiet sind, können zusammengefasst werden und müssen nur einmal durch den Autor erlernt werden.

Existiert für ein Anwendungsgebiet kein Dokumentensystem, dann muss es zuerst entwickelt werden, was einen zusätzlichen Aufwand bedeutet. Dieser wird bei den erweiterbaren Ansätzen durch die Erweiterbarkeit vermieden. Sie ermöglicht es, nachträglich anwendungsspezifische Funktionalität zu integrieren, wobei nur die Erweiterung realisiert werden muss und nicht ein neues Dokumentensystem.

In Tabelle 2-1 werden die drei Ansätze bezüglich der Erfüllung der Anforderungen zusammengefasst. Ein „+“ bedeutet, dass die Anforderung erfüllt wird. Ein „-“ heißt, dass die Anforderung

nicht erfüllt wird. Die „0“ im Fall der Erweiterbarkeit generischer Systeme bedeutet, dass mittels Skriptprogrammierung anwendungsspezifische Funktionalität realisiert werden kann. Dabei findet aber keine Erweiterung eines Spezifikationsmodells auf höherem Abstraktionsniveau statt.

| | Generische Systeme | Anwendungs-spezifische Systeme | Erweiterbare Systeme |
|---|---|---|---|
| Abstraktionsniveau | - | + | + |
| Ad-hoc Erweiterbarkeit | 0 | - | + |
| Abgeschlossenheit | - | + | + |
| Wiederverwendbarkeit | - | - | + |

*Tabelle 2-1 : Vergleich der Ansätze*

Die Eigenschaften des erweiterbaren Ansatzes lassen sich wie folgt zusammenfassen: Er basiert auf einem Grundmodell und alle Erweiterungen können in *ein* System integriert werden. Durch die Erweiterbarkeit ist es ausreichend, *ein* Präsentationssystem, *ein* Autorenwerkzeug und *ein* Transferformat zur verwenden. Durch die Verwendung von nur einem Autorenwerkzeug für Multimedia-Dokumente aus unterschiedlichen Anwendungsgebieten werden die Anforderungen an die Kenntnisse und Fähigkeiten der Autoren reduziert. Durch die ad-hoc Erweiterbarkeit wird die Konfiguration des Autorenwerkzeugs und des Präsentationssystems für den Benutzer automatisch erledigt.

Ein erweiterbarer Ansatz dient der Reduzierung des Aufwands, der bei der Erstellung von Multimedia-Dokumenten vor dem Hintergrund der unterschiedlichen Anwendungsgebiete anfällt. Wesentlicher Teil eines erweiterbaren Ansatzes ist das Dokumentenmodell. Es hat Auswirkungen auf die Konzeption eines erweiterbaren Präsentationssystems und eines erweiterbaren Autorenwerkzeugs.

Zur Erfüllung der gestellten Anforderungen muss ein entsprechendes Dokumentenmodell, die Architektur des erweiterbaren Präsentationssystems und die Architektur des erweiterbaren

Autorenwerkzeugs entworfen werden. Die Konzeption und Realisierung eines erweiterbaren Multimedia-Dokumentensystems, das alle genannten Anforderungen erfüllt, ist Gegenstand dieser Abhandlung.

# 3 EIN ERWEITERBARES META-DOKUMENTENMODELL

Im Folgenden wird ein Multimedia-Dokumentenmodell entworfen, das den im vorherigen Kapitel gestellten Anforderungen genügt. Das Dokumentenmodell kann auf zwei verschiedene Arten spezifiziert werden. Die erste Möglichkeit ist eine textuelle Spezifikation. Bei der zweiten Möglichkeit handelt es sich um die graphische Darstellung der textuellen Spezifikation. Die graphische Darstellung ist äquivalent zur textuellen Darstellung. Sie bietet aber den Vorteil, Spezifikationen übersichtlicher darzustellen. Aus diesem Grund wird für die Beispiele in dieser Abhandlung die graphische Darstellung gewählt.

Zuerst wird das Grundmodell eingeführt, das auf der Idee basiert, alle Beziehung zwischen Medienelementen mittels Operatoren zu beschreiben. Damit das Dokumentenmodell auch in der praktischen Anwendung ausreichende Möglichkeiten bietet, werden anschließend zwei Erweiterungen vorgenommen. Von dem hier beschriebenen Meta-Dokumentenmodell können Instanzen für ein konkretes Anwendungsgebiet gebildet werden. Die Eigenschaften des Meta-Dokumentenmodells werden anhand von Beispielen konkreter Instanzen illustriert.

## 3.1 Das Meta-Dokumentenmodell

Für das Dokumentenmodell des erweiterbaren Ansatzes wurde ein Operatoren-Ansatz gewählt. Durch Operatoren lassen sich Beziehungen und Abhängigkeiten zwischen Medienelementen beschreiben. Die Operatoren können in einem Autorensystem graphisch dargestellt und interaktiv manipuliert werden. Dieser Ansatz liegt allen verwendeten Anwendungskonzepten zu Grunde, was einen geringen Aufwand beim Wechsel des Anwendungsgebiets bedeutet.

Damit müssen alle Konzepte, die verwendet werden sollen, durch Operatoren beschrieben werden können. Diese Einschränkung hat zur Konsequenz, dass ein Konzept, das nicht mittels Operatoren beschrieben werden kann, nicht verwendet werden kann. Beispielsweise lässt sich der Ansatz, Dokumente mittels temporalen Petri-Netzen [LiG90] zu spezifizieren, nicht durch Operatoren darstellen. Dies liegt daran, dass den temporalen Petri-Netzen ein anderes Modell zu

Grunde liegt. Der Nachteil ist in diesem Fall aber nicht schwerwiegend, da es Alternativen gibt, die stattdessen verwendet werden können, beispielsweise temporale Intervall-Operatoren.

### 3.1.1 Medienelemente

Medienelemente repräsentieren die Informationseinheiten, die in einem Multimedia-Dokument zu einer Präsentation zusammengefasst werden. Eine ausführliche Beschreibung der Medienelemente wurde bereits in Kapitel 2.1 gegeben.

Neben den passiven Medienelementen, die lediglich ihre Mediendaten präsentieren, gibt es auch Interaktionselemente. Diese nehmen an der Interaktion mit dem Benutzer teil. Für die Auswertung von Interaktionen können Medienelemente auch Zustandsinformationen speichern, die später zur Bestimmung der Reaktion auf eine Interaktion herangezogen werden. Das Kontrollkästchen ist ein Beispiel für ein Interaktionselemente. Es kann durch Benutzerinteraktion selektiert oder deselektiert werden und speichert seinen aktuellen Zustand. Ist das Kontrollkästchen Teil einer Multiple-Choice-Frage, wird sein Zustand bei der Auswertung abgefragt.

Ein Medienelement umfasst eine Reihe von Parametern, die durch den jeweiligen Typ des Medienelements bestimmt werden. Parameter beschreiben Eigenschaften von Medienelementen, die zur Präsentationszeit von Bedeutung sind. Einer der am häufigsten vorkommenden Parameter ist beispielsweise der Uniform Resource Locator (URL) der Mediendaten. Er beschreibt den Ort, an dem sich die Mediendaten befinden. Je nach Typ kommen weitere Attribute hinzu, beispielsweise Präsentationsdauer oder Breite und Höhe der Präsentationsfläche.

### 3.1.2 Operatoren

Wie schon eingangs erwähnt, wird ein Operator zur Beschreibung einer Beziehung zwischen Medienelementen verwendet.

Bei den Medienelementen, die durch einen Operator miteinander in Beziehung gesetzt werden, wird zwischen Quell- und Zielmedien unterschieden. Das Beispiel eines *inFrontOf*-Operators, der die Überlappung von Medienelementen bei der Darstellung bestimmt, verdeutlicht dies. Der Operator kann verwendet werden, um festzugelegen, dass ein Text vor einem Bild erscheint, denn in der umgekehrten Reihenfolge wäre er für den Betrachter nicht sichtbar. Abbildung 3-1

zeigt die Verwendung des *inFrontOf*-Operators. In Abbildungen werden Medienelemente durch abgerundete Rechtecke und Operatoren durch Rechtecke dargestellt. Ein Quellmedium ist durch einen Pfeil zum Operator und ein Zielmedium durch einen Pfeil zum Medienelement gekennzeichnet. Eine äquivalente textuelle Darstellung wäre “Text *inFrontOf* Bild”.

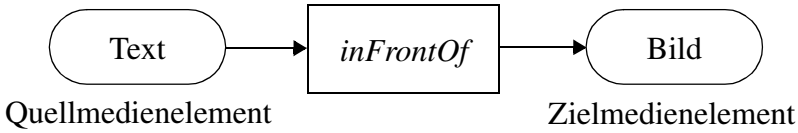


**Abb. 3-1 :** Beispiel für Quell- und Zielmedien

Bezüglich der Kardinalität der Operatoren macht das Meta-Dokumentenmodell keine Vorgaben. Prinzipiell kann ein Operator beliebig viele Quell- und Zielmedien haben. Die Operatoren eines bestimmten Konzepts geben die Kardinalität aber genau vor. In Abbildung 3-2 wird diese Beziehung graphisch veranschaulicht.

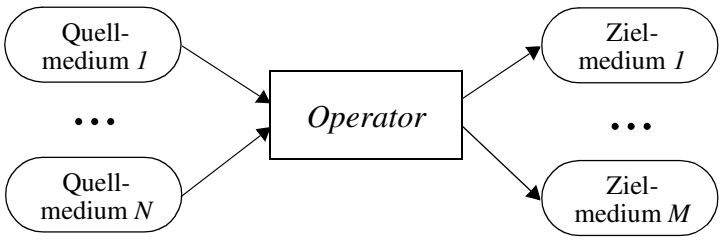


**Abb. 3-2 :** Beziehung zwischen Operator und Medienelementen

Im Dokumentenmodell sind die Operanden der Operatoren die Medienelemente. Wenn ein Operator mehr als zwei Operanden hat, ist eine graphische Darstellung leichter zu verstehen als beispielsweise eine rein textuelle. Dies liegt daran, dass der Beziehungsgraph zwischen Operatoren und Medienelementen auf eine textuelle Darstellung abgebildet werden muss.

Neben den Medienelementen besitzen auch die Operatoren Parameter. Mittels Parametern lassen sich Eigenschaften der Beziehungen, die durch Operatoren beschrieben werden, exakt festlegen. Ein *setPosition*-Operator legt beispielsweise die Position der linken Oberen Ecke eines Medienelements in der Präsentationsfläche des Dokuments fest. Für jedes Medienelement kann

eine andere Position festgelegt werden. Diese Position wird durch den Parameter des *setPosition*-Operators bestimmt. Temporale Operatoren haben häufig Parameter, die Zeitintervalle beschreiben, wie im nächsten Beispiel erläutert wird.

In Abbildung 3-3 ist ein einfaches Multimedia-Dokument dargestellt. Das Dokument besteht aus zwei Medienelementen und zwei Operatoren. Das erste Medienelement ist ein Bild, das zweite eine Audiosequenz. Der erste Operator bestimmt die Position des Bildes innerhalb der Präsentationsfläche des Dokumentes. Die Parameter des Operators geben dabei die Position der linken oberen Ecke des Bildes an. Bei dem zweiten Operator handelt es sich um einen *while*-Operator. Die Absicht des Autors ist, dass während der Präsentation des Bildes eine Erklärung zu hören ist. Diese Semantik wird mit dem *while*-Operator beschrieben, bei dem es sich um einen so genannten Intervall-Operator handelt; detailliertere Informationen über Intervall-Operatoren sind in [All83, WaR93] zu finden. Die zwei Parameter des *while*-Operators beschreiben die Verzögerung zwischen dem jeweiligen Beginn der Präsentationsintervalle der Medienelemente und dem Ende der Präsentationsintervalle. Der erste Parameter des Operators besagt, dass drei Sekunden nach dem Beginn der Präsentation des Bildes das Abspielen der Audiosequenz beginnen soll. Dementsprechend bedeutet die “5” für die Präsentation, dass fünf Sekunden nach dem Ende der Audiosequenz auch die Präsentation des Bildes beendet werden soll.

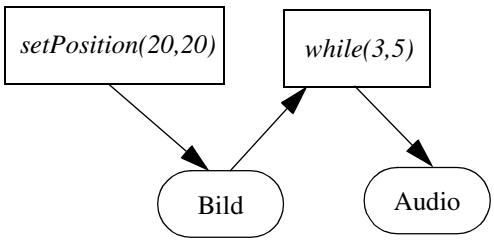


**Abb. 3-3 :** Eine einfache Spezifikation

## 3.2 Erweiterungen des Meta-Dokumentenmodells

Das Grundmodell kann bereits für die Beschreibung einfacher Multimedia-Dokumente herangezogen werden. Zur Beschreibung umfangreicher Multimedia-Dokumente werden noch zwei Erweiterungen des Modells benötigt.

Die erste Erweiterung sind die so genannten Container. Um von einem Teil der Spezifikation zu abstrahieren, kann ein Container verwendet werden. Neben der Aufgabe, einen Teil der Spezifikation zusammenzufassen und von ihr zu abstrahieren, hat ein Container zusätzlich eine anwendungsspezifische Bedeutung. Beispielsweise abstrahiert ein *Frage*-Container in einem computerbasierten Training von der Spezifikation einer Frage.

Die zweite Erweiterung sind die so genannten Effektoperatoren. Der Name bezieht sich auf die Tatsache, dass zu einem bestimmten Zeitpunkt in der Präsentation etwas mit einem Medienelement geschehen soll. Beispielsweise soll eine Szene ausgeblendet oder eine vorher unterbrochene Animation fortgesetzt werden.

### 3.2.1 Container

Container werden zur Strukturierung großer Dokumente verwendet. Logisch zusammenhängende Teile der Spezifikation können damit zusammengefasst werden. Um rekursive Strukturen bilden zu können, wird ein Container wiederum als Medienelement aufgefasst. Der Wurzelcontainer einer Hierarchie repräsentiert das Multimedia-Dokument.

Eine weitere Eigenschaft von Containern ist die anwendungsspezifische Abstraktion vom Containerinhalt. Ein Beispiel hierfür ist der *Frage*-Container eines computergestützten Trainingskonzepts.

Die durch die Container eingeführte Abstraktion von Teilspezifikationen vereinfacht die Navigation in Dokumenten beim Editieren und hilft, die Übersicht über große Dokumente zu behalten. Das folgenden Szenario beschreibt die Anwendungsmöglichkeit von Containern.

Abbildung 3-4 zeigt eine Spezifikation, die durch die Verwendung von Containern strukturiert wurde. Container werden in Abbildungen immer als Rechtecke mit einem Schatten dargestellt. Die Spezifikation besteht aus drei Containern und zwei Operatoren. Der linke obere Container (*Frage*) beinhaltet die Spezifikation einer Frage. Die beiden anderen Container beinhalten jeweils einen Präsentationsabschnitt. Sie werden während der Präsentation in Abhängigkeit davon, ob die Frage richtig oder falsch beantwortet wurde, ausgewählt. Mit welchem Container die Präsentation fortgesetzt wird, wird durch die beiden Operatoren (*correct* und *false*) bestimmt.

Das Beispiel macht deutlich, dass auf dieser Ebene der Spezifikation, nämlich der Reaktion auf die Beantwortung der Frage, der jeweilige Aufbau der Frage oder der anderen beiden Präsentationsabschnitte nicht relevant ist und vor dem Autor verborgen werden kann.

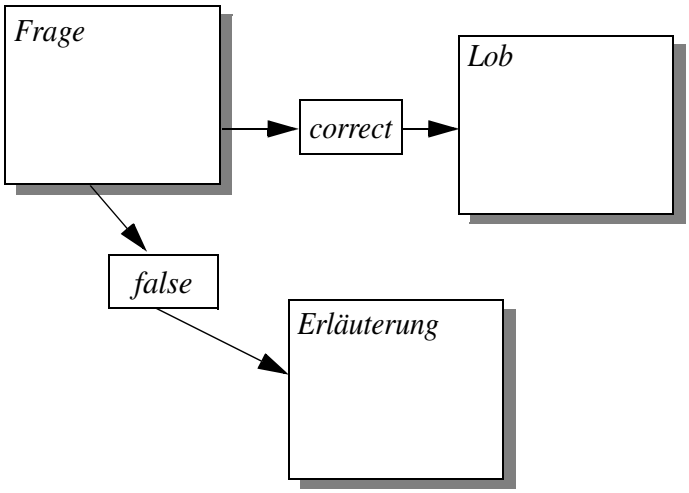


**Abb. 3-4 :** Strukturierung einer Spezifikation durch Container

### 3.2.2 Effektoperatoren

Durch eine Dokumentenspezifikation wird bestimmt, zu welchem Zeitpunkt Medienelemente abgespielt werden. Für komplexe Medienelemente sind nicht nur die Start- und Endzeitpunkte ihrer Präsentation von Bedeutung, sondern auch Zwischenzustände während ihrer Darstellung. Ein Beispiel für solch ein komplexes Medienelement ist eine Animation. Eine Animation besteht aus mehreren Schritten, wobei jeder Schritt als Zustandsübergang aufgefasst werden kann.

In einem Lerndokument über Rechnernetze kann das "Stop and Wait"-Protokoll durch eine Animation veranschaulicht werden. Das Protokoll-Medienelement und die Ausgangssituation der Animation sollen sofort dargestellt werden. Die weiteren Animationsschritte (bspw. ein Sender schickt ein Datenpaket an einen Empfänger) werden erst im Verlauf der restlichen Präsentation dargestellt. Es wird also zwischen der Darstellungszeit eines Medienelements und den Zuständen des Medienelements während der Darstellung unterschieden.

Als Effektmedienelemente werden Medienelemente bezeichnet, die die Eigenschaft haben, mehrere Zustände zu besitzen, auf die während der Präsentation Bezug genommen werden kann.

Um eine Beziehung zwischen den einzelnen Zuständen eines Effektmedienelements und der Präsentation herzustellen, werden die so genannten Effektoperatoren eingeführt. Durch einen Effektoperator kann der Zeitpunkt bestimmt werden, zu dem ein Zustandsübergang des Effektmedienelements stattfinden soll. Hierzu wird der Effektoperator mit temporalen Operatoren verbunden. Anschließend wird der Effektoperator mit einem Effektmedienelement verbunden. Dadurch unterscheiden sich die Effektoperatoren von den bereits eingeführten Operatoren. Mit einem Effektoperator können die Zeitpunkte für die Animationsschritte des "Stop and Wait"-Protokolls bestimmt und somit synchron mit ihrer Erklärung präsentiert werden.

Abbildung 3-5 zeigt die grafische Darstellung eines Effektoperators. In Abbildungen wird er als Raute dargestellt. Im Inneren der Raute steht der Typ des Effektoperators. Ein Effektoperator kann mit mehreren Effektmedien verbunden werden. Diese Verbindung wird in Abbildungen durch einen grauen Pfeil zu diesen Medien dargestellt.

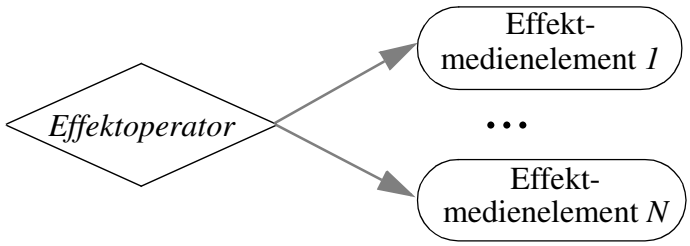


**Abb. 3-5 :** Graphische Darstellung eines Effektoperators

In Abbildung 3-6 ist ein Beispiel für die Verwendung eines Effektoperators gegeben. Anhand dieses Beispiels soll die Verwendung von Effektoperatoren erläutert werden. Das Beispiel umfasst drei Medienelemente. Das erste ist eine Visualisierung, die das “Stop and Wait”-Protokoll graphisch aufbereitet und animiert darstellt. Die beiden Audio-Medienelemente repräsentieren jeweils die gesprochenen Erklärungen eines Protokollschrittes. Die einzelnen Schritte der Animation des Protokolls stellen einen Zustandsübergang des Effektmedienelements dar. Dieser Zustandsübergang kann durch einen speziellen Effektoperator, den *nextStep*-Operator, ver-

anlasst werden. Um dies zu spezifizieren, wird der *nextStep*-Effektoperator mit der Protokollvisualisierung verbunden. Als nächstes muss festgelegt werden, wann der Zustandsübergang, stattfinden soll. Dazu werden ein *nextStep*-Operator und das zugehörige Audio-Medienelement mittels *cobegin*-Operator verbunden. Der *cobegin*-Operator ist ein temporaler Intervalloperator [WWR95, Wir99a], der besagt, dass zwei Medienelemente zum gleichen Zeitpunkt mit ihrer Darstellung beginnen. Effektoperatoren übernehmen aus Sicht des temporalen Operators die Funktion eines Medienelements, somit wird der Zustandsübergang zeitgleich mit dem Beginn der Präsentation der Audiosequenz angestoßen.

Durch die alleinige Verwendung des *cobegin*-Operators kann der Zeitpunkt der Darstellung der Protokollvisualisierung bestimmt werden, nicht aber die Synchronisation auf einzelne Animationsschritte. Aus diesem Grund werden das Audiomedienelement „audio1“ und die Protokollvisualisierung nicht direkt mittels dem *cobegin*-Operator verbunden sondern zusätzlich noch der *nextStep*-Operator verwendet. In Abbildung 3-6 wird dies noch mal durch einen zweiten *nextStep*-Effektoperator verdeutlich, der sich auf den zweiten Schritt der Animation und nicht auf den Anfangszeitpunkt der Darstellung der Protokollvisualisierung bezieht.

Um das Beispiel zu vervollständigen, muss die zeitliche Beziehung zwischen den beiden Audiomedienelementen spezifiziert werden. Dies kann durch einen *before*-Operator [Wir99a] erledigt werden. Der *before*-Operator bewirkt, dass Audio1 vor Audio2 abgespielt wird.

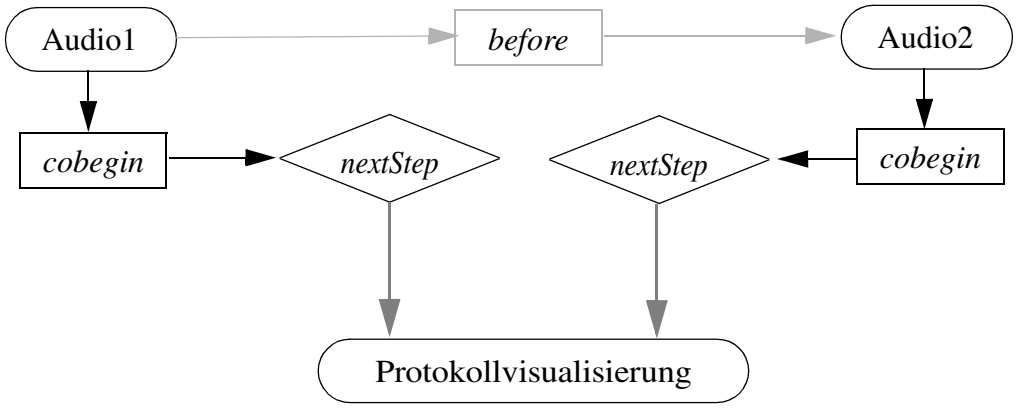


**Abb. 3-6 :** Steuerung einer Animation durch einen *nextStep*-Effektoperator

## 3.3 Anwendungsspezifische Instanzen des Meta-Dokumentenmodells

Das vorgestellte, auf einem Operatoren-Ansatz basierende Dokumentenmodell ist ein Meta-Dokumentenmodell. Es beschreibt den schematischen Aufbau von Multimedia-Dokumenten und die Eigenschaften der Dokumentelemente. Es beschreibt keine speziellen Medienelemente oder Operatoren. Dies ist Aufgabe von anwendungsspezifischen Instanzen des Meta-Dokumentenmodells. Durch anwendungsspezifische Instanzen werden konkrete Medienelemente und Operatoren festgelegt. Zur Illustration der Eigenschaften des Meta-Dokumentenmodells wurden für die Beispiele in diesem Kapitels bereits Instanzen des Meta-Dokumentenmodells gebildet.

Bisherige Ansätze, die auch auf einem Operatoren-Ansatz basieren [Wir99a, JLR98], verwenden eine fest vorgegebene Anzahl von Operatoren eines bestimmten Konzepts. Im Gegensatz dazu erlaubt der erweiterbare Ansatz, basierend auf dem Meta-Dokumentenmodell, das Hinzufügen neuer Instanzen des Meta-Dokumentenmodells für spezielle Anwendungsgebiete, die neue Operatoren, neue Medienelemente und neue Container umfassen.

Um beispielsweise einen Reiseführer zu realisieren, muss eine Erweiterung als Instanz des Meta-Dokumentenmodells definiert werden. Dazu muss festgelegt werden, durch welche Medienelemente und Operatoren das Reiseführerkonzept auf das Meta-Dokumentenmodell abgebildet werden kann. Diese Dokumentelemente werden anschließend von einem Autor dazu verwendet, einen Reiseführer zu erstellen.

An dieser Stelle kann nur eine eingeschränkte Spezifikationssprache für multimediale Reiseführer beschrieben werden. Eine komplette Spezifikationssprache, die alle möglichen Varianten und Bedürfnisse von potentiellen Autoren abdeckt, wäre vom Umfang zu groß, um sie in dieser Abhandlung zu verwenden. Die Ideen und Konzepte des erweiterbaren Ansatzes lassen sich aber bereits an Spezifikationssprachen mit eingeschränktem Funktionsumfang demonstrieren. Dies gilt für alle Beispiele dieser Abhandlung.

Das Konzept hinter einem Reiseführer soll beispielsweise wie folgt sein: Ein Reiseführer deckt ein bestimmtes geographisches Gebiet ab, in dem sich eine Reihe von Sehenswürdigkeiten befinden. Zu jeder Sehenswürdigkeit gibt es eine multimedial aufbereitete Präsentation. Kommt

ein Benutzter bei der Erkundung des Gebiets nahe genug an eine Sehenswürdigkeit, wird die Präsentation dieser Sehenswürdigkeit gestartet.

Die Spezifikationssprache für multimediale Reiseführer stellt zwei Container bereit. Der erste Container ist der so genannte *LocationArea*-Container. Er legt die Größe des abgedeckten Gebiets fest und enthält die *Location*-Container. Sie enthalten wiederum die für eine bestimmte Sehenswürdigkeit entsprechende Multimedia-Präsentation. Durch den *setLocation*-Operator werden die Koordinaten der Sehenswürdigkeit festgelegt. Der *activationRadius*-Operator gibt einen Radius (beispielsweise 10m) vor, innerhalb dessen sich der Benutzer befinden muss, damit die Präsentation über die Sehenswürdigkeit gestartet wird. Abbildung 3-7 zeigt eine Beispiel-Spezifikation, die die beschriebenen Dokumentelemente verwendet. In Anhang A ist ein umfangreicheres Beispiel für einen Reiseführer-Dokument gegeben.

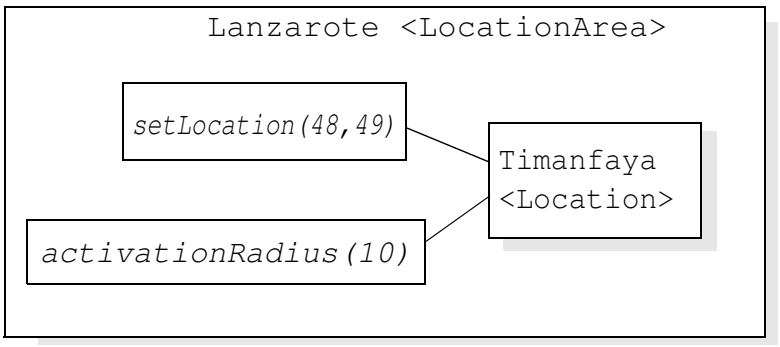


**Abb. 3-7 :** Beispiel für eine Reiseführer-Erweiterung

## 3.4 Klassifikation anwendungsspezifischer Instanzen

Die Vielzahl der möglichen Erweiterungen für unterschiedliche Anwendungsgebiete machen deren Klassifikation notwendig. Eine Klassifikation bietet dem Autor Hilfestellung bei der Erstellung eines Dokumentes, d.h. er kann zum Beispiel sehen, welche Medienelemente oder Konzepte es in einem Anwendungsgebiet bereits gibt oder welche Erweiterungen noch benötigt werden.

Zusätzlich ermöglicht die Klassifikation der Anwendungsgebiete eine spätere automatische Einordnung von Dokumenten in bestimmte Anwendungsgebiete, je nach dem, welche Erweite-

rungen sie verwendet. Verwendet ein Dokument beispielsweise Konzepte aus dem Bereich computergestütztes Training und Medienelemente für Molekül-Visualisierungen aus dem Anwendungsbereich Chemie, so kann mit hoher Wahrscheinlichkeit gefolgert werden, dass es sich bei diesem Dokument um ein Lehrdokument für Chemiker handelt.

Medienelemente und Operatoren können entsprechend ihrem Anwendungsgebiet klassifiziert werden. Das Anwendungsgebiet computergestütztes Training, bietet beispielsweise Operatoren zur Spezifikation von Dokumenten und spezielle Medienelemente zur Spezifikation von Fragen. Zur Unterstützung eines Anwendungsgebiets müssen nicht immer Medienelemente und Operatoren vorhanden sein. Es sind Anwendungsgebiete denkbar, die nur Medienelemente zur Verfügung stellen. Ein Beispiel hierfür sind 3D-Visualisierungen in der Chemie.

Abbildung 3-8 zeigt einen Ausschnitt aus der Klassifikation von Erweiterungen. Die einzelnen Anwendungsgebiete bilden einen so genannten Sektor. Beispielhaft sind die Sektoren computergestütztes Lernen, 3D-Visualisierungen und Reiseführer in der Abbildung aufgeführt.

Monomedienelemente, beispielsweise ein Video-Medienelement, werden auch als Basiselemente bezeichnet. Basiselemente sind keinem Anwendungsgebiet zugeordnet. Sind die Dokumentelemente einem Anwendungsgebiet zugeordnet, werden sie Sektorelemente genannt (beispielsweise das Konzept für computergestütztes Training) und wenn sie für eine bestimmte Anwendung innerhalb des Sektors gedacht sind, werden sie Anwendungselemente genannt (beispielsweise die Animation von Rechner-Protokollen aus dem Anwendungsbereich computergestütztes Tutorial).

Ein Dokument kann Elemente aus unterschiedlichen Sektoren gleichzeitig verwenden, zum Beispiel ein Lerndokument für Chemiker oder ein Reiseführer, der ein Quiz enthält, in dem abgefragt wird, was der Benutzer auf der Erkundungstour gelernt hat.

Die Erweiterbarkeit des Ansatzes erlaubt es, unabhängig voneinander entwickelte Erweiterungen für das gleiche Anwendungsgebiet zu integrieren. Da diesen Erweiterungen unterschiedliche Konzepte und Modelle zu Grunde liegen, macht es keinen Sinn, diese unterschiedlichen Dokumentelemente in einem Dokument miteinander zu kombinieren. Beispiele hierfür sind die gleichzeitige Verwendung von Operatoren unterschiedlicher Konzepte für die temporale Spezifikation (beispielsweise Intervalloperatoren und eine Zeitachse, dargestellt durch *start*- und

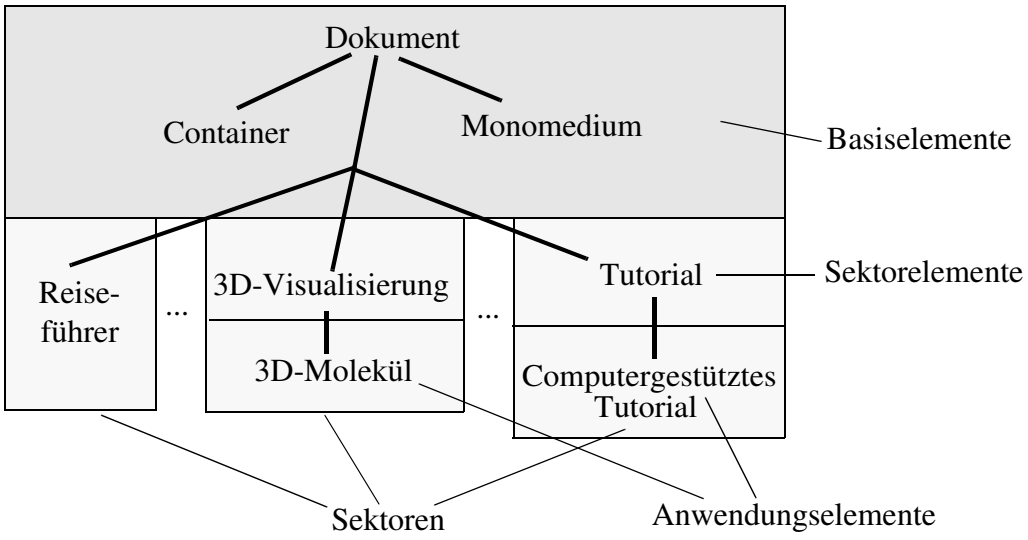


**Abb. 3-8 :** Ausschnitt aus der Sektorenhierarchie

*stop*-Operatoren) oder Operatoren von alternativen Modellen zur Spezifikation von computergestützten Lerneinheiten.

Die Wiederverwendung von Medienelementen und Operatoren ermöglicht Autoren die Erstellung von Multimedia-Dokumenten in den entsprechenden Anwendungssektoren. Das Zusammenfassen von Erweiterungen und ihre Klassifikation stellt ein Konzept für die Wiederverwendung anwendungsspezifische Funktionalität dar.

Nach der Entwicklung neuer Medienelemente oder Operatoren kann sich der Autor entschließen, sie für die Wiederverwendung durch andere Autoren zur Verfügung zu stellen. Um Medienelemente und Operatoren zur Verfügung stellen zu können, müssen sie zuerst klassifiziert und damit in die Sektorenhierarchie eingefügt werden. Die Sektorenhierarchie wird durch die Einordnung aufgebaut und im Laufe der Verwendung des erweiterbaren Ansatzes durch das Hinzufügen neuer Medienelemente und Operatoren immer mehr Unterstützung für den Autor bieten. Dies bedeutet gleichzeitig, dass es immer seltener notwendig wird, eine Erweiterung neu zu entwickeln.

## 3.5 Einordnung und Zusammenfassung

In diesem Kapitel wurde das Meta-Dokumentenmodell eingeführt, auf dem Instanzen gebildet werden können, um eine Spezifikationssprache für ein konkretes Anwendungsgebiet zu erhalten. Die Idee ist, dass eine Multimedia-Präsentation aus Medienelementen und Beziehungen zwischen diesen Medienelementen besteht und dies im Meta-Dokumentenmodell sichtbar ist. Beziehungen werden durch Operatoren dargestellt, und um Dokumente besser strukturieren zu können, werden Container zur Verfügung gestellt.

Als ein erweiterbares Multimedia-Präsentationssystem wird im folgenden ein multimediales Präsentationssystem verstanden, das Dokumente darstellen kann, die auf unterschiedlichen, anwendungsspezifischen Spezifikationssprachen basieren. Ein erweiterbares Autorenwerkzeug bietet die Möglichkeit, die Autorenschnittstelle so zu erweitern, dass der Autor bei der Erstellung von Multimedia-Dokumenten in unterschiedlichen Anwendungsgebieten durch spezifische Spezifikationssprachen unterstützt wird. Dabei soll für den Erstellungsprozess selbst nur *ein* Paradigma durch das System unterstützt werden. Dies führt dazu, dass sich die Benutzungsschnittstelle und die Art und Weise, wie Dokumente erstellt werden, bei einem Wechsel des Anwendungsgebiets nicht ändern, was den Einarbeitungsaufwand für Autoren reduziert.

Um die Forderung einer sich nicht ändernden Benutzerführung zu genügen, werden die Eigenschaften der unterschiedlichen Spezifikationssprachen durch das Meta-Dokumentenmodell zusammengefasst. Basierend auf diesem Meta-Dokumentenmodell kann anschließend die Schnittstelle des Autorenwerkzeugs konzipiert werden. Um die Benutzungsschnittstelle einfach zu halten, muss das Meta-Dokumentenmodell eine geringe Komplexität besitzen, was durch nur vier unterschiedliche Typen von Dokumentenelemente erreicht wird.

Um die Nachteile der existierenden Ansätze zu vermeiden, wird in dieser Abhandlung ein auf der Ebene der Benutzungsschnittstelle und der zur Verfügung gestellten Spezifikationssprachen erweiterbares Autorenwerkzeug eingeführt. Dem Autorenwerkzeug kann eine Spezifikationssprache für ein neues Anwendungsgebiet hinzugefügt werden. Zur Darstellung der Dokumente muss dem Präsentationssystem ebenfalls die Implementierung der Spezifikationssprachen hinzugefügt werden.

Das in diesem Kapitel spezifizierte Meta-Dokumentenmodell hat weitreichende Auswirkungen auf den weiteren Verlauf der Konzeption und Entwicklung eines erweiterbaren Präsentations- und Autorensystems. Die Konzeption des erweiterbaren Präsentationssystem muss die Erweiterbarkeit der Spezifikationssprache unterstützen. Das erweiterbare Autorenwerkzeug stellt dem Benutzer eine Schnittstelle zur Spezifikation von Dokumenten bereit, die auf dem Meta-Dokumentenmodell basiert.

Im folgenden Kapitel wird vorgestellt, wie das Meta-Dokumentenmodell die Architektur eines erweiterbaren Präsentationssystems beeinflusst. Zusätzlich wird die Implementierung anwendungsspezifischer Instanzen diskutiert.

# 4 ARCHITEKTUR ERWEITERBARER MULTIMEDIA-PRÄSENTATIONSSYSTEME

Nachdem im vorangegangenen Kapitel das Meta-Dokumentenmodell und anwendungsspezifische Instanzen eingeführt wurden, werden in diesem Kapitel die Architektur eines erweiterbaren Präsentationssystems und die Implementierung einer anwendungsspezifischen Instanz betrachtet.

Grundlage dafür ist das so genannte Objektmodell, das einer Abbildung des Meta-Dokumentenmodells auf Objekte einer Programmiersprache entspricht. Die Operatoren eines Anwendungsgebiets werden durch einen Manager implementiert. Dabei kommunizieren Manager und Medienelemente mittels Ereignissen. Die Aufteilung in Benachrichtigungsereignisse, Steuerereignisse und Manager als zentrale Komponenten zur Implementierung der Operatoren spiegelt sich in der Architektur eines erweiterbaren Präsentationssystems als Schnittstellenbeschreibung wider.

Abschließend wird die Vorgehensweise bei der Entwicklung einer Erweiterung anhand des Beispiels für ein computergestütztes Training verdeutlicht.

## 4.1 Beziehung zwischen Managern und Medienelementen

Aus der Architektur des erweiterbaren Präsentationssystems werden hier zwei Bereiche im Detail betrachtet, in denen die Erweiterbarkeit eine besondere Rolle spielt. Es handelt sich dabei um das Dokumentenmodell und die Manager/Operatoren-Beziehung.

In den abgebildeten Diagrammen wird als Notation UML (*U*niversal *M*odeling *L*anguage) verwendet. In [GRJ99] wird eine ausführliche Beschreibung der Notation gegeben, eine kurze Einführung findet man in [Qua98].

Der erste, durch die Erweiterbarkeit betroffene Bereich, ist das Meta-Dokumentenmodell. In diesem Kapitel wird ein vereinfachtes Meta-Dokumentenmodell ohne Effektoperatoren betrachtet, da durch sie keine neuen Kenntnisse gewonnen werden.

Die Eigenschaften der Dokumentelemente des Meta-Dokumentenmodells gelten für alle Erweiterungen und werden deshalb durch Schnittstellen beschrieben (Abbildung 4-1). Die Eigenschaft der Container, Medienelemente und Operatoren enthalten zu können, wird durch eine Aggregation beschrieben. Die Beziehung zwischen Operatoren und Medienelementen wird als Assoziation dargestellt. Diese Eigenschaften sind in den entsprechenden Schnittstellen als Methoden zum Aufbau des Objektmodells enthalten.

Erweiterungen, die durch das Instanziieren des Meta-Dokumentenmodells konkrete Operatoren, Container und Medienelemente definieren (z.B. für computergestütztes Training, einen Reiseführer, aber auch zeitliche und räumliche Konzepte), fügen dem Diagramm Klassen hinzu, die die Schnittstellen implementieren. In Abbildung 4-1 ist das durch die grau unterlegten Klassen namens `UserDefined*` dargestellt. Aus Instanzen dieser hinzugefügten Klassen wird das Objektmodell eines Dokuments für die Präsentation erzeugt.

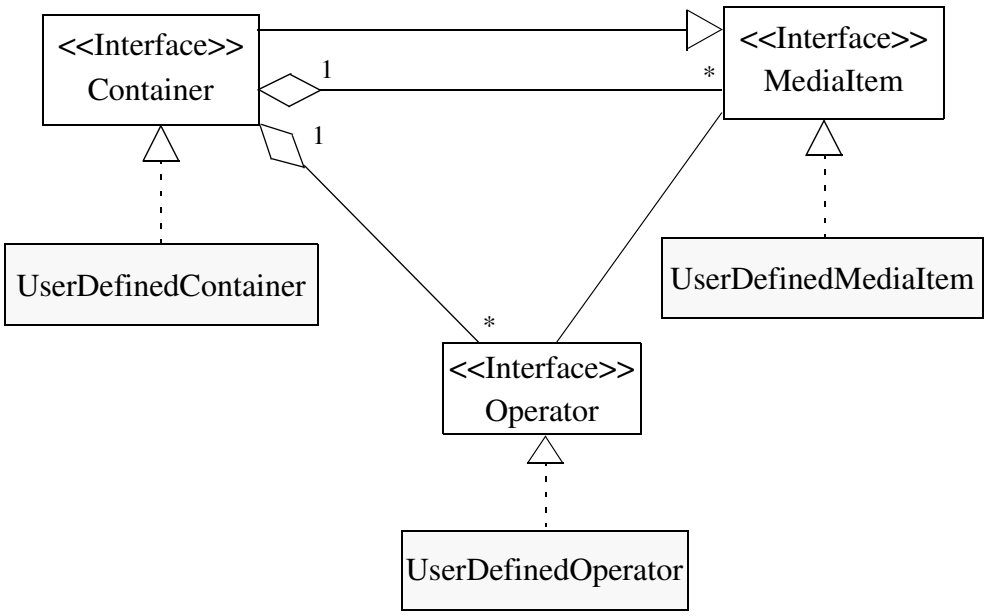


**Abb. 4-1 :** Abbildung des Dokumentenmodells auf Schnittstellen

Ein Manager implementiert die Operatoren für ein Anwendungskonzept. Beispielsweise werden die zehn temporalen Intervalloperatoren von TIEMPO [WaR93] durch einen TIEMPO-Manager implementiert. Da alle Operatoren eines Anwendungskonzepts in einem einzigen Manager realisiert werden, bestehen keine Abhängigkeiten zwischen den Managern.

Ein alternativer Ansatz wäre, jedem Operator eine Klasse zuzuordnen, die diesen implementiert. Dies scheitert an der “eingeschränkten Sicht” eines einzelnen Operators auf das Dokument. An einem Beispiel zur Implementierung eines Managers für temporale Intervall-Operatoren kann dies verdeutlicht werden: Das temporale Layout eines Dokumentes wird durch alle verwendeten Operatoren beschrieben, d.h. ein Operator allein kann nicht einen Teil des temporalen Layouts für sich selbst berechnen. Zur Berechnung wird eine Instanz benötigt, die den Gesamtüberblick über das Dokument besitzt und damit das temporale Layout berechnen kann. Wie dies für temporale Intervall-Operatoren aussieht wird in [Wir99b, Wir97] beschrieben. Im erweiterbaren Ansatz sind diese Instanzen die Manager.

Neben der technischen Realisierung gibt es auch einen konzeptionellen Grund für die Trennung der Dokumentenbeschreibung von der Implementierung. Durch die Trennung hat ein Operator eine ausschließlich beschreibende Funktion als Teil der Dokumentenspezifikation. Die Implementierung der Operatoren erfolgt in Managern als Erweiterung des Präsentationssystems.

Um Operatoren zu implementieren, tauschen Manager und Medienelemente Ereignisnachrichten aus (Abbildung 4-2). Die Ereignisnachrichten lassen sich in zwei Mengen aufteilen. Die erste Menge besteht aus Benachrichtigungsereignissen und die zweite Menge aus Steuerereignissen. Steuerereignisse werden von Managern an Medienelemente geschickt, um deren Präsentation zu steuern. Durch Benachrichtigungsereignisse teilen Medienelemente Managern besondere Vorkommnisse mit (beispielsweise ein Mausklick).

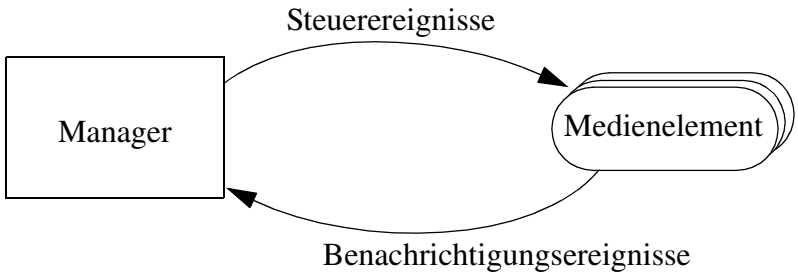


**Abb. 4-2 :** Nachrichtenaustausch zwischen Managern und Medienelementen

Für den Entwurf einer Architektur ist die Umsetzung der ereignisbasierten Kommunikation von Bedeutung. Mit der Einführung der JavaBeans, einem Komponenten-Konzept in Java, wurden gleichzeitig einheitliche Mechanismen zur Erzeugung und Verarbeitung von Ereignissen (engl.

events) festgelegt [Sun97]. Die Kommunikation der Manager und Medienelemente kann auf diese Mechanismen abgebildet werden. Damit verhalten sich Erweiterungs-Komponenten (Manager und Medienelemente) im Prinzip wie JavaBeans. Lediglich die Forderung nach der grafischen Manipulierbarkeit wird von den Erweiterungs-Komponenten nicht unterstützt.

Manager haben die Möglichkeit, sich als so genannte Event-Listener zu registrieren (engl. subscribe). Anschließend werden die von einem Medienelement erzeugten Benachrichtigungsereignisse an alle registrierten Managern weitergeleitet (engl. publish). Das Weiterleiten entspricht dem Aufruf einer dem Ereignis entsprechenden Methode. Dabei beinhaltet der Aufruf eine Beschreibung des Ereignisses als Parameter. Welche Methoden aufgerufen werden, wird durch die so genannte Listener-Schnittstelle festgelegt. Damit besteht die Möglichkeit, alle Benachrichtigungsereignisse nach funktionalen Aspekten (bspw. temporale Benachrichtigungsereignisse) in einer Schnittstelle zusammenzufassen.

Alle Medienelemente stellen temporale Benachrichtigungsereignisse zur Verfügung. Das heißt, dass ein Manager, der daran interessiert ist, sich für diese Ereignisse bei einem Medienelement registrieren kann.

Für einen zeitlichen Manager beinhaltet die Menge der Benachrichtigungsereignisse unter anderem das Ereignis `stopped` und die Menge der Steuerereignisse das Ereignis `start`. Empfängt nun ein zeitlicher Manager das Ereignis `stopped` von einem Medienelement, so kann er für die im Dokument verwendeten Operatoren entsprechende Steuerereignisse generieren und diese an die entsprechenden Medienelemente weiterleiten. Damit können nicht-deterministische Zeitpunkte bestimmt werden, die vor der Präsentation nicht feststanden. Dies ist beispielsweise beim Warten auf eine Interaktion der Fall.

Das Verschicken eines Steuerereignisses entspricht einem Methoden-Aufruf bei einem Medienelement. Welche Steuerereignisse einem Medienelement geschickt werden können, wird dadurch festgelegt, ob es die entsprechenden Methoden zur Verfügung stellt. Steuerereignisse und die dazugehörenden Methoden können nach funktionalen Aspekten (bspw. räumliche Anordnung von Medienelementen in der Präsentationsfläche) zusammengefasst werden.

In Java kann durch die Definition einer Schnittstelle festgelegt werden, welche Methoden zusammengehören. Ob ein Medienelement eine bestimmte Schnittstelle realisiert, kann durch

die Java-Funktion `instanceof` abgefragt und im positiven Fall die Methode aufgerufen werden.

Die Eigenschaften der auf dem Bildschirm sichtbaren Medienelemente werden in der Schnittstelle `VisibleMediaItem` zusammengefasst. Ein Manager kann dann feststellen, ob ein beliebiges Medienelement sichtbar ist, indem er überprüft, ob das Medienelement die `VisibleMediaItem`-Schnittstelle implementiert. Die Steuerereignisse sind im Fall des sichtbaren Medienelements die Positionierung in der Präsentationsfläche oder die Bestimmung der Größe eines Medienelements. Die Steuerereignisse werden durch die Methode `setPosition()` oder die Methode `setSize()` übermittelt. Positions- oder Größenangaben werden durch Parameter übergeben.

Dies bedeutet, dass ein bestimmtes Anwendungskonzept zur Beschreibung von Beziehung zwischen Medienelementen durch die Menge von Operatoren, die zwei Ereignismengen und den Manager festgelegt wird.

Abbildung 4-3 zeigt die Klassenhierarchie für Manager und Operatoren und deren Beziehung zu den Schnittstellen des Meta-Dokumentenmodells. Wie schon bei der Beschreibung der Schnittstellen und Klassen des Meta-Dokumentenmodells steht `UserDefined*` für diejenigen Stellen, an denen eine Erweiterung vorgenommen werden kann.

Damit lässt sich die Aufgabe der Manager wie folgt zusammenfassen: der Autor spezifiziert das gewünschte Verhalten seines Dokumentes durch die Beschreibung der Beziehungen zwischen den Medienelementen unter Zuhilfenahme von Operatoren. Zur Implementierung der Operatoren während der Präsentation kommunizieren die Manager der Operatoren mit den Medienelementen über Ereignisse. Damit ist die Hauptaufgabe eines Managers die Verarbeitung der empfangenen Ereignisse und die Generierung weiterer notwendiger Ereignisse für die Steuerung der Medienelemente. Dabei hängt das Generieren von Ereignissen von allen verwendeten Operatoren des Managers im Dokument ab.

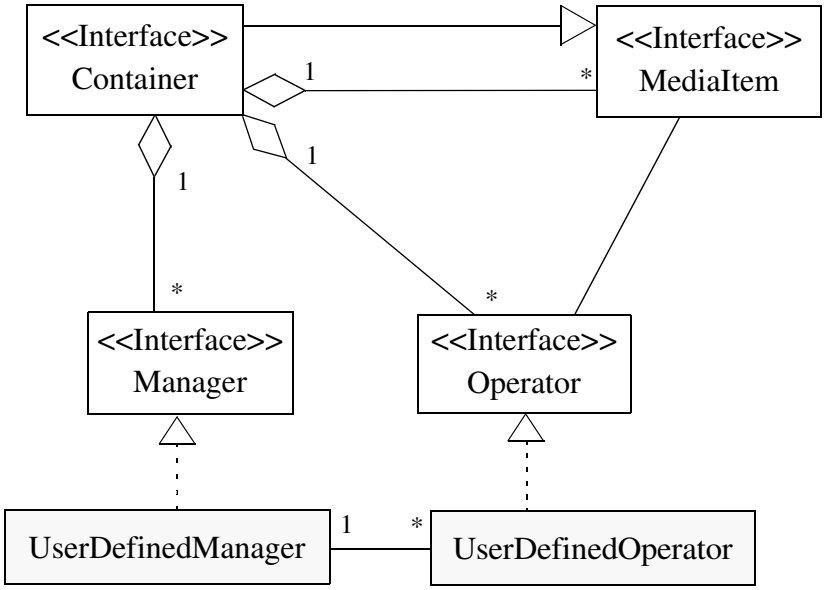


**Abb. 4-3 :** Klassenhierarchie der Manager und Operatoren

## 4.2 Kommunikation zwischen Medienelementen und Managern

Die Architektur des Präsentationssystems und die Komponenten, um die das System erweiterbar ist, werden durch das Präsentationssystem-Framework beschrieben. Es umfasst drei Schichten: (1) das Basis-System, (2) Manager und (3) Medienobjekte (Abbildung 4-4). Die zweite und dritte Schicht sind durch neue Komponenten erweiterbar.

Zwischen den Schichten werden jeweils definierte Schnittstellen verwendet. Eine Komponente (Manager oder Medienelement) kann nur dann in das Framework integriert werden, wenn sie die entsprechenden Schnittstellen beachtet, die weiter unten beschriebenen sind.

Die zentrale Komponente und erste Schicht des Frameworks ist das Basis-System. Es bietet die Grundfunktionalität an, die für das Präsentationssystem benötigt wird. Dazu gehören das Laden von Dokumenten, das Laden von Erweiterungen und die Steuerung des Ablaufs der Präsentation.

Als zweite wichtige Schicht definiert das Framework die Manager. Sie implementieren die Operatoren aus einem Anwendungsgebiet. In der dritten Schicht befinden sich Medienobjekte, die die Medienelemente und deren Funktion während der Präsentation umsetzen.

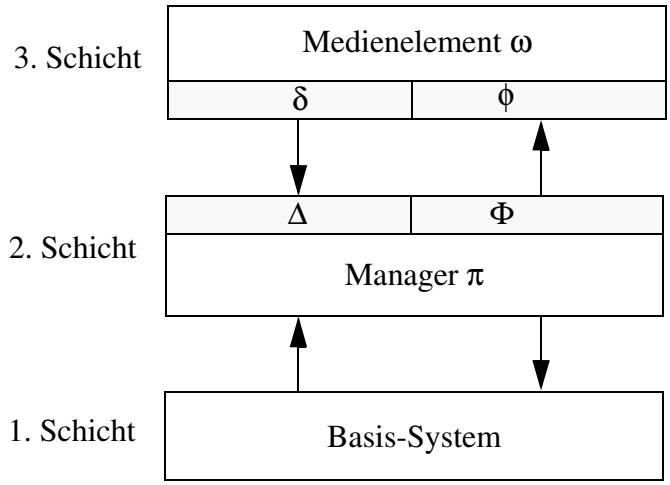


**Abb. 4-4 :** Die drei Schichten des Frameworks

In diesem Schichtenmodell werden die Operatoren nicht aufgeführt, da sie bezüglich der Kommunikation nur passive Komponenten sind. Sie beschreiben die Struktur des Dokumentes, ihre Implementierung während der Präsentation hingegen ist Aufgabe der Manager.

Die Unterstützung durch das Präsentationssystem-Framework kann in zwei Bereiche aufgeteilt werden. Zum einen sind das Verwaltungsaufgaben und die Steuerung der Präsentation, und zum anderen die Unterstützung für den Entwickler von Managern durch die Definition geeigneter Anwendungsprogrammierschnittstellen (kurz API, von engl. application programming interface).

Die folgende Beschreibung des strukturellen Aufbaus von Dokumenten und Spezifikation der Ereignisverarbeitung bilden die Grundlage für die Überprüfung der syntaktischen Konsistenz von Dokumenten. Außerdem ist die Ereignisverarbeitung Grundlage für den späteren Entwurf der Architektur des Präsentationssystems, die Implementierung der vorgestellten Konzepte und des Basis-Systems.

### 4.2.1 Struktur von Dokumenten

Für die weiteren Betrachtungen wird ein vereinfachtes Dokumentenmodell, das nur aus Medienelementen und Operatoren besteht herangezogen. Medienelemente stellen Informatio-

nen während der Präsentation des Dokuments dar: Operatoren beschreiben Beziehungen zwischen Medienelementen.

$\Omega$ ist die Menge aller Medienelemente, die in einem Dokument verwendet werden können. Ein Medienelement $\omega$ aus dieser Menge ist ein Element der Strukturbeschreibung. Es repräsentiert ein Medienelement und abstrahiert von seiner Verarbeitungslogik. Beispiele für Medienelemente sind $\omega_1$=video oder $\omega_2$=text.

$\Sigma$ ist die Menge aller Operatoren, die in einem Dokument verwendet werden können. Ein Operator $\sigma$ aus dieser Menge ist ein Element der Strukturbeschreibung. Er repräsentiert den Typ einer Beziehung zwischen Medienelementen. Beispiele hierfür sind $\sigma_1$=*while* oder $\sigma_2$=*correct*.

Nach der Betrachtung der Dokumentelemente wird nun die Struktur von Dokumenten selbst betrachtet. Mit einem Operator kann eine Beziehungen zwischen Medienelementen beschreiben werden. Diese wird als ein geordnetes 3-Tupel beschrieben. An erster Stelle in dem 3-Tupel steht der Operator, an der zweiten Stelle die Quellmedienelemente und an der dritten Stelle die Zielmedienelemente. Bei einer Beziehung wird zwischen Quell- und Zielmedien unterschieden, da es beispielsweise einen Unterschied macht, ob ein Text vor einem Bild dargestellt werden soll oder umgekehrt.

So ein 3-Tupel kann beispielsweise folgendermaßen aussehen:

$$y_1 = (while, (audio1), (video1))$$

Die Beziehung $y_1$ besagt, dass ein Quellmedienelement namens *audio1* und ein Zielmedienelement namens *video1* durch einen *while*-Operator miteinander verbunden sind. *Audio1* ist das Quellmedienelement und *video1* das Zielmedienelement.

Ein weiteres Beispiel zeigt den Sonderfall, dass ein Operator keine Quellmedienelemente und mehrere Zielmedienelemente besitzt:

$$y_2 = (start, (\,), (bild,text))$$

Die Beziehung $y_2$ besagt, dass die beiden Zielmedienelemente *bild* und *text* mit einem *start*-Operator verbunden sind. Er bedeutet, dass die Präsentation der Medienelemente *bild* und *text*

zum gleichen, durch den *start*-Operator festgelegten Zeitpunkt beginnen soll. Quellmedienelemente sind keine angegeben, deswegen steht an der zweiten Stelle des 3-Tupels eine leere Klammer.

Die Struktur eines Dokuments D wird durch eine Menge dieser 3-Tupel gebildet. Diese Menge wird mit $\psi_D$ bezeichnet. Sie beinhaltet alle Beziehungen zwischen Operatoren und Medienelementen des Dokuments. In Abbildung 4-5 ist ein Beispiel für eine Menge gegeben. In diesem Beispiel werden temporale Intervalloperatoren aus TIEMPO verwendet.

$\psi_{D1}$ = {
(*while*, (audio1), (video1)),
(*cobegin*, (video1), (text1)),
...}

**Abb. 4-5 :** Beispiel für eine Menge $\psi_{D1}$ von Beziehungen

Nach der Definition aller Komponenten eines Dokumentes kann nun das Dokument selbst beschrieben werden. Ein Dokument *D* kann als 3-Tupel folgender Form definiert werden:

$$D = (\Sigma_D, \Omega_D, \psi_D)$$

wobei $\Sigma_D$ die Menge der verwendeten Operatoren des Dokumentes D, $\Omega_D$ die Menge der Medienelemente des Dokumentes D und $\psi_D$ die Menge der Relationen zwischen Medienelementen ist.

### 4.2.2 Ereignismengen von Managern und Medienelementen

Nachdem der strukturelle Aufbau eines Dokuments erläutert wurde, wird nun die zweite Schicht des Präsentationssystem-Frameworks betrachtet. Sie umfasst die Manager, die die Beziehungen zwischen Medienelementen während der Präsentation implementieren. Ein Manager $\pi$ für ein Anwendungskonzept ist ein 3-Tupel

$$\pi = (\Sigma, \Delta, \Phi)$$

wobei $\Sigma$ die Menge der unterstützten Operatoren, $\Delta$ die Menge der Benachrichtigungsereignisse und $\Phi$ die Menge der Steuerereignisse ist. $\Delta$ und $\Phi$ sind die Ereignismengen, die der Manager bei der Implementierung seiner Operatoren für die Kommunikation mit den Medienelementen verwendet.

Sei $\Sigma(\pi)$ die Menge der Operatoren eines bestimmten Managers $\pi$. Zur Implementierung eines Operators $\sigma \in \Sigma(\pi)$ werden die folgenden Ereignis-Mengen benötigt:

$$\delta(\sigma)$$

$$\phi(\sigma)$$

Sei $\delta(\sigma)$ die Menge der Benachrichtigungsereignisse, also der Benachrichtigungen von einem Medienelement für einen Manager. Analog dazu sei $\phi(\sigma)$ die Menge der Steuerereignisse von einem Manager für ein Medienelement. Diese zwei Mengen legen fest, welche Steuer- und Benachrichtigungsereignisse zur Umsetzung des Operators $\sigma$ benötigt werden.

Abbildung 4-6 zeigt für das Beispiel eines *correct*-Operators die beiden benötigten Ereignismengen. Das Benachrichtigungsereignis `correctAnswer` zeigt an, dass die Antwort des Benutzers auf eine Frage richtig war. Das Steuerereignis `start` wird dazu benötigt, die Präsentation des auf die Frage folgenden Containers zu starten.

$\sigma$ = *correct*
$\delta(\sigma)$ = {`correctAnswer`}
$\phi(\sigma)$ = {`start`}

**Abb. 4-6 :** Ereignismengen des *correct*-Operators

Die Definition von $\delta(\sigma)$ und $\phi(\sigma)$ für alle $\sigma \in \Sigma(\pi)$ ermöglicht es, $\Delta(\pi)$ und $\Phi(\pi)$ für einen Manager $\pi$ zu definieren. $\Delta$ ist die Vereinigungsmenge aller Benachrichtigungsereignisse, die für die

Umsetzung der Operatoren benötigt werden. Dementsprechend ist Φ die Vereinigungsmenge aller Steuerereignisse.

$$\Delta(\pi) = \bigcup_{\sigma \in \Sigma(\pi)} \delta(\sigma)$$

$$\Phi(\pi) = \bigcup_{\sigma \in \Sigma(\pi)} \phi(\sigma)$$

Nun lassen sich die Anforderungen an ein Medienelement zur Integration in das Framework beschreiben.

Sei Ω die Menge aller Medienelemente. Ein Medienelement $\omega \in \Omega$ kann als folgendes Tupel dargestellt werden

$$\omega = (\delta, \phi)$$

wobei $\phi(\omega)$ die Menge der Steuerereignisse und $\delta(\omega)$ die Menge der Benachrichtigungsereignisse ist, die von einem Medienelement ω unterstützt werden.

Anhand des *Frage*-Containers aus dem Anwendungsgebiet computergestütztes Training soll die Verwendung der zwei Ereignismengen aus Sicht eines Medienelements, genauer aus der eines Containers, dargestellt werden (Abbildung 4-7). Für den Fortgang der Präsentation ist es erforderlich, das Ergebnis der Auswertung der Benutzerantwort zu erhalten. Die Mitteilung, ob der Benutzter die Frage richtig oder falsch beantwortet hat, erfolgt durch zwei Benachrichtigungsereignisse des *Frage*-Containers. Die unterstützten Steuerereignisse des *Frage*-Containers dienen zum Starten und Beenden seiner Darstellung während der Präsentation.

$\omega$ = Frage
$\delta(\omega)$ = {`correctAnswer`, `wrongAnswer`}
$\phi(\omega)$ = {`start`, `stop`}

**Abb. 4-7 :** Ereignismengen des *Frage*-Containers

Um einen Operator $\sigma$ eines bestimmten Anwendungskonzepts mit einem Medienelement $\omega$ verwenden zu können, muss das Medienelement beide Ereignismengen, d.h. $\delta(\sigma)$ und $\phi(\sigma)$, unterstützen. Daraus folgt, dass die folgenden zwei Bedingungen für jeden Operator $\sigma$, der mit einem Medienelement $\omega$ verwendet werden soll, gelten müssen.

$$\delta(\sigma) \subseteq \delta(\omega)$$

$$\phi(\sigma) \subseteq \phi(\omega)$$

Die erste Bedingung besagt, dass die für die Implementierung eines Operators benötigten Benachrichtigungsereignisse eine Teilmenge der von einem Medienelement zur Verfügung gestellten Benachrichtigungsereignisse sein müssen. Ein Medienelement muss also alle Benachrichtigungsereignisse erzeugen können. Die zweite Bedingung besagt das Gleiche für die Steuerereignisse, d.h. ein Medienelement muss alle Steuerereignisse verarbeiten können. Es ist in beiden Fällen die Teilmenge, da ein Medienelement mit mehreren unterschiedlichen Operatoren beziehungsweise Managern verwendet werden kann. Werden die beiden Bedingungen nicht erfüllt, kann der Operator $\sigma$ nicht mit dem Medienelement $\omega$ in Beziehung gesetzt werden.

Es ist möglich, dass mehrere Manager auf den gleichen Ereignismengen basieren. Es gibt beispielsweise mehrere unterschiedliche Modelle zur Spezifikation von zeitlichen Beziehungen in Multimedia-Dokumenten [PeL96]. Die zwei Ereignismengen zur Implementierung zeitlicher Manager sehen folgendermaßen aus: $\Delta(\pi)$ = {`started`, `stopped`} sind die Benachrichtigungsereignisse und $\Phi(\pi)$ = {`start`, `stop`} die Steuerereignisse. Ein Medienelement, das diese beiden Ereignismengen unterstützt, kann mit unterschiedlichen zeitlichen Manager verwendet werden. Diese Manager könnten beispielsweise eine Zeitachse oder ein intervallbasiertes Modell implementieren.

## 4.3 Syntaktische Konsistenz

Aus der Tatsache, dass es anwendungsspezifische Manager und Medienelemente gibt, folgt, dass nicht alle Operatoren mit allen Medienelementen verwendet werden können. Welche Operatoren mit welchen Medienelementen verwendet werden können, hängt vom Anwen-

dungsgebiet ab, das neben den Dokumentelementen durch die beiden Ereignismengen definiert wird.

Im weiteren Verlauf werden folgende Hilfsfunktionen benötigt:

8. *operands*(y): Diese Funktion bestimmt die Menge der Operanden einer Beziehung $y \in \Psi_D$.
9. *operator*(y): Diese Funktion bestimmt den Operator einer Beziehung $y \in \Psi_D$.

Die Verwendung der zwei Hilfsfunktionen wird an folgendem Beispiel verdeutlicht: $y_1$=*(while, (audio), (video))* ergibt *operands*($y_1$) = {*audio, video*} und *operator*($y_1$)=*while*.

Mit Hilfe dieser beiden Funktionen kann nun der Begriff der syntaktischen Konsistenz definiert werden.

**Def:** Ein Dokument D ist genau dann *syntaktisch konsistent*, wenn die folgende Bedingung zutrifft:

$$\forall(y \in \psi_D)\forall(\omega \in operands(y))(\delta(operator(y)) \subseteq \delta(\omega)) \wedge \phi(operator(y))) \subseteq \phi(\omega))$$

Diese Definition besagt, dass es in einem syntaktisch konsistenten Dokument keine Beziehung zwischen einem Operator und einem Medienelement geben darf, bei der das Medienelement nicht die beiden Ereignismengen des Operators unterstützt. Die richtige Verwendung muss für alle in einem Dokument enthaltenen Operatoren überprüft werden. Damit kann später bei der Präsentation der Fall nicht eintreten, dass ein Medienelement Steuerereignisse nicht verarbeiten kann oder keine Benachrichtigungsereignisse liefert, die für die Implementierung des Operators aber notwendig sind.

Anhand des Beispiels des *Frage*-Containers und des *correct*-Operators kann die syntaktische Konsistenz veranschaulicht werden. Damit der *correct*-Operator mit dem *Frage*-Container verbunden werden kann, muss dieser das Benachrichtigungsereignis `correctAnswer` bereitstellen. Mit einem Container, der dieses Benachrichtigungsereignis nicht zu Verfügung stellt, kann der *correct*-Operator nicht verwendet werden. In diesem Fall ist die Spezifikation syntaktisch nicht konsistent.

Die syntaktische Konsistenz kann schon während des Editierens eines Dokuments durch das Autorenwerkzeug überprüft werden. In diesem Fall findet die Überprüfung statt, wenn ein Operator mit einem Medienelement verbunden wird. Dabei muss nicht jedes Mal die komplette Gültigkeitsbedingung ausgewertet werden, sondern nur die Bedingungen für das betroffene Medienelement. Das Präsentationssystem überprüft die syntaktische Konsistenz ebenfalls, um Laufzeitfehler zu vermeiden.

Die Konsistenzprüfung einer Dokumentenspezifikation lässt sich in zwei Phasen aufteilen. In der ersten Phase wird die syntaktische Konsistenz geprüft. Diese kann im Basis-System realisiert werden, da sie unabhängig von den verwendeten Managern ist. In der zweiten Phase erfolgt die Prüfung der anwendungsspezifischen Konsistenz. Dabei wird überprüft, ob Operatoren entsprechend dem Anwendungskonzept verwendet werden. Die zweite Phase der Konsistenzprüfung hängt vom jeweiligen Anwendungskonzept ab und kann deswegen nicht allgemein überprüft werden. Sie muss deshalb vom jeweiligen Manager durchgeführt werden.

Für die Überprüfung der anwendungsspezifischen Konsistenz muss beispielsweise ein Manager für temporale Intervalloperatoren feststellen, ob für alle Medienelemente ohne Widersprüche eine Präsentationsdauer bestimmt werden kann. Eine Inkonsistenz tritt z.B. auf, wenn zwei unterschiedliche Operatoren, die den Präsentationsanfangszeitpunkt bestimmen, benutzt werden, oder das Präsentationsende eines Medienelements vor seinem Präsentationsanfang liegt. Weitere Informationen zur Konsistenzprüfung von temporalen Intervalloperatoren finden sich in [Wir99a].

## 4.4 Darstellbarkeit von Dokumenten

Um ein Multimedia-Dokument präsentieren zu können, muss das Präsentationssystem um die benötigten Erweiterungen ergänzt werden. Da dies automatisch, ohne Zutun des Benutzers unmittelbar vor dem Beginn der Präsentation geschieht, wird diese Eigenschaft des erweiterbaren Präsentationssystems auch ad-hoc Erweiterbarkeit genannt.

Für die Präsentation eines Dokumentes muss nicht nur die Dokumentenspezifikation konsistent sein, sondern auch der komplette Programmcode für alle verwendeten Erweiterungen (Medie-

nelemente, Container, Operatoren und Manager) muss zur Verfügung stehen. Dies führt zur Definition eines darstellbaren Dokuments.

**Def:** Ein Dokument $D=(\Sigma_D,\Omega_D,\psi_D)$ ist genau dann *darstellbar*, wenn es syntaktisch konsistent ist, die benötigten Programmkomponenten für alle Manager und alle Dokumentelemente vom Basis-System lokalisiert und nachgeladen werden können und jeder Manager die anwendungsspezifische Konsistenz geprüft hat.

## 4.5 Architektur des Präsentationssystems

Die Ausführungen in den vorangegangenen Abschnitten führen zu der in Abbildung 4-8 gezeigten Architektur des erweiterbaren Präsentationssystems. Sie besteht aus den folgenden Komponenten:

- *Basis-System*

  Das Basis-System stellt die Grundfunktionalität bereit. Diese umfasst unter anderem das Laden des Dokuments, das Laden von Erweiterungen, das Abspielen oder das Beenden der Präsentation. Zusätzlich stellt das Basis-System eine Benutzungsoberflächen-Komponente zur Verfügung, in der die Präsentation dargestellt wird.

- *Dokumentenlader*

  Der Dokumentenlader ist für das Laden eines Dokuments verantwortlich. Er parst das zu ladende Dokument und erstellt das zugehörige Objektmodell. Um den Programmcode der Klassen des Objektmodells zu erhalten, beauftragt er den Klassenmanager, die entsprechenden Erweiterungen zu laden.

- *Klassenmanager*

  Der Klassenmanager stellt die vom Dokumentenlader benötigten Klassen zur Verfügung. Dabei verwaltet er die zu einer Erweiterung gehörenden Programmcode-Pakete, lädt nicht vorhanden Pakete und führt ein Caching der Erweiterungen im lokalen Dateisystem durch. Die Funktionsweise des Klassenmanagers und des Dokumentenladers werden im nächsten Kapitel betrachtet.

Die bisher beschriebenen Module bilden das Framework, in das die Erweiterungen integriert werden.

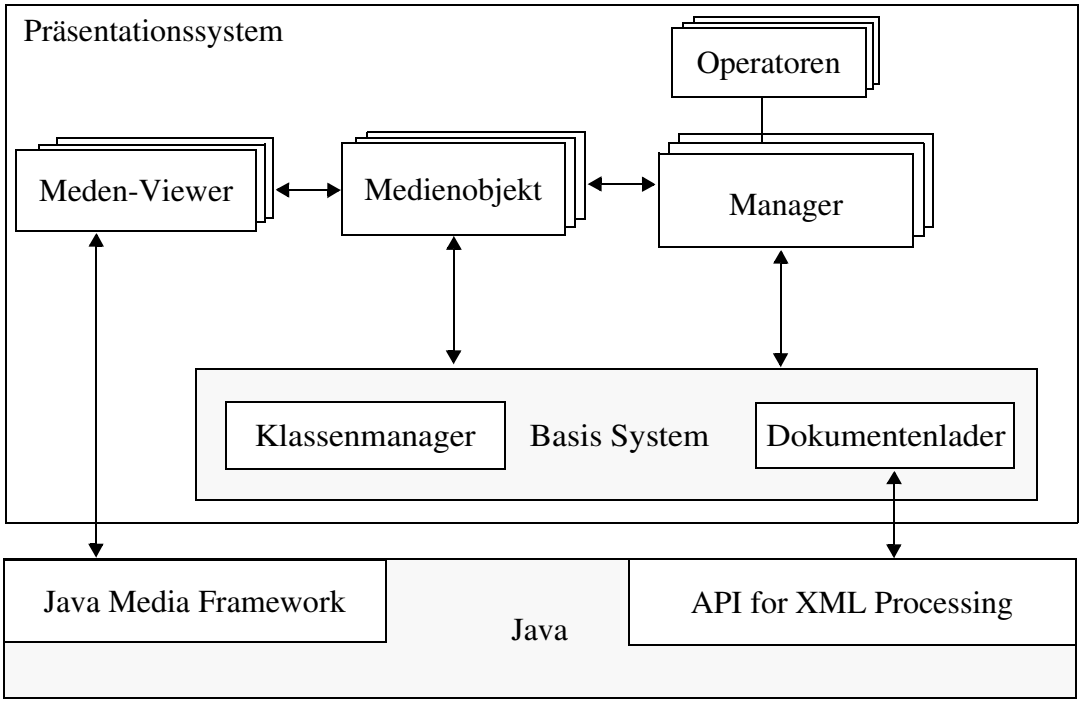


**Abb. 4-8 :** Architektur des erweiterbaren Präsentationssystems

Die nächsten Module können alle mehrfach vorhanden sein. Welche Erweiterungen tatsächlich integriert werden, hängt vom jeweiligen Dokument ab. Die Erweiterungen werden vor der Präsentation geladen und in das Präsentationssystem integriert.

Die folgenden Komponenten bilden das Objektmodell des Meta-Dokumentenmodells.

- *Operatoren*

  Operatoren haben eine deskriptive Funktion im Objektmodell. Sie enthalten die aktuellen Parameter und Referenzen auf die Medienelemente, mit denen sie verbunden sind.

- *Medienobjekte*

  Medienobjekte sind zum einen die Repräsentanten der Medienelemente im Objektmodell, und zum anderen realisieren sie die logische Funktionalität eines Medienelements. So speichern sie einen Zustand, der sich beispielsweise durch Interaktion während der Präsentation

ändern kann. Ein Kontrollkästchen beschreibt durch seinen Zustand, ob es ausgewählt ist oder nicht. Das zugehörige Medienobjekt stellt diesen Zustand zur weiteren Verarbeitung zur Verfügung, z.B. für den Manager des computergestützten Trainings.

Die nachfolgenden Komponenten werden zur Durchführung der Präsentation benötigt.

- *Manager*

  Manager sind funktionale Einheiten, die ein Konzept für ein bestimmtes Anwendungskonzept während einer Präsentation implementieren. Das Anwendungskonzept wird durch eine Menge von Operatoren beschrieben. Die Implementierung von Managern wird in den nächsten beiden Unterkapiteln genauer betrachtet.

- *Medien-Viewer*

  Medien-Viewer sind für die Präsentation des zugehörigen Medienobjekts zuständig. Dies umfasst die graphische und/oder akustische Darstellung der Mediendaten. Sie werden in Abhängigkeit vom Medientyp und Format der Mediendaten ausgewählt.

In der prototypischen Implementierung der Architektur ist neben der Java-Laufzeitumgebung das Java Media Framework [Sun01c] von Bedeutung. Es wird verwendet, um die Medien-Viewer für Video- und Audio-Medienelemente zu realisieren. Für das Laden XML-basierter Dateien wie Dokumente oder die Benutzereinstellungen wird auf das Java API für XML Processing [Sun01a] zurückgegriffen.

## 4.6 Implementierung von Managern

Im Folgenden wird die Implementierung von Operatoren durch Manager näher betrachtet. Für die Vorgehensweise bei der Implementierung gibt es zwei unterschiedliche Ansätze. Der erste Ansatz geht davon aus, dass generische Operatoren bereits vorhanden sind. Dabei werden die anwendungsspezifischen Operatoren mit Hilfe der generischen Operatoren realisiert. Im zweiten Ansatz erfolgt eine Implementierung der Operatoren durch den Manager selbst.

Um die Abbildung von Operatoren auf generische Operatoren zu untersuchen, muss zuerst die Annahme getroffen werden, welche generischen Operatoren zur Verfügung stehen. Für die fol-

genden Beispiele sind dies temporale und räumliche Operatoren. Als Beispiele werden die beiden Anwendungsgebiete aus der Einleitung verwendet.

Das erste Beispiel war das computergestützte Training mit Multiple-Choice-Fragen. In Kapitel 3 wurde der *correct*-Operator eingeführt, mit dem der Autor bestimmen kann, an welcher Stelle die Präsentation nach einer richtigen Antwort fortgesetzt werden soll. Der *correct*-Operator besitzt keinen räumlichen Aspekt. Bei der Implementierung dieses Operators muss überprüft werden, ob die durch den Benutzer ausgewählten Kontrollkästchen der richtigen Antwort entsprechen. Wenn dies so ist, muss an eine bestimmte Stelle in der Präsentation gesprungen werden. Der Sprung kann als eine temporale Beziehung realisiert werden. Die Überprüfung, ob die Antwort richtig ist, besitzt weder einen räumlichen noch einen zeitlichen Aspekt. Der *correct*-Operator lässt sich somit nicht nur durch eine Abbildung auf temporale und räumliche Operatoren realisieren. Er muss demnach als generischer Operator zur Verfügung stehen.

Im Beispiel des interaktiven Reiseführers legt der Autor mit Hilfe des *activationRadius*-Operators (Kapitel 3.3) sensitive Bereiche auf einer Landkarte fest. Während der Präsentation muss die aktuelle physische Position des Benutzers bestimmt werden. Anschließend muss geprüft werden, ob er sich in einem sensitiven Bereich befindet und gegebenenfalls die Präsentation der zugehörigen Sehenswürdigkeit starten. Auch dies lässt sich nicht nur mit temporalen und räumlichen Operatoren durchführen. Die Bestimmung der physischen Position ist eine spezifische Operation (Abfragen des GPS-Empfängers). Ob sich die Position innerhalb eines sensitiven Bereichs befindet muss berechnet werden. Deshalb muss der *activationRadius*-Operator in diesem zweiten Beispiel als generischer Operator zur Verfügung stehen.

Die Operatoren aus beiden Beispielen wurden als generisch eingestuft. Dies bedeutet, dass sie implementiert werden müssen. Das legt den Schluss nahe, von vornherein den zweiten Ansatz zu verfolgen, bei dem Operatoren durch einen Manager implementiert werden. Durch die Erweiterbarkeit des Ansatzes wird sichergestellt, dass Operatoren nur einmal implementiert werden müssen. Nach ihrer Implementierung, können sie von anderen Autoren wiederverwendet werden.

### 4.6.1 Interne Repräsentationen von Anwendungskonzepten

Durch das Meta-Dokumentenmodell wird eine Graphstruktur beschrieben, in der Medienelemente mittels Operatoren miteinander in Beziehung gesetzt werden. Dieser Beziehungsgraph ist nicht in allen Fällen als Datenstruktur für die Implementierung eines Anwendungskonzepts geeignet. Ein Manager, der als Animation Medienelemente über die Präsentationsfläche bewegt, merkt sich die aktuell zu bewegenden Medienelemente beispielsweise in einer Liste. Ein Manager, der ein temporales Konzept implementiert, kann beispielsweise einen temporalen Scheduling-Graphen verwenden, wie er in [Wir99a] beschrieben wird. Dieser Graph ist detaillierter als der Beziehungsgraph des Meta-Dokumentenmodells, da er für jedes Medienelement zwei Knoten vorsieht, die dem Start- und Endzeitpunkt der Präsentation eines Medienelements entsprechen.

Aus diesem Grund wird bei der Entwicklung von Managern im Allgemeinen nicht das Meta-Dokumentenmodell als Datenstruktur für die Implementierung verwendet. Im Folgenden wird deshalb die Vorgehensweise der Verwendung spezieller Datenstrukturen, die das Anwendungskonzept besser repräsentieren als das Meta-Dokumentenmodell, näher beschrieben.

Wie in vielen Bereichen der Softwareentwicklung wird auch hier der objektorientierte Ansatz verfolgt. Durch diesen werden Datenstrukturen als Objekte mit einer Funktionalität modelliert, die über Klassen zugeordnet ist. Diese objektorientierte Datenstruktur wird als interne Repräsentation eines Anwendungskonzepts bezeichnet. Sie ist für den Autor nicht sichtbar und er muss sie auch nicht kennen, um ein Dokument zu erstellen. Sie ist lediglich für den Entwickler eines Managers relevant. In dem Beispiel des temporalen Managers ist der Scheduling-Graph die interne Repräsentation. In Kapitel 4.7 wird ein ausführliches Beispiel gegeben, wie die interne Repräsentation für ein computergestütztes Trainingskonzept aussehen und wie es objektorientiert realisiert werden kann.

Bei der Verwendung von internen Repräsentationen können die Aufgaben eines Managers in zwei Phasen eingeteilt werden. In der ersten Phase führt der Manager die Aufgaben durch, die vor der Präsentation stattfinden müssen. Die zweite Phase findet während der Präsentation statt.

Nachdem das Präsentationssystem die syntaktische Konsistenz überprüft hat analysieren die Manager die Verwendung ihrer Operatoren in dem zu präsentierenden Dokument. Während die-

ser Analyse wird geprüft, ob die Operatoren konsistent im Sinne des Anwendungskonzepts verwendet werden. Im Gegensatz zur syntaktischen Konsistenz ist dies spezifisch für jeden Manager und muss deshalb durch ihn selbst ausgeführt werden. Anschließend wird die interne Repräsentation durch den Manager aufgebaut. Diese Aufgaben werden in der ersten Phase, der so genannten Analyse-Phase, erledigt.

Da die zweite Phase zur Präsentationszeit stattfindet, wird sie Präsentations-Phase genannt. In dieser werden mit Hilfe der internen Repräsentation des Anwendungskonzepts die Operatoren implementiert. Dabei wird die in Kapitel 4.2 beschriebene Ereignisverarbeitung umgesetzt. Auf diese wird an dieser Stelle nicht näher eingegangen, da es sich dabei um die technische Realisierung handelt, die für jeden Manager anders ist.

Im Kapitel 4.7 wird exemplarisch für das Anwendungsgebiet computergestütztes Training eine interne Repräsentation und ihre Implementierung beschrieben. Dabei werden beide Phasen betrachtet.

Durch die Aufteilung der Aufgaben eines Managers in zwei Phasen und der Verwendung einer internen Repräsentation ergeben sich zwei Vorteile. Der erste Vorteil ist der, dass die interne Repräsentation durch den Entwickler eines Managers frei wählbar ist. Dabei hängt die Wahl vom jeweiligen Anwendungskonzept ab. Aus diesem Grund lässt sich hier keine allgemeine Vorgehensweise bei der Wahl angeben. Der zweite Vorteil ist die Wiederverwendung und Erweiterung einer existierenden internen Repräsentationen. Durch Vererbung und Spezialisierung von Klassen wird die Funktionalität der internen Repräsentation angepasst. Ein Konzept zur Spezifikation von computergestützten Trainings kann beispielsweise durch eine Erweiterung des Konzepts für interaktive Anwendungen realisiert werden. Wie dies im Detail aussieht, ist Gegenstand des nächsten Unterkapitels.

## 4.7 Beispiel einer Erweiterung

An dieser Stelle wird exemplarisch die Vorgehensweise bei der Konzeption und dem Entwurf einer Erweiterung für das Anwendungsgebiet computergestütztes Training gezeigt. Die Konzeption einer Erweiterung ist nur notwendig, falls sich keine entsprechende Erweiterung in der Sektorenhierarchie befindet, die wiederverwendet werden kann.

Für die Beschreibung interaktiver Multimedia-Anwendungen kann auf die in MHEG-5 eingeführten Konzepte zurückgegriffen werden. In MHEG-5 wird ein Multimedia-Dokument Anwendung (engl. application) genannt. Eine Anwendung ist dabei in mehrere Szenen aufgeteilt. Durch Interaktion kann der Benutzer zwischen diesen Szene hin- und herspringen. MHEG-5 stellt keine operatorenbasierte Sprache zur Verfügung, diese muss neu konzipiert werden.

Eine interaktive Anwendung soll aus mehreren Szenen bestehen. Szenen sind ein Strukturierungselement und werden deshalb durch Container repräsentiert. Der Autor spezifiziert den Inhalt der einzelnen Szenen und anschließend die Reihenfolge und Interaktionsmöglichkeiten zwischen den Szenen durch Operatoren.

Zur besseren Strukturierung werden zwei unterschiedliche Typen von Szenen eingeführt: (1) eine Szene ohne Interaktionen wird durch einen *Scene*-Container dargestellt, und (2) eine Szene mit Interaktionen wird durch einen *InteractionScene*-Container repräsentiert. Dies ermöglicht eine Unterscheidung der beiden Szenentypen an der Benutzungsoberfläche eines Autorensystems. Zusätzlich kann durch die Überprüfung der Gültigkeit sichergestellt werden, dass der *interaction*-Operator nur mit einem *InteractionScene*-Container verwendet wird.

Folgende Operatoren werden für die Spezifikation einer interaktiven Anwendung benötigt: Mit Hilfe des *start*-Operators kann eine Szene bestimmt werden, mit der die Präsentation beginnen soll. Der *next*-Operator erlaubt die Bestimmung der Reihenfolge von Szenen. Die von einer Szene durch Interaktion erreichbaren weiteren Szenen werden durch den *interaction*-Operator bestimmt. Zu jedem *interaction*-Operator gehört eine Schaltfläche (ein *Button*-Medienelement) innerhalb des *InteractionScene*-Containers, um zu bestimmen, mit welcher Szene die Präsentation fortgesetzt wird.

Abbildung 4-9 fasst die Container und Medienelemente (a) und Operatoren (b) zusammen, die zur Spezifikation einer interaktiven Anwendung verwendet werden können.

### 4.7.1 Interne Repräsentation der Beispielanwendung

Für die Implementierung der Operatoren müssen eine interne Repräsentation von interaktiven Anwendungen und die Benachrichtigungsereignisse der Container festgelegt werden.

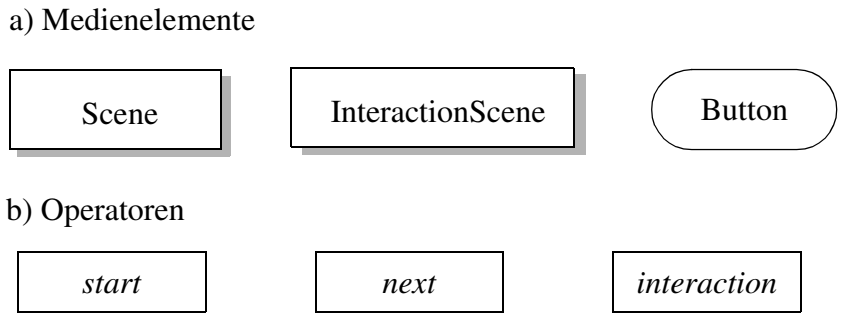


**Abb. 4-9 :** Medienelemente und Operatoren der interaktiven Anwendung

Der *Scene*-Container erzeugt das Benachrichtigungsereignis `EndOfScene`, um mitzuteilen, dass die Präsentation einer Szene beendet wurde. Der *InteractionScene*-Container erzeugt das Benachrichtigungsereignis `InteractionHappened`, um mitzuteilen, dass eine Interaktion stattgefunden hat. Diese beiden Benachrichtigungsereignisse bilden die Grundlage für die Steuerung der Präsentation interaktiver Anwendungen.

Für die interne Repräsentation einer interaktiven Anwendung wird ein gerichteter Graph verwendet. Der Graph besitzt zwei unterschiedliche Typen von Knoten (Abbildung 4-10). Der erste Knotentyp ist ein Präsentationsknoten, der eine Referenz auf einen *Scene*-Container besitzt. Von einem Präsentationsknoten kann nur eine Kante ausgehen. Diese Kante zeigt auf den Knoten, der in der Präsentation als nächstes folgt. Der zweite Knotentyp ist der Interaktionsknoten. Er besitzt eine Referenz auf eine Szene, die Interaktionsmöglichkeiten zur Auswahl der folgenden Szene enthält. Für jede Wahlmöglichkeit gibt es eine vom Interaktionsknoten ausgehende gerichtete Kante zu einer weiteren Szene. Bei einer Interaktion wird die Präsentation durch die Szene fortgeführt, die durch den Benutzer ausgewählt wurde.

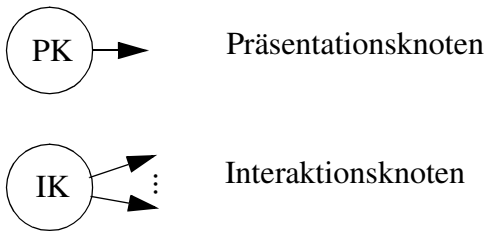


**Abb. 4-10 :** Interne Repräsentation für interaktive Anwendungen

In der Analyse-Phase erzeugt der Anwendungsmanager die interne Repräsentation einer interaktiven Anwendung und registriert sich als Event-Listener bei den Szenen. Während der Realisierungs-Phase startet er die Präsentation bei dem Knoten, der durch den *start*-Operator als Anfangszene bestimmt wurde. Anschließend wartet er auf Benachrichtigungsereignisse, um zu bestimmen, mit welchen Knoten die Präsentation fortfährt. Hat ein Knoten keinen Nachfolger, wird die Präsentation des Dokumentes beendet.

Abbildung 4-11 zeigt eine Beispielspezifikation, die die Konzepte einer interaktiven Anwendung verwendet. Das Dokument besteht aus vier verschiedenen Szenen. Durch den *start*-Operator wird bestimmt, dass die Präsentation des Dokumentes mit der Szene 1, einer kurzen Einleitung, beginnt. Durch den *next*-Operator wird nach dem Ende der Szene 1 die Präsentation mit der Szene 2 fortgesetzt. Wie schon der Name der Szene 2 (“Menü”) verrät, kann der Benutzer in dieser Szene eine Auswahl treffen, wie die Präsentation fortgeführt werden soll. Als die zwei möglichen Folgeszenen wurden Szene 3 und Szene 4 festgelegt. Dementsprechend bietet die Menüszene dem Benutzer zwei Schaltflächen an, über die er die Folgeszene auswählen kann.

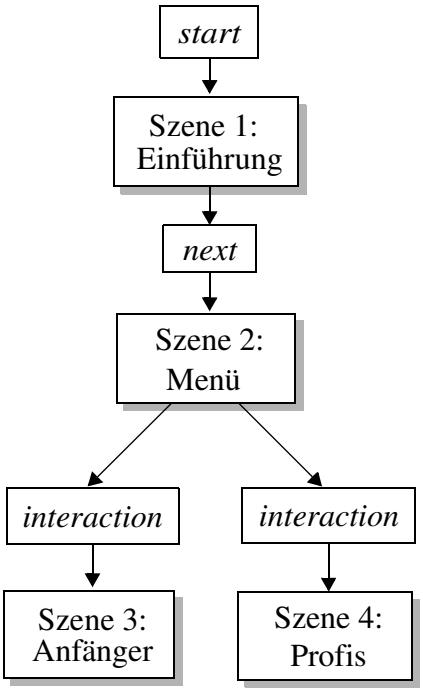


**Abb. 4-11 :** Beispielspezifikation eines Menüs

Der Manager für interaktive Anwendungen erzeugt die in Abbildung 4-12 dargestellte interne Repräsentation der Beispielspezifikation. Sie besteht aus vier Knoten: Drei Präsentationsknoten und einem Interaktionsknoten. Der Anwendungsmanager hat sich bei allen vier zugehörigen Containern als Event-Listener registriert. Er startet die Präsentation mit dem Präsentationsknoten *PK1*. Erhält er vom *Scene*-Container, der zum Präsentationsknoten *PK1* gehört, das Benachrichtigungsereignis `EndOfScene`, führt er die Präsentation mit dem Interaktionsknoten *IK* fort.

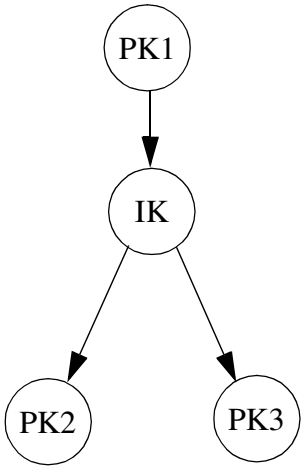


**Abb. 4-12 :** Interne Repräsentation eines Menüs

### 4.7.2 Erweiterung des Konzepts für computergestütztes Training

Das im vorherigen Abschnitt vorgestellte Konzept für interaktive Anwendungen wird nun zu einem computergestützten Training erweitert. Durch die Erweiterung soll es möglich sein, Fragen und die Reaktion auf die Beantwortung der Fragen zu spezifizieren.

Der Autor kann die in Abbildung 4-13 dargestellten Medienelemente und Operatoren zur Spezifikation von Reaktionen auf Fragen verwenden. Als neues Medienelement gibt es einen *Question*-Container. Eine Frage wird immer innerhalb dieses Containers spezifiziert. Mit Hilfe zweier weiterer Operatoren, den *correct*- und *false*-Operatoren, wird bestimmt, mit welcher Szene im Falle einer richtig beziehungsweise falsch beantworteten Frage fortgefahren wird.

Um Dokumente für computergestützte Trainings darstellen zu können, wird der in 4.7.1 vorgestellte Graph zur internen Repräsentation von interaktiven Anwendungen durch einen weiteren

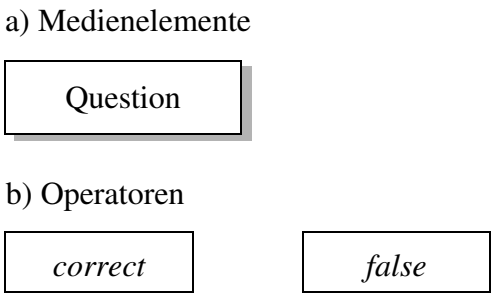


**Abb. 4-13 :** Medienelemente und Operatoren des computergestützten Trainings

Knoten-Typ ergänzt. Der Algorithmus zur Durchführung der Präsentation kann unverändert bleiben, da die Entscheidung, wie die Präsentation fortgeführt werden soll, in den einzelnen Knoten getroffen wird.

Diese Entscheidungslogik muss für den neuen Knoten implementiert werden. Dafür stellt der *Question*-Container zwei Benachrichtigungsereignisse zur Verfügung. Das erste ist das `correctAnswer`-Ereignis und das zweite das `wrongAnswer`-Ereignis. Diese beiden Ereignisse werden bei der Verarbeitung im Manager herangezogen, um zu bestimmen, welche Szene nach der Frage präsentiert werden soll.

Durch die Wahl einer graph-basierten internen Repräsentation und Aufteilung der Funktionalität kann in diesem Fall eine Erweiterung mit geringem Aufwand realisiert werden. Die hier beschriebene Vorgehensweise ist ein Beispiel für die in Abschnitt 4.6 beschriebene Erweiterung einer existierenden internen Repräsentation.

In Abbildung 4-14 ist ein Ausschnitt aus einem computergestützten Training dargestellt. Es beinhaltet eine Frage, die vom Benutzer beantwortet werden muss. Je nachdem, ob die Frage richtig oder falsch beantwortet wurde, wird eine unterschiedliche Szene präsentiert. Im Falle einer falschen Antwort kann beispielsweise eine Erklärung der richtigen Antwort erfolgen, um den Lerneffekt für den Benutzers zu erhöhen. Bei einer richtigen Antwort wird eine Szene abgespielt, die den Benutzer lobt.

In Abbildung 4-15 ist die zu der Fragespezifikation gehörende interne Repräsentation dargestellt. Die interne Repräsentation verwendet einen Frageknoten zur Darstellung der Frage.

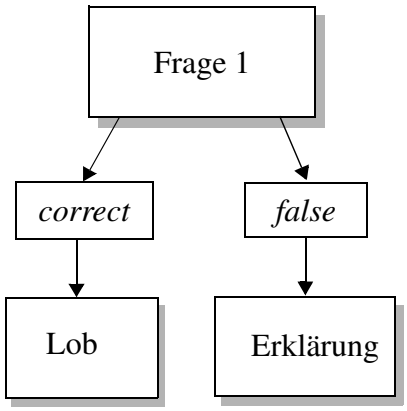


**Abb. 4-14 :** Beispiel für die Verwendung einer Frage

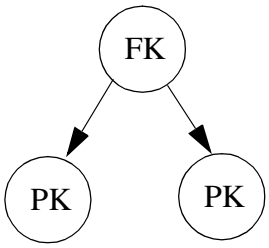


**Abb. 4-15 :** Internen Repräsentation eines Frage-Dokuments

Die hier beschriebene Vorgehensweise ist ein Beispiel dafür, wie der Aufwand zur Implementierung des Konzeptes für ein computergestütztes Training durch die Erweiterung des Graphen für interaktive Anwendungen reduziert werden kann.

In dem bisher beschriebenen Beispiel wurde der Verlauf der Präsentation nur auf Containerebene betrachtet. Um computergestützte Trainingseinheiten erstellen zu können, müssen zusätzlich Medienelemente und Operatoren für die Spezifikation von Fragen und Antworten definiert werden. Davon wird in dieser Abhandlung abgesehen, da sich daraus keine neuen Erkenntnisse gewinnen lassen.

### 4.7.3 Implementierung des Managers für computergestütztes Training

Anhand der in diesem Kapitel betrachteten Erweiterung für computergestütztes Training wird in diesem Abschnitt gezeigt, wie die Ereignismengen mit der Implementierung zusammen-

hängen. Als Beispiel dienen die Benachrichtigungsereignisse des *Question*-Containers, die noch einmal in Abbildung 4-16 angegeben sind.

$$\omega = \text{Question}$$
$$\delta(\omega) = \{\texttt{correctAnswer}, \texttt{wrongAnswer}\}$$
$$\phi(\omega) = \{\}$$

**Abb. 4-16 :** Ereignismengen des *Question*-Containers

$\delta(\omega)$ ist eine Menge von Ereignissen, die durch die Implementierung des *Question*-Containers an potentielle Interessenten weitergeleitet wird. Dazu wird eine `QuestionContainerListener`-Schnittstelle definiert, die in Java wie in Abbildung 4-17 angegeben aussieht. Falls das `correctAnswer`-Ereignis eintritt, wird die Methode `answerCorrect()` aufgerufen. Jedem Ereignis ist eine Methode in der Schnittstelle zugeordnet.

```
public interface QuestionContainerListener
{
  public void answerCorrect(AnswerEvent ae);
  public void answerWrong(AnswerEvent ae);
}
```

**Abb. 4-17 :** Die `QuestionContainerListener`-Schnittstelle

Eine Klasse, die einen *Question*-Container implementieren möchte, muss die `QuestionContainerListener`-Schnittstelle unterstützen. Abbildung 4-18 zeigt die `QuestionContainer`-Schnittstelle. Sie erlaubt es einem Interessenten, der die `QuestionContainerListener`-Schnittstelle implementiert, sich bei dem *Question*-Container zu registrieren oder sich zu entfernen. Nach dem Registrieren erhält der Interessent die entsprechenden Benachrichtigungsereignisse. Der Manager für computergestütztes Training wird sich bei jedem *Question*-Container als Interessent registrieren und, sobald eines der Benachrichtigungsereignisse eintrifft, den Fortgang der Präsentation bestimmen.

Die Vorgehensweise für Steuerereignisse ist einfacher, dazu muss lediglich eine Schnittstelle definiert und von der entsprechenden Medienelement-Klasse implementiert werden. Die

```
public interface QuestionContainer
{
  public addQuestionContainerListener(
                        QuestionContainerListener fcl);
  public removeQuestionContainerListener(
                        QuestionContainerListener fcl);
}
```

**Abb. 4-18 :** Die `QuestionContainer`-Schnittstelle

Schnittstelle aus Abbildung 4-19 wird benötigt, um die Steuerereignisse $\phi(\sigma)$={`start`, `stop`}, die von temporalen Managern verwendet werden, an Medienelemente zu schicken. Da es sich hierbei um elementare Steuerereignisse handelt, sind sie Teil der `MediaItem`-Schnittstelle, die von jedem Medienelement implementiert werden muss. Zur Vereinfachung ist hier nur ein Ausschnitt der Schnittstelle gezeigt.

```
public interface MediaItem
{
  public void start();
  public void stop();
...
}
```

**Abb. 4-19 :** Ausschnitt der `MediaItem`-Schnittstelle

Durch die `instanceof`-Funktion in Java kann sehr leicht geprüft werden, ob eine Klasse eine bestimmte Schnittstelle implementiert. So kann überprüft werden, ob eine Containerklasse die `QuestionContainer`-Schnittstelle oder eine Medienelementklasse die `MediaItem`-Schnittstelle implementiert. Dies ermöglicht eine Überprüfung der syntaktischen Konsistenz eines Dokuments zur Laufzeit.

Die Umsetzung eines anwendungsspezifischen Konzepts ist mit weniger Aufwand verbunden, als die Entwicklung eines neuen anwendungsspezifischen Dokumentensystems. Wie dies im Einzelnen funktioniert, wurde durch das vorangegangene Beispiel demonstriert.

## 4.8 Zusammenfassung

In diesem Kapitel wurde der Zusammenhang zwischen einer Spezifikationssprache und der Implementierung ihrer Operatoren betrachtet. Die Implementierung der Operatoren erfolgt als Austausch von Steuer- und Benachrichtigungsereignissen zwischen Managern und Medienelementen. Darauf basierend wurde die Architektur eines erweiterbaren Präsentationssystems eingeführt.

Zur Implementierung eines Prototypen des Präsentationssystems und des Autorenwerkzeugs wurde die Programmiersprache Java gewählt. Die Plattformunabhängigkeit und das dynamische Laden von Java-Klassen zur Laufzeit ermöglichen die Entwicklung eines Präsentationssystems und Autorenwerkzeugs, mit den in dieser Abhandlung geforderten Eigenschaften.

Am Ende wurde für das Anwendungsgebiet computergestütztes Training eine Erweiterung konzipiert und der Entwurf eines zugehörigen Managers durchgeführt. An diesem Beispiel wurde nochmals verdeutlicht, welchen Vorteil der Ansatz hat, Operatoren in einem Manager zu implementieren und dabei eine interne Repräsentation zu verwenden. Die Umsetzung des Reiseführers würde sehr ähnlich verlaufen.

# 5 VERMITTLUNG MULTIMEDIALER DOKUMENTE

Multimedia-Dokumente, die basierend auf dem in Kapitel 3 eingeführten Meta-Dokumentenmodell spezifiziert wurden, müssen für die Übertragung zum Benutzer in einem speziellen Format gespeichert werden. Dieses ermöglicht die Bereitstellung von Multimedia-Dokumenten über das Internet auf so genannten Dokumenten-Servern. Unter der Vermittlung von Multimedia-Dokumenten wird die Bereitstellung, die Übertragung und die anschließende Präsentation der Dokumente bei einem Benutzer verstanden.

Damit ein Multimedia-Dokument auf dem Rechner des Benutzers dargestellt werden kann, wird nicht nur die Spezifikation des Dokuments benötigt, sondern auch die Erweiterungen des Präsentationssystems, die die verwendeten Sprachkonstrukte implementieren. Dafür muss im Dokumententransferformat eine Verbindung zur Implementierung vorgesehen werden.

So gesehen besteht ein Multimedia-Dokument aus der Dokumentenspezifikation, den Erweiterungen des Präsentationssystems für Manager und Medienelemente und den Mediendaten. Dabei handelt es sich aber lediglich um eine logische Einheit. Alle drei Teile eines Dokuments können an unterschiedlichen Stellen gespeichert werden. Mediendaten und Erweiterungen können auf speziellen Medien-Servern im Internet liegen. Für Optimierungen können die Erweiterungen lokal auf dem Rechner des Benutzers gespeichert werden.

Ein neues, proprietäres Dokumententransferformat zu definieren ist notwendig, da kein existierendes Format geeignet ist, um das eingeführte Meta-Dokumentenmodell und seine anwendungsspezifischen Instanzen darauf abzubilden. Würde so ein Format existieren, dann könnte dieses und das dazugehörende Präsentationssystem verwendet werden.

Bei der Konzeption eines Dokumententransferformats ist es vorteilhaft, auf existierende, offene Standards aufzusetzen. Dies kann durch die Verwendung der Extensible Markup Language (XML) [BrS98] erreicht werden. Eine Einführung wird in [Eck00] gegeben.

Damit kann das Format von anderen Anwendungen gelesen werden. Außerdem können Meta-Daten eingefügt werden, die in einer digitalen Bibliothek automatisch bei der Aufnahme in den Nachweiskatalog ausgewertet werden. Bei der Umsetzung der Konzepte und Modelle in einem

Prototyp kann auf existierende Parser zurückgegriffen werden, was die Entwicklung vereinfacht und beschleunigt.

## 5.1 Ein Dokumententransferformat

Da sich die existierenden Ansätze nicht dazu eigenen, die dynamische ad-hoc Erweiterbarkeit des erweiterbaren Ansatzes zu realisieren, wird im Folgenden ein Dokumententransferformat entwickelt, das den gestellten Anforderungen gerecht wird.

Das Dokumententransferformat dient dazu, eine Spezifikation, basierend auf dem Meta-Dokumentenmodell, in einer Datei zu speichern. Für die Speicherung wird die Auszeichnungssprache XML verwendet. Dafür muss das Meta-Dokumentenmodell, beziehungsweise Instanzen des Meta-Dokumentenmodells, auf eine XML-basierte Sprache abgebildet werden. Genauer betrachtet wird dabei die Abbildung der graph-basierten, nicht-linearen Spezifikationsstruktur auf die sequentielle, also lineare Struktur einer Datei.

Zuerst wird die Abbildung des Meta-Dokumentenmodells auf eine XML-basierte Sprache, dem Dokumententransferformat, eingeführt. Anschließend wird die Verbindung des Dokumententransferformats mit der Implementierung erläutert.

Bevor das Dokumententransferformat im Detail betrachtet wird, sei an dieser Stelle eine kurze Einführung in die für das Transferformat wichtigen Eigenschaften von XML gegeben.

### 5.1.1 Die Auszeichnungssprache XML

Unter der Bezeichnung XML hat das W3C eine Meta-Sprache zur Definition von Auszeichnungssprachen standardisiert. XML ist eine Meta-Sprache, da durch sie keine Sprache im eigentlichen Sinn definiert wird, sondern nur der grammatikalische Aufbau unterschiedlicher Auszeichnungssprachen, die auf XML basieren. Darauf aufbauend müssen immer noch die anwendungsspezifischen XML-basierten Sprachen standardisiert werden. Das Verhältnis zwischen XML und einer Auszeichnungssprache ist dasselbe wie zwischen dem Meta-Dokumentenmodell und einer Spezifikationssprache eines Anwendungsgebiets des in dieser Abhandlung beschriebenen erweiterbaren Ansatzes.

Die folgenden Abschnitte stellen eine Kurzeinführung in XML dar, wie sie für die Entwicklung des Dokumententransferformats benötigt wird. Den vollständigen Standard findet man unter [BrS98] beim W3C [W3C]. Eine ausführliche Beschreibung findet sich in [HaM01] und eine Kurzbeschreibung in [Eck00]. Die Beziehung zu SMIL, einem XML-basierten Standard für Multimedia-Dokumente, wurde bereits in Kapitel 2 erörtert.

Unter einem Element versteht man in XML einen bestimmten, ausgezeichneten Bereich in einem Dokument. Der Bereich wird durch zwei XML-Tags begrenzt, wobei die Tags selbst in spitzen Klammern stehen. Das XML-Tag, das den Anfang des Bereichs markiert, wird auch Start-Tag genannt und das XML-Tag, das das Ende markiert, das End-Tag. Alternativ dazu können sie auch öffnende und schließende Tags genannt werden.

Für den Spezialfall, dass kein Text ausgezeichnet werden soll und somit Start- und End-Tag direkt aufeinander folgen, wurde eine abkürzende Schreibweise eingeführt, das so genannte Leer-Element. Bei dieser Schreibweise wird auf das schließende Tag verzichtet. Um kenntlich zu machen, dass es sich um Leer-Element handelt, wird der schließenden eckigen Klammer ein "/" vorangestellt.

Um komplexe Dokumente aufzubauen, besteht die Möglichkeit, dass der Text innerhalb eines normalen XML-Elements weitere Elemente enthalten darf. Welche Elemente erlaubt sind, wie sie geschachtelt werden dürfen und wo normaler Text stehen darf, wird durch die Document Type Definition (DTD) festgelegt. Sie wird weiter unten behandelt.

Um die Verarbeitung der XML-Elemente zu parametrisieren, können dem Start-Tag eines XML-Elements so genannte Attribute mitgegeben werden. Ein Attribut ist ein Tupel bestehend aus Attributname und Attributwert. Attributwerte werden in doppelte Anführungszeichen geschrieben und sind typenlose Zeichenketten. Das folgende Beispiel zeigt ein leeres XML-Element `MedienElement`, dem ein Attribut mit dem Namen `src` und dem Wert "`http://www.XXX.de/LaPalma.mpg`" hinzugefügt wurde.

```
<MedienElement src="http://www.XXX.de/LaPalma.mpg" />
```

Ob Attribute im XML-Dokument angegeben werden müssen, kann für jedes einzelne Attribut bestimmt werden. Diese Festlegung wird in der DTD getroffen. Es gibt drei unterschiedliche

Klassen, die festlegen, ob Attribute angegeben werden müssen oder nicht. Die erste Klasse ist die Klasse der impliziten Attribute (`#IMPLIED`). Attribute dieser Klasse müssen nicht zwingend angegeben werden. Eine Anwendung muss entsprechend reagieren, wenn das Attribut fehlt, und einen impliziten Standardwert verwenden. Die zweite Klasse beinhaltet die notwendigen Attribute (`#REQUIRED`). Damit das Dokument gültig ist, müssen für die Attribute dieser Klasse Werte angeben werden. Die dritte Klasse beschreibt die festgelegten Attribute (`#FIXED`). Attributen dieser Klasse wird in der DTD ein fester Wert zugewiesen. Die Anwendung erhält das Attribut immer mit dem festen Wert, auch wenn dieses im Dokument nicht angegeben ist.

Neben den Attributklassen gibt es auch unterschiedliche Attributtypen. An dieser Stelle sollen nur die drei später verwendeten Typen beschrieben werden. Die vollständige Liste findet sich in [BrS98]. Der erste Typ ist der am häufigsten vorkommende Typ. Es handelt sich dabei um Zeichenkettenattribute (`#CDATA`). Der Wert dieses Attributs ist eine vom XML-Parser nicht interpretierte Zeichenkette. Der zweite Typ umfasst die Bezeichner (`#ID`). Dies bedeutet, dass der Wert des Attributs ein im Dokument eindeutiger Bezeichner ist. Ein Bezeichner muss mit einem Buchstaben beginnen und darf anschließend auch Ziffern enthalten. Der dritte Typ beinhaltet Verweise auf Bezeichner (`#IDREF`). Ein Attribut diesen Typs verweist auf ein XML-Element mit dem entsprechenden Bezeichner. Ein validierender XML-Parser überprüft die Gültigkeit von Bezeichnern und Verweisen auf Bezeichner.

Die DTD beschreibt die Grammatik einer Klasse von Dokumenten. Eine Anwendung verarbeitet demnach Dokumente, die zu einer bestimmten DTD gehören und sich basierend auf dieser DTD fehlerfrei parsen lassen. In diesem Zusammenhang wird auch von Dokumenten als Instanzen einer DTD gesprochen.

Für den auf XML basierenden Datenaustausch muss eine DTD entwickelt werden. Der Aufbau dieser DTD ist der Hauptbestandteil des XML-Standards. Für spezielle Anwendungen werden oder wurden spezielle DTDs entwickelt. Ein wichtiger Punkt ist, dass weder im XML-Standard noch in der DTD Hinweise oder Vorgehensweisen zur semantischen Interpretation der XML-Elemente oder Attribute gegeben werden. Die Festlegung der Semantik muss auf eine andere Art und Weise geschehen.

Alle XML-Elemente befinden sich in einem flachen Namensraum. Dies bedeutet, dass es jedes Element unabhängig von der Schachtelungstiefe von Elementen nur einmal gibt. Damit kann ein XML-Element in einer Anwendung nur eine Bedeutung haben. Ein weiteres Problem ist das Verwenden von unterschiedlichen XML-basierten Sprachdialekten in einem XML-Dokument. Hier kann es sehr schnell zu Problemen kommen, wenn zwei Dialekte dieselben XML-Elemente (beispielsweise `Adresse`) definiert haben, aber eine unterschiedliche Semantik damit verbinden. Beispielsweise könnte ein Dialekt unter `Adresse` eine IP-Adresse und ein anderer Dialekt die Adresse einer Person verstehen.

Um diese Probleme zu lösen, hat das W3C den XML-Namespaces Standard [BHL99] eingeführt. Es handelt sich hierbei um einen sehr einfachen Standard. Er verwendet das für Namen in XML reservierte Zeichen ':'. Die Namen von XML-Elementen und Attributen werden durch so genannte qualifizierte Namen ersetzt. Beispielweise kann ein Element `adresse` mittels qualifiziertem Namen an den Namensraum `ip` gebunden werden. Im XML-Dokument würde man dann `ip:adresse` schreiben. Der Namensraum `ip` wird mittels einer URI (Uniform Resource Identifier) [BFM98] definiert. Die einzige Bedeutung der URI aus XML-Sicht ist die eindeutige Festlegung des Namensraums; die URI wird vom XML-Parser in keiner Weise interpretiert.

### 5.1.2 Definition des Transferformats

Beim Vergleich der Eigenschaften des Meta-Dokumentenmodells mit den Anwendungsmöglichkeiten von XML fallen drei Punkte auf, die näher betrachtet werden müssen. Erstens liegt dem Meta-Dokumentenmodell keine Baumstruktur zu Grunde, was der Schachtelung von XML-Elementen entsprechen würde. Zweitens sind XML-basierte Auszeichnungssprachen nicht erweiterbar, wie es die Bezeichnung *extensible*, zu Deutsch *erweiterbar*, suggeriert. Die Erweiterbarkeit von XML bedeutet, dass jederzeit neue XML-basierte Sprachen definiert werden können. Und drittens wird durch die Verwendung von XML noch keine Aussage über die semantische Interpretation der Auszeichnungssprache gemacht. Für das Transferformat müssen diese drei Punkte, basierend auf dem XML-Standard, modelliert werden.

Die DTD eines XML-Dokuments kann nur dann ergänzt werden, wenn sie nicht im Dokument gespeichert ist. In diesem Fall können im Dokument noch Ergänzungen zu der externen DTD

definiert werden. Eine Spracherweiterung bedeutet deshalb immer auch, dass eine neue beziehungsweise erweiterte DTD bereitgestellt werden muss. Für den erweiterbaren Ansatz bedeutet dies, dass zu jedem Dokument eine individuelle DTD generiert werden muss.

Die Vorgehensweise ist in Abbildung 5-1 zu sehen. Zuerst wird durch den DTD-Generator eine für das Dokument individuelle DTD generiert. Dazu wird die Dokument-Spezifikation bezüglich verwendeter Operatoren und Medienelemente analysiert. Der DTD-Generator erzeugt eine DTD für diese Dokument-Spezifikation, mit der die Gültigkeit des Transferformats des Dokuments durch einen XML-Parser überprüft werden kann. Basierend auf dieser DTD erzeugt der XML-Generator aus der Dokumentenspezifikation das Transferformat. Dieser Vorgang ist die Abbildung der Dokument-Spezifikation auf XML.

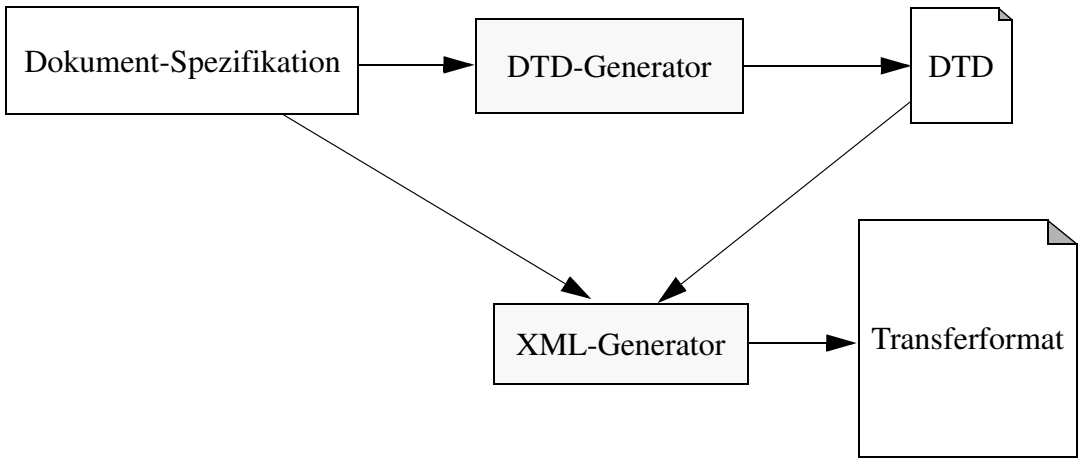


**Abb. 5-1 :** Vorgehensweise bei der Erzeugung des Transferformats

Für die Abbildung des Meta-Dokumentenmodells auf XML müssen dessen Eigenschaften genauer betrachtet werden. Im Allgemeinen sind auf dem Meta-Dokumentenmodell basierende Dokumentenspezifikationen Graphen. Werden die Verbindungen zwischen Operatoren und Medienelementen außer Betracht gelassen, so ist zu erkennen, dass die Beinhaltet-Beziehung zwischen einem Container und seinen Medienelementen und Operatoren der Schachtelung von XML-Elementen entspricht.

Im eingeführten Meta-Dokumentenmodell treten Verbindungen zwischen Operatoren und Medienelementen nur innerhalb eines Containers auf. Deshalb ist es ausreichend, innerhalb eines Containers die Graphstruktur mittels Attributen nachzubilden.

In Anhang B ist eine komplexe Dokumentenspezifikation beschrieben. Deren Abbildung auf das Transferformat ist in Anhang C abgebildet.

Medienelemente sind aus Sicht des Meta-Dokumentenmodells die einfachsten Dokumentelemente. Sie repräsentieren die Medienobjekte und den Inhalt einer Präsentation. Der komplette strukturelle Aufbau wird mit Operatoren und Containern beschrieben.

Die Abbildung eines Medienelements auf XML wird wie nachfolgend beschrieben durchgeführt. Ein Medienelement wird auf ein Leer-Element abgebildet, das den Typ des Medienelements als Elementname erhält. Beispielsweise wird ein Medienelemente vom Typ Bild (engl. image) auf das folgende XML-Element abgebildet.

```
<image />
```

Operatoren werden im Gegensatz zu Medienelementen nicht auf Leer-Elemente abgebildet. Dies liegt daran, dass die Operatoren-Tags außerdem noch zur Abbildung des Beziehungsgraphen herangezogen werden. Für einen *correct*-Operator sehen die Tags des XML-Elements wie folgt aus.

```
<correct > ... </correct>
```

Es wird ein Start-Tag generiert, das den Typ des Operators als XML-Elementname verwendet. Dazwischen steht die Modellierung des Graphen und anschließend ein End-Tag, das ebenfalls mit dem Typ des Operators als Namen besitzt. Effektoperatoren sind eine spezielle Klasse von Operatoren und werden dementsprechend identisch behandelt.

Medienelemente und Operatoren können jeweils Parameter besitzen. Diese lassen sich direkt auf XML-Attribute eines XML-Elements abbilden. Aus dem Parameternamen und dem Parameterwert lässt sich ein Tupel bilden, das aus Attributname und Attributwert besteht. Dies zeigt exemplarisch das folgende Beispiel. Die meisten Medienelemente besitzen einen Parameter, der

beschreibt, woher die Mediendaten während der Präsentation zu beziehen sind. Dieser Parameter kann URL genannt werden. In XML-Notation sieht das Beispiel folgendermaßen aus.

```
<image URL="http://www.XXX.de/images/LaGraciosa.jpg" />
```

Bei der Spezifikation von Dokumenten sind zwei Verbindungen zwischen Dokumentelementen erlaubt. Die erste Verbindung ist die Verbindung eines Operators mit einem Medienelement. Wie in der Einführung des Dokumentenmodells beschrieben, wird dabei zwischen Quell- und Zielmedien unterschieden. Die zweite Verbindung ist die Verbindung eines temporalen Operators mit einem Effektoperator.

Wie bereits erwähnt müssen diese Verbindungen über Attribute modelliert werden. Die Ziele einer Verbindung können Medienelemente und Effektoperatoren sein. Deswegen werden diese beiden Dokumentelementtypen mit einem zusätzlichen Attribut namens `ID` versehen. Dieses Attribut ist vom Typ Bezeichner (`ID`) und unterliegt somit den damit verbundenen Einschränkungen. Ebenso klar ist, dass es damit kein Parameter mit dem Namen `ID` geben kann. Das Medienelement sieht demnach folgendermaßen aus.

```
<image ID="i0" URL="http://www.XXX.de/images/LaGraciosa.jpg" />
```

Nun müssen die XML-Elemente der Operatoren um die entsprechenden Verweise ergänzt werden. Aus diesem Grund kann ein Operator-XML-Element beliebig viele der folgenden drei Elemente enthalten: `source`, `destination` und `effect`. Alle drei Elemente besitzen genau ein Attribut mit dem Namen `IDREF` vom Typ Verweis auf einen Bezeichner. Durch die Verwendung der zwei Typen, Bezeichner und Verweis auf Bezeichner, kann vom XML-Parser überprüft werden, ob die Verweise auf Bezeichner auch tatsächlich auf einen existierenden Bezeichner verweisen.

Die Abbildung eines *correct*-Operator sieht dann folgendermaßen aus. Entsprechend dem Meta-Dokumentenmodell dürfen nur `source`- und `destination`-Tags enthalten sein.

```
<correct>
  <source IDREF="i11" />
  <destination IDREF="i12" />
</correct>
```

Ein XML-Element eines Effektoperators darf ausschließlich das `effect`-XML-Element enthalten. Das folgende Beispiel zeigt einen *nextStep*-Effektoperator, der zur zeitlichen Steuerung integrierter Animationen dient.

```
<nextStep>
  <effect IDREF="i203" />
</nextStep>
```

Bei der Abbildung von Containern muss beachtet werden, dass ein Container Medienelemente und Operatoren enthalten kann. Diese Tatsache kann durch das Einschließen innerhalb des Start- und Ende-Tags des Containerelements realisiert werden. Nach der Generierung des Start-Tags wird der Containerinhalt abgebildet und anschließend das Ende-Tag hinzugefügt. Da Container eine funktionale Erweiterung von Medienelementen sind, gilt auch für sie das für Medienelemente Gesagte. Das folgende Beispiel zeigt die Abbildung eines *Frage*-Containers. Aus Platzgründen wurde dabei auf die Angabe des Containerinhalts verzichtet.

```
<question ID="i0815">
...
</question>
```

Aufgrund unabhängiger Erweiterungen durch unterschiedliche Personen kann es zur identischen Benennung von Dokumentelementen kommen. Um diesen Namenskonflikt auf XML-Ebene zu vermeiden, werden die XML-Namensräume verwendet. Zusätzlich zum Tag-Namen muss für jede Erweiterung noch ein URI angegeben werden, der als Basis für einen XML-Namensraum dient. Ein URI kann beispielsweise durch das Umdrehen einer Domänenbezeichnung erzielt werden (“`de.uni-stuttgart.informatik`”). Beim Erzeugen des XML-Formats werden diese Namensräume berücksichtigt, um Namenskonflikte zu vermeiden. Aus diesem Grund werden qualifizierte Elementnamen statt einfacher Namen generiert. Als Bezeichner für Namensräume werden alle verwendeten Namensräume eindeutig nummeriert und ein “`ns`” für *N*ame*s*pace vorne angefügt. Somit ließe sich das letzte Beispiel für den *Frage*-Container wie folgt schreiben.

```
<ns0:question ID="i0815">
...
</ns0:question>
```

Am Anfang des Dokuments muss der XML-Namensraum `ns0` durch das `xmlns`-Attribut definiert werden. Dazu wird das komplette Dokument in das XML-Element `mava` eingefügt. Das Element mit der Namensraum-Definition sieht wie folgt aus.

```
<mava xmlns:ns0="de.uni-stuttgart.informatik" >
...
</mava>
```

In Anhang C befindet sich ein kompletter Abdruck eines komplexeren Dokuments in dem beschriebenen Transferformat.

Verwendet man das in diesem Kapitel angegebene Verfahren, um die Namen von XML-Elementen zu bestimmen, kann dies dazu führen, dass die Dateien größer werden als erwartet. Die tatsächliche Dateigröße von Dokumenten oder Datenobjekten, die auf Basis von XML gespeichert werden, ist ein bekanntes Problem. Laut [LiS99] ist die XML-Datei um den Faktor 1,5 bis 3 größer als bei einem vergleichbaren binären Speicherformat. Dies liegt an den aussagekräftigen Namen für Tags und Attribute und deren Wiederholung. Mittels Kompression von XML-Dateien wie in [GiS00, LiS99] vorgeschlagen, lässt sich der Nachteil der Dateigröße aber beheben.

### 5.1.3 Aufbau des Objektmodells

Wie eingangs bereits erläutert, kann mit XML nur die Struktur von Dokumenten oder Datenobjekten beschrieben werden, nicht aber deren semantische Bedeutung. Für die Verwendung von XML heißt dies, dass für die Darstellung oder Verarbeitung eines XML-Dokuments eine Anwendung benötigt wird, die die Bedeutung der einzelnen XML-Elemente kennt. In diesem Zusammenhang bedeutet die Erweiterbarkeit des Ansatzes, dass das Präsentationssystem gegebenenfalls erweitert werden muss, falls das Dokument XML-Elemente enthält, die es nicht kennt. Damit muss ein Zusammenhang zwischen der XML-basierten Dokumentenstruktur und den benötigten Erweiterungen für das Präsentationssystem hergestellt werden.

Zur Speicherung von Datenobjekten werden im Java-Umfeld von den Firmen Sun Microsystems und IBM unabhängig voneinander jeweils ein Mechanismus zur Abbildung einer JavaBeans-Struktur auf XML-Datenobjekte vorgeschlagen [Sun01b, WeD99]. Die Idee der beiden Ansätze ist es, Datenobjekte und die Methodenaufrufe zur Verbindung der Programmiersprachenobjekte im Speicher zu beschreiben. Dies ist ein sehr generischer Ansatz, der auf beliebige Datenstrukturen anwendbar ist, was zu einer komplexen Struktur des XML-Datenobjekts führt. Des weiteren gehen beide Ansätze davon aus, dass bei beiden Abbildungsschritten, also von der Objektstruktur im Speicher auf XML-Datenobjekte und umgekehrt, die Java-Klassen vorhanden sind.

Eine sehr enge Kopplung zwischen dem Format und der Implementierung, wie sie in den nächsten Abschnitten vorgestellt wird, erlaubt einfache Dokumentenstrukturen und beschränkt den Aufwand und die Speichergröße eines Dokuments im Vergleich zur allgemeinen Abbildung von JavaBeans auf XML-Datenobjekte. Des Weiteren wird die Voraussetzung aufgehoben, dass die Java-Klassen vorhanden sein müssen, da das zu entwickelnde System ad-hoc erweiterbar sein soll. Um dies mit den beiden vorher geschilderten Ansätzen zu ermöglichen, müssten diese erweitert werden.

Ziel der Arbeit ist es, ein Dokumentensystem zu konzipieren, das den Autor durch anwendungsspezifische Spezifikationssprachen, basierend auf dem Meta-Dokumentenmodell, beim Erstellen einer Multimedia-Präsentation unterstützt. Anwendungsspezifische Sprachen werden, wie in den vorangegangenen Abschnitten erläutert, auf die jeweiligen XML-Elemente abgebildet. Diese XML-Elemente müssen mit Klassen der Implementierung verbunden werden. Um diesen Schritt vorzubereiten, muss dem System zunächst mitgeteilt werden, welche Klassen benötigt werden. Genau dies ist Aufgabe der so genannten Sektorinformation.

In Abhängigkeit von der verwendeten Spezifikationssprache, beziehungsweise vom Anwendungsgebiet, wird dem Dokument eine Beschreibung der benötigten Programmerweiterungen des Präsentationssystems mitgegeben. Dies erfolgt durch ein spezielles XML-Element.

Bei diesem XML-Element handelt es sich um das `requiredExtension`-Element. Es teilt dem System mit, dass eine bestimmte Erweiterung benötigt wird. Welche Erweiterung benötigt wird, wird durch Attribute des XML-Elements beschrieben. Es existieren zwei unterschiedliche

Arten von Erweiterungen: Operatoren und Medienelemente. Im ersten Fall enthält das Programmcode-Paket die Manager-Implementierung und die notwendigen Klassen für seine Operatoren. Für die zweite Art von Erweiterungen enthält das Programmcode-Paket Klassen zur Implementierung eines Medienelements.

```
<requiredExtension name="LocationManager"
          package="location.jar"
          ns="de.uni-stuttgart.informatik.vs.location">
```

Das XML-Element benennt also Programmcode-Pakete, die für eine Präsentation des Dokuments vorhanden sein und dementsprechend in das Präsentationssystem integriert werden müssen. Andernfalls kann das Dokument nicht präsentiert werden. Wie das Präsentationssystem zu den Erweiterungen kommt, wird in Kapitel 5.2 erläutert.

Nachdem alle Programmcode-Pakete geladen wurden, kann das Objektmodell des Dokuments erzeugt werden. Im Objektmodell werden die Dokumentelemente durch Objekte in der Programmiersprache repräsentiert. Das heißt, jedes Dokumentelement ist eindeutig mit einer Klasse der Programmiersprache verbunden. Dies wird über ein Attribut der XML-Elemente eines Dokumentelements realisiert. An dieser Stelle bietet es sich an, ein festgelegtes Attribut zu verwenden, da dieses für alle Instanzen des Dokumentelements den gleichen Wert besitzt und somit nur einmal in der DTD angegeben werden muss. Folgende Zeile aus einer Dokument-DTD beschreibt dann das gewünschte Verhalten.

```
<!ATTLIST image class #FIXED "mavax.baseelements.Image">
```

Das Attribut `class` eines XML-Elements eines Dokumentelements legt die Klasse fest, die das Dokumentelement im Objektmodell repräsentieren soll. Sofern die Klasse nicht zum Grundsystem gehört, gibt es ein entsprechendes `requiredExtension`-Element zu Beginn des Dokuments. Das dadurch bestimmte Programmcode-Paket enthält die benötigten Klassen. Der Attribut-Typ `#FIXED` besagt, dass automatisch jedes verwendete `image`-Tag das Attribut `class` besitzt, auch wenn dies nicht angegeben ist.

Attribute eines Dokumentelements (bspw. die URL der Mediendaten) werden beim Einlesen eines Dokuments in dem zugehörigen Objekt des Objektmodells gesetzt. Für ein Attribut muss

in der Klasse eine Methode nach dem Schema `set<AttributName>` vorhanden sein. Diese wird unter Verwendung von Java-Reflection [Sun98] dynamisch aufgerufen.

Nachdem die einzelnen Dokumentelemente den Klassen des Objektmodells zugeordnet wurden, muss abschließend die Dokumentenstruktur in Form von Beziehungen zwischen den Objekten des Objektmodells aufgebaut werden. Diese Struktur besteht aus zwei Teilen. Der erste Teil ist die Enthaltenseins-Beziehung zwischen Dokumentelementen und Containern. Der zweite Teil umfasst die Beziehungen zwischen Operatoren und Medienelementen.

Die Enthaltenseins-Beziehung lässt sich anhand der Schachtelung der XML-Elemente rekonstruieren. Die in einem Container-Element enthaltenen Dokumentelemente müssen dem Container hinzugefügt werden. Dazu muss der Container die entsprechenden Methodenaufrufe (`addMediaItem()` und `addOperator()`) bereitstellen. Programmtechnisch geschieht dies durch die Bereitstellung einer Schnittstellendefinition für Container, die diese Funktionalität vorschreibt und die von jeder Klasse implementiert werden muss, die einen Container im Objektmodell repräsentieren soll.

Die Beziehungsstruktur zwischen Operatoren und Medienelementen wird, wie in Kapitel 5.1.2 beschrieben, durch drei XML-Elemente (`source`, `destination` und `effect`) dargestellt. Für die Erzeugung des Objektmodells wird an dieser Stelle nur die Umkehrfunktion benötigt. Enthält ein Operator-Element beispielsweise ein `destination`-Tag, so muss dem Operatorobjekt ein Verweis auf das durch das `ID`-Attribut bestimmte Medienelement hinzugefügt werden. Das gleiche gilt für das `source`- und `effect`-Tag. Dass einem Operator Quell- und Zieldokumentelemente und einem Effektoperator Effektmedienelemente hinzugefügt werden können, wird durch zwei Schnittstellendefinitionen für Operatoren und Effektoperatoren abgebildet. Dementsprechend müssen Klassen, die im Objektmodell einen Operator oder Effektoperator repräsentieren, die entsprechenden Schnittstellen implementieren.

Um ein Dokument in sequentieller Reihenfolge parsen zu können, ist die korrekte Reihenfolge der XML-Elemente bei der Abbildung eines Containers wichtig. Dies reduziert den Aufwand erheblich, da beim Parsen kein Ableitungsbaum erzeugt werden muss. Beim Aufbau des Objektmodells eines Containers ist sicherzustellen, dass alle Dokumentelemente, auf die durch einen Operator verwiesen werden kann, bereits verarbeitet wurden. Dann kann das `IDREF`-

Attribut auf ein bereits im Speicher liegendes Objekt aufgelöst werden und somit die Objektstruktur direkt aufgebaut werden.

Bei der Abbildung eines Containers müssen als erstes alle diejenigen Dokumentelemente abgebildet werden, auf die durch einen Operator verwiesen werden kann. Das sind die Container und die Medienelemente. Aus Sicht eines temporalen Operators ist ein Effektoperator ein Zielelement und besitzt deshalb einen Bezeichner, um von einem temporalen Operator referenziert werden zu können. Auf Grund dieser Abhängigkeit müssen die Effektoperatoren vor den restlichen Operatoren abgebildet werden. Erst nach Medienelementen und Effektoperatoren dürfen die Operatoren abgebildet werden. Damit sind alle Dokumentelemente, die von Operatoren referenziert werden können, bekannt und die entsprechenden Objekte im Objektmodell bereits erzeugt.

Das Transferformat des erweiterbaren Ansatzes beschreibt neben der Dokumentenstruktur auch die Konfiguration des Präsentationssystems. Unter der Konfiguration des Präsentationssystems wird die Auswahl und Integration einzelner Erweiterungen in das Präsentationssystem verstanden. Die tatsächliche Konfiguration des Präsentationssystems hängt dabei vom momentan präsentierten Dokument ab.

## 5.2 Speicherung von Dokumenten auf Internet-Servern

Bei dem vorgestellten Verfahren ist es nicht mehr ausreichend, ein Dokument als eine Einheit, die auch in einer Datei gespeichert ist, bereitzustellen. Ansätze, die eine Skriptsprache verwenden, speichern die Skripte in der Dokumentdatei. Ansätze, die keine Funktionalitätserweiterung erlauben, bestehen ebenfalls aus einer Einheit, die auf der vorgegebenen Spezifikationssprache basiert.

In diesem Abschnitt soll eine flexiblere Vorgehensweise konzipiert werden, die auf der Trennung der Dokumentstruktur und der Implementierung in zusätzlichen Programmcode-Paketen basiert. Ein mehrstufiges “Suchen” nach den Programmcode-Paketen erlaubt Optimierungen, wie zum Beispiel das Caching der Programmcode-Pakete im lokalen Dateisystem.

### 5.2.1 Logische Sicht auf Dokumente

Ein Multimedia-Dokument besteht nicht nur aus der Strukturbeschreibung, der Spezifikation und den Mediendaten, vielmehr werden auch die dazugehörigen Programmcode-Pakete als Teil des Dokuments angesehen. Ohne diese Programmcode-Pakete kann das Dokument nicht präsentiert werden. Diese Beziehung zwischen der Dokumentenspezifikation und dem implementierenden Programmcode ist als eine logische Dokumenteneinheit in Abbildung 5-2 graphisch dargestellt. Das Dokument umfasst neben der Spezifikation auch den Manager für computergestütztes Training und das Kontrollkästchen-Medienelement.

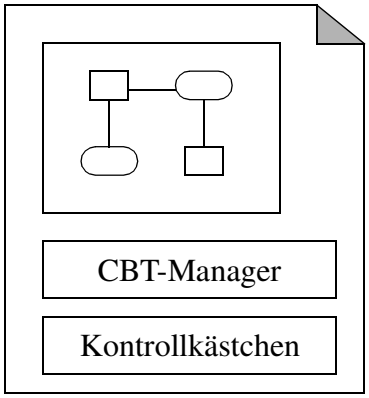


**Abb. 5-2 :** Logische Sicht auf ein Dokument

Um mehr Flexibilität beim Laden des Dokuments und den Erweiterungen zu erhalten, werden die Dokumentenstruktur und die Erweiterungen getrennt gespeichert. Wie in Abbildung 5-3 dargestellt und in der Beschreibung der Sektorinformation bereits vorgesehen, werden in einem Dokument nur Referenzen auf die benötigten Erweiterungen gespeichert. Die grauen Pfeile stellen die Referenzen dar. Im Beispiel sieht man, dass das Dokument zwei Referenzen auf die Programmcode-Pakete des CBT-Managers und des Kontrollkästchens besitzt.

Durch die Verwendung von Referenzen können nun die Erweiterungen auf unterschiedlichen Servern gespeichert werden. Ein weiterer Vorteil ist außerdem, dass die Erweiterungen nur einmal auf dem Server gespeichert werden müssen, auch wenn Sie in mehreren Dokumenten verwendet werden. Dies ist bei Skripten, die Teil eines Dokuments sind, nicht möglich.

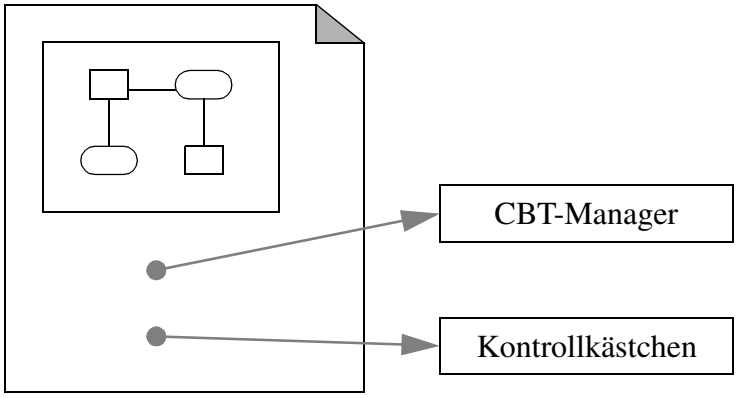


**Abb. 5-3 :** Speicherung eines Dokuments auf verschiedenen Servern

### 5.2.2 Der Dokumentenladevorgang

Beim Ladevorgang wird zuerst das Dokument von einem Internet-Server geladen. Der Server und das Protokoll, mit dem das Dokument geladen werden soll, werden durch eine URL bestimmt. Dokumente können durch die Verwendung des HTTP-Protokolls [Http99] von Internet-Servern und durch das FILE-Protokoll aus dem lokalen Dateisystem geladen werden.

Anschließend werden bei der Analyse der Sektorinformation die `requiredExtension`-Elemente ausgewertet. Diese Verfügen über ein URL-Attribut, das auf das zu ladende Programmcode-Paket verweist. Der Klassenmanager des Systems wird nun das Programmcode-Paket von der entsprechenden URL laden und die darin enthaltenen Klassen dem Präsentationssystem zur Verfügung stellen.

Danach wird die Dokumentenstruktur analysiert, daraus das Objektmodell des Dokuments wie in Abschnitt 5.1.3 erzeugt und die Kontrolle der Präsentation durch die Übergabe der Zuständigkeit an die Manager übernommen.

Um das URL-Attribut der `requiredExtension`-Elemente flexibler verwenden zu können, wird das Verfahren des Programmcode-Paket-Ladens erweitert. Ein Programmpaket wird zuerst unter der URL des Dokuments gesucht. Befindet sich ein Programmcode-Paket unter der gleichen URL wie das Dokument, wird es von dieser Stelle geladen. Die URL wird dabei nicht interpretiert. Damit kann die URL auf einen bestimmten Server verweisen, der sicherstellen kann, dass die Programmcode-Pakete im Internet verfügbar sind. Somit kann das URL-Attribut

vom Ersteller der Erweiterung vorgeben werden, und der Autor muss sich um dieses Attribut nicht kümmern.

Um häufiger verwendete Programmcode-Pakete nicht jedes Mal neu aus dem Internet zu laden, bietet es sich an, Programmcode-Pakete lokal auf der Festplatte zu speichern. Dazu wird der Ladevorgang um die Überprüfung, ob eine Erweiterung bereits lokal vorhanden ist, ergänzt. Erweiterungen werden anschließend nur aus dem Internet geladen, wenn sie lokal nicht vorhanden sind. Neben dem Laden der Erweiterung muss in diesem Fall noch eine Kopie in den lokalen Cache auf die Festplatte geschrieben werden.

### 5.2.3 Dynamisches Nachladen von Erweiterungen

Um die in dieser Abhandlung beschriebene dynamische ad-hoc Erweiterbarkeit zu realisieren, wird ein Klassenmanager für das Präsentationssystem benötigt, der die folgende Funktionalität bereit stellt: die Auflösung von Abhängigkeiten zwischen Programmcode-Paketen, das Laden von Klassen aus den Programmcode-Paketen, das Caching der Programmcode-Pakete als Optimierung des Ladevorgangs und die Versionsverwaltung der Erweiterungen.

Standardmäßig werden in Java Klassen aus dem aktuellen Klassenpfad durch den Systemklassenlader geladen. Klassen können in so genannten JAR-Datein (*J*ava *Ar*chive) zusammengefasst werden. Der Klassenpfad ist eine Aufzählung von JAR-Dateien und Verzeichnissen im lokalen Dateisystem, in denen der Systemklassenlader der Java-Laufzeitumgebung nach benötigten Klassen sucht. Der Klassenpfad wird immer vor der Ausführung einer Anwendung festgelegt.

Seit der Einführung von Java 2 bietet die Java-Laufzeitumgebung die Möglichkeit, Klassen dynamisch zur Laufzeit aus einer JAR-Datei oder einem Datei-Verzeichnis nachzuladen und Objekte dieser Klasse zu erzeugen. Dazu wird ein so genannter Klassenlader verwendet. Die entsprechende Java-Klasse heißt `ClassLoader`. Um Java-Klassen dynamisch zu laden, muss zuerst ein `ClassLoader` instanziiert werden. Dazu wird dem Konstruktor der Klasse `ClassLoader` eine Liste von URLs als Parameter übergeben, unter denen später nach Klassen gesucht wird. Um eine Klasse zu verwenden, muss zuvor durch die Methode `loadClass(KlassenName)` eine entsprechende Klassenbeschreibung geladen werden.

Anschließend kann, basierend auf der Klassenbeschreibung, ein neues Objekt erzeugt werden. Der von der Java 2 Plattform zur Verfügung gestellte Klassenlademechanismus erfüllt die oben gestellten Anforderungen noch nicht, weshalb der Klassenmanager eingeführt wird.

Der Ablauf des Dokumentenladevorgangs ist schematisch in Abbildung 5-4 dargestellt. Während des Ladevorgangs initialisiert und konfiguriert der Dokumentenlader den Klassenmanager. Dazu analysiert er die im Dokument enthaltenen `requiredExtension`-Tags und teilt dem Klassenmanager mit, die mit den Tags assoziierten JAR-Datei in die Suchliste aufzunehmen. Die JAR-Dateien enthalten zusätzlich die Information, von welchen weiteren Programmcode-Paketen sie abhängen und durch welche weiteren JARs diese repräsentiert werden. Der Klassenmanager löst diese Abhängigkeiten auf und fügt alle weiteren benötigten JAR-Dateien ebenfalls der Suchliste hinzu.

Um die Klassen einer Erweiterung zu verwenden, wird durch den Aufruf der Methode `KlassenManager.classForName(Klassenname)` das Klassenobjekt der Klasse mit dem Name `Klassenname` angefordert. Der Aufrufende muss dabei nicht mehr wissen, in welcher JAR-Datei die Klasse enthalten ist.

Durch lokales Caching reduziert der Klassenmanager die Ladezeiten. Falls die JAR-Datei auf einem Server im Internet liegt und zum ersten Mal verwendet wird, kopiert der Klassenmanager zuerst die JAR-Datei in ein spezielles Verzeichnis im lokalen Dateisystem. Anschließend und bei allen weiteren Verwendungen wird die lokale JAR-Datei verwendet, und nicht die entfernte aus dem Internet.

In [Fre01] wird die Implementierung einer Erweiterung des Java-Klassenladers beschrieben, die Erweiterungen unter der Dokumenten-URL sucht und ein Caching auf der lokalen Festplatte durchführt. Des weiteren wurde der Klassenmanager zur Verwaltung von Versionen erweitert, die ein nachträgliches Verbessern von Programmcode-Paketen im Fehlerfall oder eine Erweiterung der enthaltenen Funktionalität erlauben. Durch die Versionsverwaltung wird sichergestellt, dass bei der Präsentation eines Dokuments auch immer die dazugehörige Version der Erweiterung verwendet wird. Dadurch wird es möglich, mit dem gleichen Präsentationssystem Dokumente zu präsentieren, die unterschiedliche Versionen eines Managers oder Medienelements verwenden.

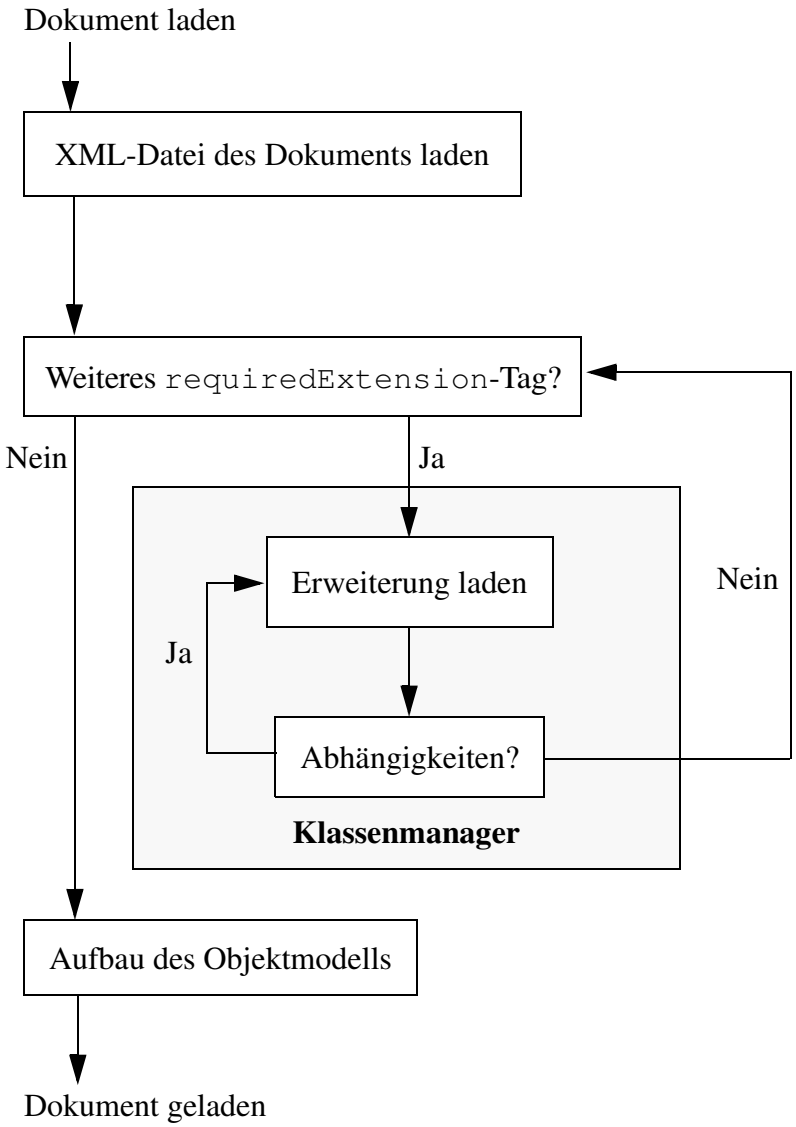


**Abb. 5-4 :** Ablauf des Dokumentenladevorgangs

## 5.3 Zusammenfassung

Das in diesem Kapitel vorgestellte Dokumententransferformat erlaubt die Speicherung von Dokumentspezifikationen. Es basiert auf dem offenen Standard XML des W3C und ermöglicht damit die Weiterverarbeitung durch andere Anwendungen, wie z.B. das Einspielen von Meta-Daten in ein Nachweissystem einer digitalen Bibliothek.

Im Allgemeinen sind Dokumentspezifikationen Graphen. Wird ausschließlich die Beinhaltet-Beziehung betrachtet, die durch das Container-Element eingeführt wurde, so wird eine Baum-

struktur sichtbar. Diese Baumstruktur kann direkt auf die verschachtelten XML-Elemente, die ebenfalls eine Baumstruktur bilden, abgebildet werden. Die Graphstruktur innerhalb eines Containers wird durch Attribute vom Typ `ID` und `IDREF` nachgebildet und wird beim Laden einer Dokumentenspezifikation anhand der Attribute aufgebaut.

Die Verknüpfung der Spezifikationssprache und der dazugehörenden Implementierung wird ebenfalls durch die Verwendung von Attributen erreicht. Dabei ist es ausreichend, den Verweis auf eine Klasse in einer Programmiersprache als unveränderliches Attribut in der DTD anzugeben. Über die Sektorinformation bekommt das Präsentationssystem mitgeteilt, welche Programmcode-Pakete für das Abspielen eines Dokuments benötigt werden. Der Klassenmanager übernimmt das Laden der Programmcode-Pakete und führt ein lokales Caching der Erweiterungen durch.

# 6 ARCHITEKTUR EINES ERWEITERBAREN AUTORENWERKZEUGS

Das Anwendungsprogramm für die Erstellung von Multimedia-Präsentationen, basierend auf Multimedia-Dokumenten, wird Autorenwerkzeug genannt [EHV93]. Das Autorenwerkzeug visualisiert die Spezifikation eines Multimedia-Dokuments und stellt eine Benutzungsschnittstelle zur Bearbeitung der Spezifikation bereit.

In diesem Kapitel wird die Konzeption eines Autorenwerkzeugs für den erweiterbaren Ansatz beschrieben. Dabei werden die Visualisierung des Dokumentenmodells, die notwendigen Interaktionsschritte zur Modifikation einer Spezifikation und Konzepte zur Vereinfachung des Umgangs mit der Spezifikation betrachtet. Zur Darstellung der Konzepte wird auf Bildschirmabzüge des im MAVA-Projekt entwickelten Prototypen des MAVA-Autorenwerkzeugs (genannt MEd, als Kurzform für *M*AVA-*Ed*itor) zurückgegriffen.

Nach der Konzeption wird die Architektur eines erweiterbaren Autorenwerkzeugs vorgestellt. Dabei wird die Vorgehensweise bei der Umsetzung der Erweiterbarkeit genauer dargestellt.

## 6.1 Grundfunktionalität des Autorenwerkzeugs

Die Grundfunktionalität des Autorenwerkzeugs umfasst die Visualisierung des Dokumentenmodells und die Festlegung der Interaktionsschritte, die ein Autor während der Dokumentenerstellung benötigt. In diesem Abschnitt werden schrittweise die graphische Visualisierung und die Interaktionsschritte vorgestellt.

### 6.1.1 Graphische Visualisierung des Dokumentenmodells

Für die Visualisierung des Dokumentenmodells kann die in Kapitel 3 eingeführte graphische Darstellung von Dokumentelementen direkt umgesetzt werden. Sie steht dem Autor als Icon-Ansicht zur Verfügung.

Aus den Knoten des Spezifikationsgraphen werden Icons, die auf einer Fläche dargestellt werden. In Abbildung 6-1 sind die Icons für die vier Elemente des Meta-Dokumentenmodells, nämlich das Medienelement, der Container, der Effekt-Operator und der Operator dargestellt.


**Abb. 6-1 :** Darstellung der Dokumentelemente

Im Folgenden wird die Darstellung der Dokumentelemente aus Abbildung 6-1 von links nach rechts besprochen. Damit sich ein Autor in einer Spezifikation besser zurecht findet, können Medienelementen und Containern aussagekräftige Namen gegeben werden. So hat das Medienelement den Namen “Lied” und der Container “1. Frage”. Neben dem Namen wird der Typ des Elements in eckigen Klammern angegeben. Das erste Element ist also ein Audiomedienelement. Der Container ist vom Typ “Question”, also ein Container, der die Spezifikation einer Frage enthält. Das dritte Dokumentelement ist ein Effekt-Operator, das vierte ein normaler Operator. Bei Operatoren wird lediglich der Typ des Operators unterhalb des Icons angegeben. So werden in Abbildung 6-1 ein *stepForward*-Effektoperator und ein *correct*-Operator gezeigt. Die Bedeutung der farbigen Punkte neben beziehungsweise über den Operatoren werden im Abschnitt 6.1.2 erläutert.

Neben dem Namen und dem Typ können alle Dokumentelemente Parameter besitzen. In Abbildung 6-2 ist der Parameterdialog eines Bild-Medienelements (engl. image) zu sehen. In diesem Dialog können der Name des Bildes, seine Breite, seine Höhe, die URL, die auf die Mediendaten verweist und eine Präsentationsdauer des Bildes festgelegt werden. Dieser Dialog kann durch das Kontextmenü des Dokumentelement-Icons aufgerufen werden. Um Dateinamen zu bestimmen, kann ein Dateiauswahldialog geöffnet werden (“Choose File...”-Schaltfläche in Abbildung 6-2); der gewählte Dateiname wird anschließend automatisch in das URL-Feld übernommen. Weitere Unterstützung bietet der so genannte Medieninspektor. Nach einem Klick auf die “Run Inspector”-Schaltfläche, wird die Mediendatei gelesen und alle daraus bestimmbaren Parameter automatisch gesetzt. Im Beispiel des Bildes kann damit die Breite und Höhe ohne

großen Aufwand bestimmt werden. Bei kontinuierlichen Medienelementen, wie Audios und Videos, kommt die Präsentationsdauer hinzu.

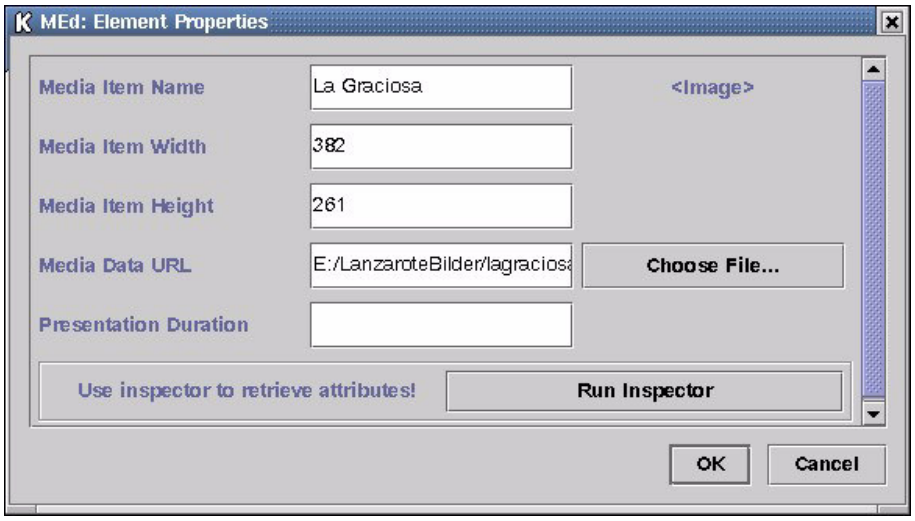


**Abb. 6-2 :** Festlegung von Parametern von Medienelementen

### 6.1.2 Interaktion während des Spezifikationsprozesses

Für die Spezifikation eines auf dem erweiterbaren Ansatz basierenden Multimedia-Dokuments gibt es drei Klassen von notwendigen Interaktionen durch den Autor. Es handelt sich dabei um das Hinzufügen von Dokumentelementen, das Festlegen von Parametern der Dokumentelemente und das Verbinden von Operatoren mit Medienelementen. Alle weiteren Interaktionsmöglichkeiten dienen lediglich der Vereinfachung des Spezifikationsprozesses reduzieren damit die Anzahl der ursprünglich benötigten Interaktionsschritte.

Bei der Spezifikation eines Dokuments soll so viel wie möglich graphisch interaktiv mit der Maus als Eingabegerät erledigt werden. Nur in Fällen, in denen es nicht anders möglich ist, soll auf die Tastatur zurückgegriffen werden (beispielsweise zur Eingabe des Namens eines Medienelements).

Dokumentelemente können einem Dokument per Drag&Drop hinzugefügt werden. Dazu stellt das Autorenwerkzeug zwei Auswahlfenster zur Verfügung: Ein Fenster, um Medien- oder Containerelemente und eines, um Operatoren auszuwählen. In den Auswahlfenstern wird ein Doku-

mentelement selektiert und bei gedrückter Maustaste auf ein Fenster der Icon-Ansicht gezogen (drag). An der gewünschten Stelle wird die Maustaste losgelassen (drop) und damit das Dokumentelement dem Dokument hinzugefügt.

Die drei möglichen Arten von Verbindungen werden in drei unterschiedlichen Farben dargestellt. Die Quellmedienelement-Verbindung wird durch eine rote Verbindungslinie zwischen einem Operator und dem entsprechenden Medienelement dargestellt. Die Zielmedienelement-Verbindung wird durch eine blaue Verbindungslinie dargestellt (Abbildung 6-3).

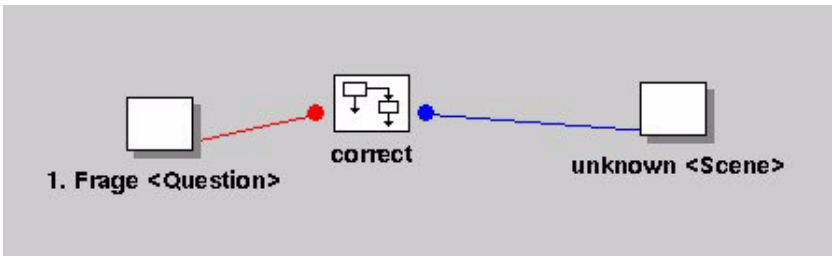


**Abb. 6-3 :** Verbindung eines Operators mit Medienelementen

Wie in Kapitel 3 beschrieben hat ein Effektoperator keine Quell- und Zielmedienelemente. Er besitzt stattdessen Effektmedienelemente. Das sind diejenigen Medienelemente, auf die der Effekt angewendet werden soll. Verbindungen mit Effektmedien werden durch eine grüne Verbindungslinie dargestellt (Abbildung 6-4).

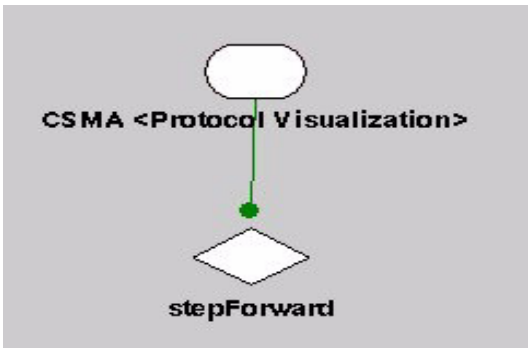


**Abb. 6-4 :** Verbindung eines Effektoperators mit einem Medienelement

Die runden Punkte links, rechts oder über dem Operator stellen den Ausgangspunkt einer Verbindung dar. Um einen Operator mit einem Medienelement zu verbinden, drückt der Autor über dem entsprechenden Punkt die linke Maustaste und hält sie gedrückt. Dann bewegt er den Mauszeiger über das Icon des Medienelements, das er mit dem Operator verbinden möchte. Befindet sich der Mauszeiger über dem Icon des Medienelements, kann der Autor die Maustaste loslassen. Durch eine grafische Hervorhebung des Medien-Icons erhält der Autor eine zusätzliche Rückmeldung, ob das unter der Maus liegende Medienelement ausgewählt ist. Damit ist der Operator mit dem Medienelement verbunden, und die Verbindungslinie bleibt dauerhaft bestehen.

Aufgrund ihres Typs können nicht alle Operatoren mit allen Medienelementen verbunden werden (vergleiche Kapitel 4.2). So hat es beispielsweise keinen Sinn, einem Audiomedienelement eine Position in der Präsentationsfläche zuzuweisen oder für einen Container, der nicht vom Typ Frage ist, die Antwort auszuwerten und zu bestimmen, womit die Präsentation fortfahren soll. In diesen Fällen wird keine Verbindung zwischen dem Operator und den Medienelementen akzeptiert. Anhand der unterstützen Schnittstellen wird vom Autorenwerkzeug die Überprüfung der syntaktischen Konsistenz durchgeführt. Dies verhindert, dass der Benutzer sinnlose Spezifikationen erstellt.

In Anhang B sind Bildschirmabzüge abgebildet, die während der Spezifikation eines Reiseführerdokuments erstellt wurden. Sie verdeutlichen die Vorgehensweise und Verwendung der Icon-Ansicht beim Editieren eines Dokuments.

## 6.2 Autorenunterstützung für die Spezifikation

Die im vorherigen Abschnitt eingeführten Interaktionsschritte sind sehr einfacher Natur. Außerdem handelt es sich nur um sehr wenige unterschiedliche Interaktionsschritte. Nichtsdestotrotz kann der Spezifikationsprozess für den Autor noch weiter vereinfacht werden, z.B. dadurch, dass die Anzahl der Interaktionsschritte reduziert oder bei der Verwaltung des Dokuments durch den Einsatz entsprechender Werkzeuge geholfen wird.

### 6.2.1 Navigationsunterstützung durch Visualisierung der Baumstruktur

Um auch in komplexen Dokumenten die Übersicht über die Spezifikation zu behalten, wird der Inhalt jeden Containers in einem eigenen Fenster dargestellt. Das Fenster für einen Container wird durch einen Doppelklick auf das Icon des Containers geöffnet. Über die Anwendungsmenüs und die Werkzeugleiste kann das Fenster des Wurzelcontainers eines Dokuments geöffnet werden.

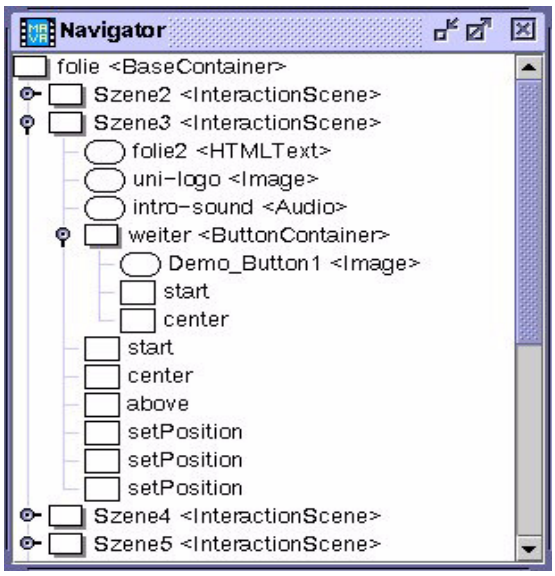


**Abb. 6-5 :** Die Baumansicht eines Dokuments

Dies führt dazu, dass in größeren Dokumenten viele Fenster geöffnet sind und dass eventuell viele Eltern-Container geöffnet werden müssen, bevor das gewünschte Container-Fenster geöffnet werden kann. Um diese beiden Fälle für den Autor zu vereinfachen, stellt das Autorenwerkzeug eine Baumansicht zur Verfügung. In dieser Darstellung können wie in einem Datei-Browser einzelne Container, die den Verzeichnissen entsprechen, auf- und zugeklappt werden. Die in Abbildung 6-5 gezeigte Baumansicht stellt die Beinhaltet-Beziehung der Dokumentelemente und Container graphisch dar. Die Graphstruktur wird in ihm nicht angezeigt. Die einzelnen Einträge der Baumansicht stellen den Namen und den Typ des Dokumentelements dar. Der Typ wird außerdem zusätzlich über ein Icon dargestellt. Wird ein Container selektiert, wird gleichzeitig das zugehörige Fenster geöffnet oder, falls es schon geöffnet ist, in den Vordergrund

geholt. Werden ein Medienelement oder ein Operator ausgewählt und der zugehörige Container ist bereits geöffnet, wird das Dokumentelement im Container-Fenster selektiert.

### 6.2.2 Filterung der Visualisierung von Dokumenten

Mit jedem Konzept wird ein anderer Aspekt einer Multimedia-Präsentation spezifiziert. So kann beispielsweise in einem Dokument der räumliche Aspekt, der zeitliche Aspekt oder der Verlauf eines computergestützten Trainings spezifiziert sein. Jeder dieser Aspekte wird mit einer unterschiedlichen Menge von Operatoren ausgedrückt.

Im Container-Fenster werden alle Medienelemente und Operatoren des entsprechenden Containers dargestellt. Damit werden die Operatoren aller verwendeten Aspekte in einem Container gleichzeitig angezeigt. In Abbildung 6-6 ist ein Container-Fenster dargestellt, in dem Operatoren für den zeitlichen und den räumlichen Aspekt enthalten sind.

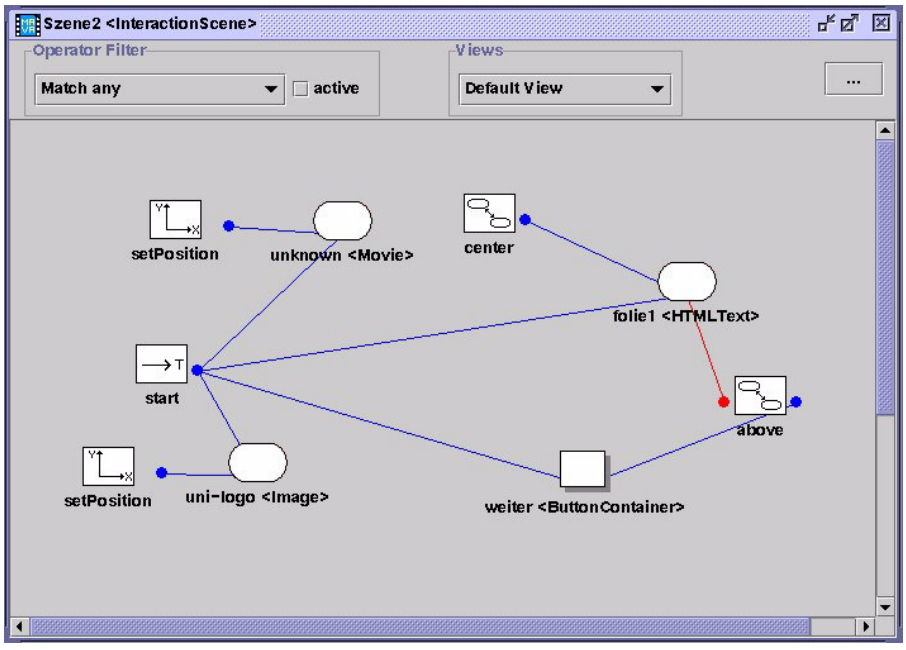


**Abb. 6-6 :** Darstellung eines Containers ohne Filterung

Dies kann dazu führen, dass der Autor leicht die Übersicht verliert. An dieser Stelle lässt sich die Tatsache ausnutzen, dass die Operatoren unterschiedlicher Konzepte unabhängig voneinan-

der verwendet werden. Das heißt, wenn der Autor einen Aspekt des Dokument editiert, ist er nicht an den Operatoren anderer Aspekte interessiert.

An der Benutzungsschnittstelle können Operatoren, die nicht zu einem Aspekt gehören, ausgeblendet werden. Für den bereits betrachteten Container aus Abbildung 6-6 ist die Aktivierung eines Filters, der nur räumliche Operatoren anzeigt, in Abbildung 6-7 gezeigt. Links oben im Container-Fenster ist zu sehen, dass der räumliche (engl. spatial) Filter aktiviert ist. Der Autor sieht nur die räumlichen Operatoren und kann sich somit auf die Spezifikation des räumlichen Layouts des Dokuments konzentrieren. Möchte er das temporale Layout editieren, wählt er dies aus der Operatoren-Filter-Liste aus, und es werden nur noch die temporalen Operatoren angezeigt.

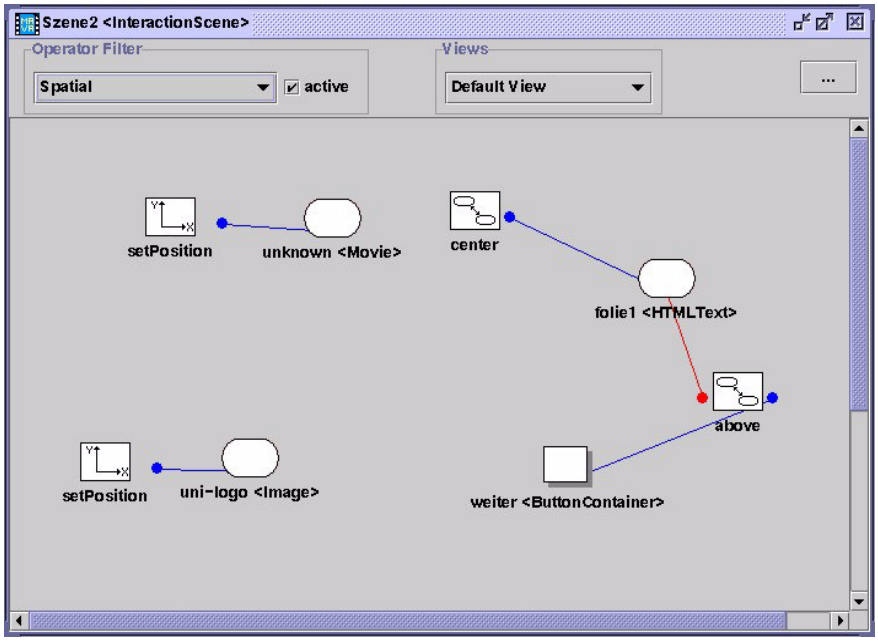


**Abb. 6-7 :** Darstellung eines Containers mit aktivierter Filterung

### 6.2.3 Vorlagen zur Wiederverwendung von Teilen von Dokumenten

Bei genauer Betrachtung von Spezifikationen, ist zu erkennen, dass sich bestimmte Muster in einem Dokument häufig wiederholen. Abbildung 6-8 zeigt ein Beispielmuster. Bei der Erstel-

lung eines computergestützten Trainings wiederholt sich die Spezifikation, die beschreibt, wie nach der Auswertung einer Frage reagiert werden soll.

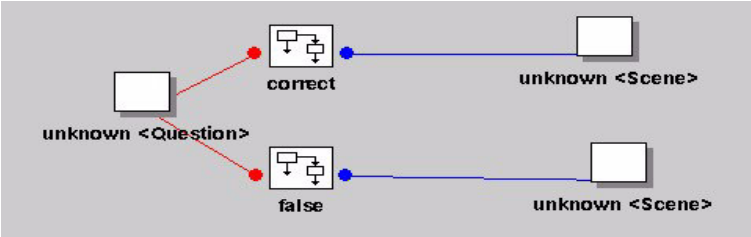


**Abb. 6-8 :** Beispiel für eine wiederkehrende Struktur

Die Erstellung der Struktur ist nicht kompliziert, wohl aber sehr zeitintensiv. Bei mehreren Fragen wird die Struktur aus einfachen Interaktionsschritten aufgebaut, die für jede Frage wiederholt werden. Das Erstellen von Dokumenten kann erheblich beschleunigt werden, wenn ein Konzept zur Wiederverwendung von Teilen einer Spezifikation eingeführt wird.

Um eine Dokumenten-Teilstruktur im gleichen oder in anderen Dokumenten wiederzuverwenden, muss sie zuerst gespeichert werden. Dafür bietet das Autorenwerkzeug die Möglichkeit, die aktuell selektierten Dokumentelemente in einer Vorlagen-Datei zu speichern.

Ein zum Medien- oder Operatorenauswahlfenster analoges Fenster bietet die Möglichkeit, alle Vorlagen aus einem Verzeichnis des Dateisystems mittels Drag&Drop dem geöffneten Dokument hinzuzufügen. Die in der Vorlagen-Datei enthaltenen Medienelemente und Operatoren werden dem Dokument direkt hinzugefügt. Es ist also nicht zu erkennen, ob die Dokumentelemente aus einer Vorlage stammen oder nicht.

Um dem Autor bei der Entwicklung von Vorlagen mehr Flexibilität zu gewähren, wurde speziell für Vorlagen ein so genannter Platzhalter (engl. placeholder) eingeführt. Ein Platzhalter kann für ein Medienelement stehen, das später genau an diese Stelle eingefügt werden muss. Der Platzhalter kann bereits mit Operatoren verbunden sein. Beim Ersetzen des Platzhalters durch ein konkretes Medienelement oder einen Container übernimmt das neue Dokumentelement die Verbindungen mit den Operatoren. Bei der Übernahme der Verbindungen wird überprüft, ob alle Verbindungen gültig sind. Sind nicht alle Verbindungen gültig, wird der Ersetzungsprozess

abgebrochen, da davon ausgegangen werden muss, dass die Vorlage nicht im Sinne ihres Erstellers verwendet wird.

Abbildung 6-9 zeigt nochmals die Struktur aus Abbildung 6-8, nur sind diesmal die drei Container durch Platzhalter ersetzt. Als Icon für die Platzhalter wurde eine stilisierte Wolke gewählt, was zum Ausdruck bringen soll, dass noch nicht festgelegt ist, welches Medienelement sich an dieser Stelle befinden wird.

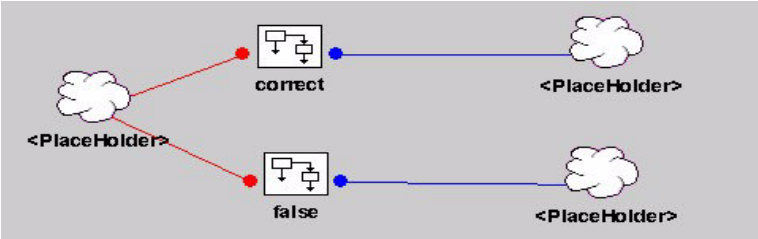


**Abb. 6-9 :** Die Struktur als Vorlage mit Platzhaltern

Nun steht es dem Autor frei, zu bestimmen, wie es beispielsweise nach der korrekten Beantwortung der Frage weitergehen soll. Er kann in der Präsentation entweder eine positive Rückmeldung geben oder direkt zur nächsten Frage kommen.

Würde die Struktur aus Abbildung 6-8 als Vorlage gewählt, wäre festgelegt, dass es nach der richtigen Beantwortung mit einem Szenen-Container weiterginge. Um mit einer Frage weiterzumachen, müsste in diesem Fall der Szenen-Container gelöscht, dann ein Frage-Container eingefügt und auf Grund des Löschens der *correct*-Operator wieder mit dem Frage-Container verbunden werden.

Platzhalter unterstützen die Kombination von Vorlagen, indem ein Platzhalter durch eine Vorlage ersetzt werden kann. Dadurch kann im vorangegangenen Beispiel das Löschen des Szenen-Containers vermieden werden. Eine zweite Vorlage für die Frage selbst besteht aus einem *Frage*-Container. Diese zweite Vorlage kann direkt den Platzhalter ersetzen und dessen Verbindungen übernehmen.

Vorlagen sind ein Konzept, um von einer konkreten Spezifikation abstrahieren zu können. So ist beispielsweise eine Multiple-Choice-Frage eine Vorlage und kein neues Containerelement.

Damit muss die Multiple-Choice-Frage auch nicht programmiert werden, sie wird im Autorenwerkzeug spezifiziert und als Vorlage zur Wiederverwendung bereitgestellt.

Damit lässt sich die in Kapitel 3.4 eingeführte Klassifikation der Anwendungselemente dahingehend erweitern, dass sie nicht nur Medienelemente und Operatoren enthält, sondern auch anwendungsspezifische Vorlagen.

### 6.2.4 Konzeptspezifische Sichten auf Dokumente

Im Rahmen der prototypischen Implementierung des erweiterbaren Ansatzes wurden die Möglichkeiten untersucht, inwieweit spezielle WYSIWYG-Sichten in das Autorenwerkzeug integriert werden können.

Unter einer konzeptspezifischen Sicht wird eine graphische Visualisierung des Konzepts verstanden, die gleichzeitig eine graphisch interaktive Manipulation der Spezifikation ermöglicht.

Bedingt durch die zeitliche und interaktive Komponente von Multimedia-Dokumenten ist das WYSIWYG-Prinzip für Multimedia-Dokumente im Allgemeinen nicht haltbar. Werden einzelne Aspekte des Dokuments herausgenommen und separat betrachtet, ist es möglich, die tatsächliche Erscheinungsform während der Präsentation des Dokumentes schon beim Editieren für den Autor darzustellen.

Bei der räumlichen Anordnung von Medienelementen kann beispielsweise durch einen Schnappschuss zu einem bestimmten Zeitpunkt die aktuellen Positionen der Medienelemente dargestellt und gegebenenfalls editiert werden.

Für den zeitlichen Verlauf eines Dokumentes können mittels Balken über einer Zeitachse komplexere Spezifikationsmodelle (z.B. temporale Intervall-Operatoren [All83, WaR93]) vereinfacht dargestellt werden. So bieten existierende Ansätze wie TIEMPO [Wir99a] oder Madeus [JLR98], die beide die temporale Spezifikation des Dokumentes mittels Intervall-Operatoren erlauben, eine zusätzliche Sicht auf Dokumente mittels der beschriebenen Zeitachse.

Dazu ist anzumerken, dass es nicht für alle Konzepte eine WYSIWYG-Visualisierung gibt. Beispielsweise bereiten Interaktionen und daraus folgende, nicht vorab bestimmbare Zeitintervalle, Probleme bei der Visualisierung.

Abbildung 6-10 zeigt ein Beispiel für eine konzeptspezifische Sicht, die der Autor mittels Auswahlmenü aufgerufen hat. Die konzeptspezifische Sicht ersetzt die Icon-Ansicht im entsprechenden Fenster. In dem Beispiel handelt es sich um die Visualisierung des Konzepts der absoluten räumlichen Positionierung [PAP95] von Medienelementen in der Präsentationsebene. Die aktuelle Position und Größe eines Medienelements wird durch graue Rechtecke dargestellt. Die Darstellung basiert auf den im Container verwendeten Operatoren des räumlichen Konzepts. Damit erhält der Autor einen Eindruck, wie das Layout tatsächlich aussieht. Die graphisch interaktive Manipulation der Spezifikation erlaubt mittels Drag&Drop die Neupositionierung von Medienelementen oder das Verändern der Größe eines Medienelements. Entspricht das Layout den Wünschen des Autors, kann er die konzeptspezifische Sicht dazu veranlassen, die Operatoren des Containers entsprechend der Darstellung zu verändern. Dabei werden nicht vorhandene Operatoren erzeugt und die Parameter der Operatoren so angepasst, dass das Layout bei der Präsentation entsprechend wiedergegeben wird.

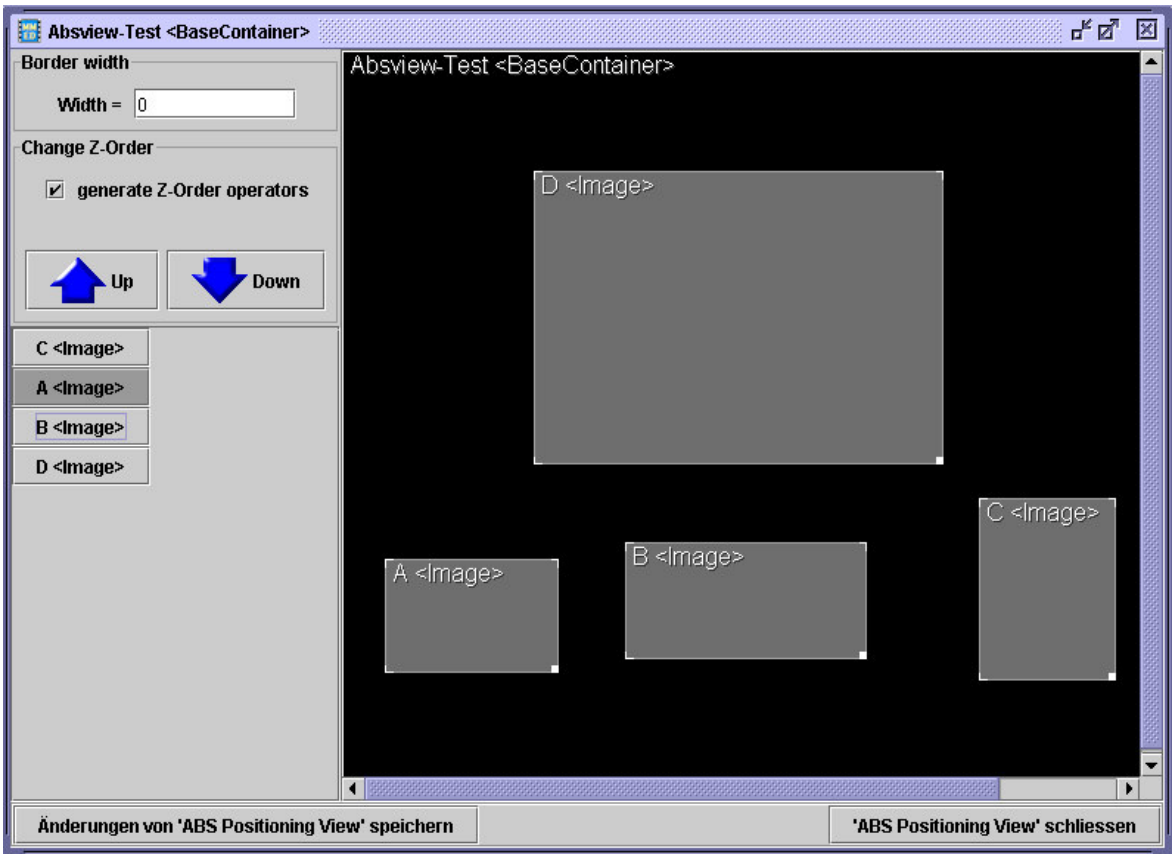


**Abb. 6-10 :** Ein Beispiel für eine konzeptspezifische Sicht

## 6.3 Das Autorenwerkzeug

In Abbildung 6-11 ist ein Bildschirmabzug des Autorenwerkzeugs zu sehen. Es verdeutlicht die Arbeitsweise des Autors. Während des Erstellungsprozesses sind dabei verschiedene Fenster gleichzeitig geöffnet.

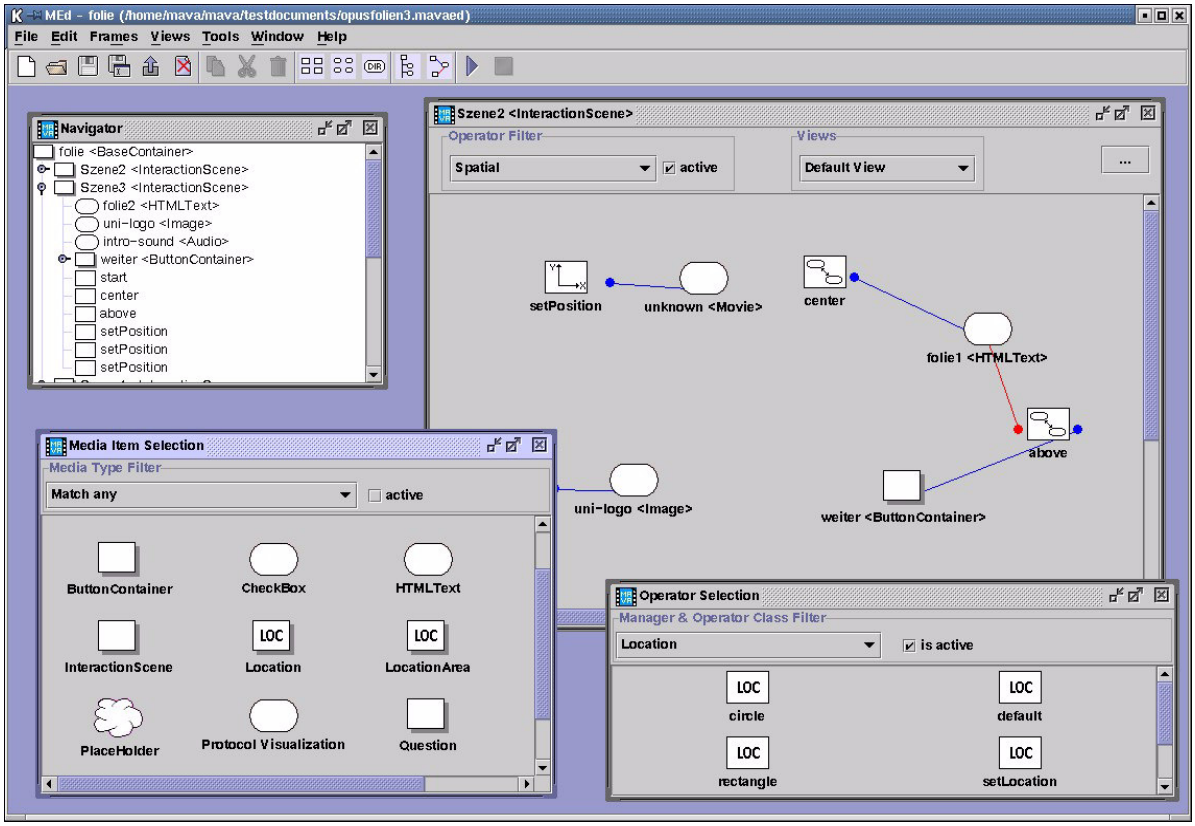


**Abb. 6-11 :** Bildschirmabzug des Autorenwerkzeugs

Das Autorenwerkzeug bietet eine Menüstruktur zur Steuerung des Ablaufs und eine Werkzeugleiste, die es erlaubt, häufig benötigte Funktionen schneller zu erreichen. Neben den Menüs und der Werkzeugleiste sind die Baumansicht und ein Container-Fenster zu sehen.

In der Abbildung sind die beiden Dokumentelemente-Auswahlfenster zu sehen. Links unten befindet sich das Medien- und Containerelemente-Auswahlfenster. Rechts unten ist das Operatoren-Auswahlfenster zu sehen. Beide Auswahlfenster bieten die Möglichkeit, die dargestellten Dokumentelemente nach unterschiedlichen Gesichtspunkten zu filtern.

Durch Filter ist es möglich, nur diejenigen Medienelemente anzuzeigen, die zu einem bestimmten Anwendungsgebiet gehören. Der *Frage*-Container gehört beispielsweise zum Anwendungsbereich computergestütztes Training und bleibt somit sichtbar, wenn der entsprechende Filter aktiviert wird. Ein Filter für Operatoren kann dazu verwendet werden, nur Operatoren für die Spezifikation des räumlichen Layouts eines Dokuments anzuzeigen.

Um Dokumente schon während des Editierens anzeigen und testen zu können, bietet das Autorenwerkzeug die Möglichkeit, das Präsentationssystem zu starten. Dabei kann der Benutzer entweder die gesamte Präsentation betrachten oder nur einen gerade editierten Container. Diese Auswahlmöglichkeit ermöglicht ein schnelles und flexibles Testen der Multimedia-Präsentation, da die Präsentation nicht immer komplett betrachtet werden muss. Das ist von Vorteil, wenn beispielsweise nur am Ende der Präsentation etwas geändert wurde.

Wie das Präsentationssystem wurde auch das Autorenwerkzeug in Java implementiert. Abbildung 6-12 zeigt die Architektur des erweiterbaren Autorenwerkzeugs. Im Folgenden werden die einzelnen Komponenten des Autorenwerkzeugs beschrieben. Im nächsten Abschnitt wird die Umsetzung der dynamischen Erweiterbarkeit im Detail dargestellt.

- *Benutzungsschnittstelle*
  Das Benutzungsschnittstellen-Modul umfasst alle Element der Benutzungsoberfläche des Autorensystems. Es bildet die Hülle des Autorensystems, in die alle Komponenten integriert werden. Durch Interaktion mit der Benutzungsoberfläche werden die integrierten Komponenten gesteuert.

- *Dokument Elemente und Ansichten Repository (DEAR)*
  Das DEAR dient der zentralen Verwaltung und dem Abrufen der aktuellen Editorkonfiguration. Es wird von anderen Komponenten des Autorenwerkzeugs, beispielsweise der grafischen Benutzungsoberfläche, zur Realisierung der Erweiterbarkeit verwendet.

- *DEAR-Steuerung*
  Die DEAR-Steuerung ist für das Laden und Speichern der Editor-Konfiguration verantwortlich. Zusätzlich enthält sie die Funktionalität, um Beschreibungen und Programmcode-Pakete von Erweiterungen dynamisch der aktuellen Konfiguration hinzuzufügen.

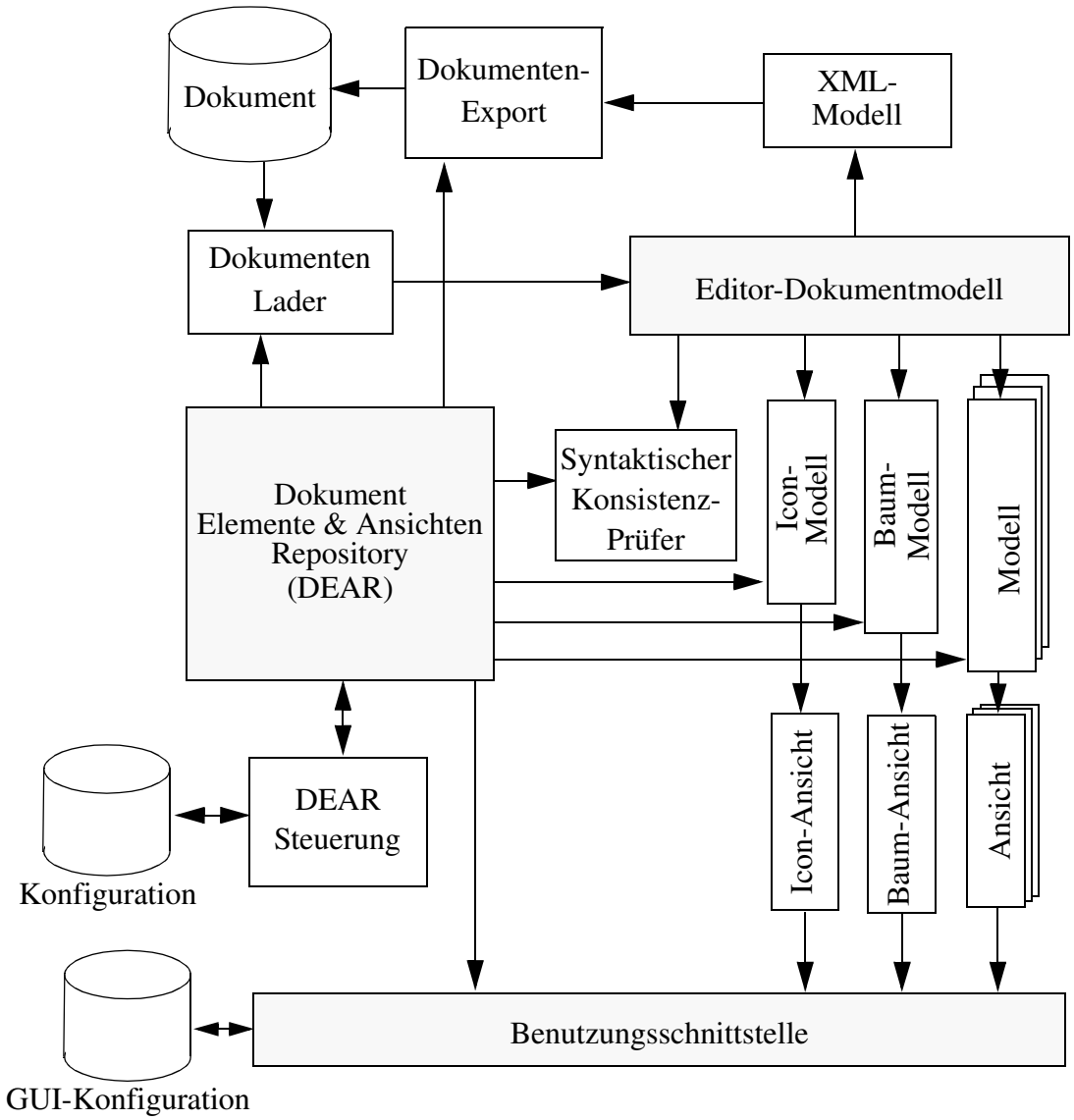


**Abb. 6-12 :** Architektur des Autorenwerkzeugs

- *Editor-Dokumentenmodell*

  Das Editor-Dokumentenmodell ist die interne Darstellung des gerade editierten Dokuments. Es stellt Funktionen zu seiner Manipulation (hervorgerufen durch das Editieren) zur Verfügung. Im Vergleich zum Dokumentenmodell des Präsentationssystems enthält es ausschließlich Funktionalität, die vom Autorensystem benötigt wird. Dementsprechend enthält das Dokumentenmodell des Präsentationssystems nur die Funktionalität, die für die Präsentation benötigt wird. Es findet also eine Trennung der Funktionalität je nach Verwen-

dung statt. Damit werden die Programmcode-Pakete, die vor einer Präsentation geladen werden kleiner und das Laden benötigt weniger Zeit.

- *Dokumenten-Lader*

  Der Dokumenten-Lader lädt einzelne Dokumente. Dabei wird das im vorherigen Kapitel beschriebene XML-basierte Transferformat verwendet. Die Funktionalität des Dokumenten-Lader sowie des Dokumenten-Exports wurden bereits ausführlich in Kapitel 5 beschrieben.

- *Dokumenten-Export*

  Der Dokumenten-Export ist für die Speicherung des gerade editierten Dokuments zuständig. Mit dieser Komponente können zwei unterschiedliche Arten von Speicherformaten erzeugt werden. Die erste Variante dient zum Speichern der kompletten Editorinformationen (z.B. Layout der Visualisierung in der Icon-Ansicht, gerade geöffnete Fenster oder aktivierte Filter). Die zweite Variante speichert lediglich die Informationen, die für die Präsentation des Dokumentes notwendig sind. Das bedeutet, dass diese Variante später nicht mehr zum Editieren verwendet werden kann. Die zweite Variante benötigt weniger Speicherplatz und bietet sich deswegen zur Speicherung auf Servern an.

- *Syntaktischer Konsistenz-Prüfer*

  Der syntaktische Konsistenz-Prüfer realisiert die erste Phase der Konsistenzprüfung von Dokumenten. Dabei wird die syntaktisch korrekte Verwendung der Operatoren überprüft (siehe Kapitel 4). Die konsistente Verwendung von Operatoren kann von ihm nicht überprüft werden, da diese Prüfung von den entsprechenden Anwendungskonzepten abhängt. Dafür müssen mit der Erweiterung entsprechende Konsistenzprüfungsmodule mitgeliefert werden, die der Konsistenz-Prüfer dann aufruft.

- *Modell*

  Mit einem Modell kann eine geeignete spezialisierte Sicht auf das Editor-Dokumentenmodell realisiert werden. Das Autorensystem verfügt über drei unterschiedliche Modelle, die nachfolgend beschrieben werden.

- *XML-Modell*

  Das XML-Modell stellt die aus Sicht einer Weiterverarbeitung in XML benötigten Infor-

mationen zur Verfügung. Hauptfunktionalität ist die Abbildung des Editor-Dokumentenmodells auf die XML-basierte Darstellung, wie sie in Kapitel 5 beschrieben wurde. Der Dokumenten-Export baut auf dieses Modell auf.

- *Icon-Modell*

  Das Icon-Modell ist die Grundlage für die Darstellung des Inhalts eines Containers durch einen Graphen. Die Dokumentelemente werden als Knoten und die Verbindungen zwischen Operatoren und Medienelementen durch Kanten dargestellt. Das Icon-Modell umfasst dabei alle Informationen, die für die graphische Darstellung wichtig sind (z.B. Positionen von Icons in Fenstern).

- *Baum-Modell*

  Das Baum-Modell ist die Grundlage zur Darstellung der Beinhaltet-Beziehung zwischen Containern und anderen Dokumentelementen. Es vermittelt zwischen der Baumansicht und dem Editor-Dokumentenmodell.

- *Ansicht*

  Eine Ansicht ist eine (graphische) Visualisierung eines Modells. Ein Teil der Ansichten erlaubt über direkte Manipulation das Verändern des Dokuments. Das Autorensystem verfügt über zwei verschiedene Ansichten, die nachfolgend beschrieben werden.

- *Icon-Ansicht*

  Die aus Sicht des Benutzers wichtigste Sicht, die Icon-Ansicht, setzt die durch das Icon-Modell repräsentierten Informationen in eine graphische Darstellung um. Die Ansicht erlaubt das Editieren des Dokumentes durch die direkte Manipulation des Dokumenten-Graphen.

- *Baum-Ansicht*

  Die Baum-Ansicht stellt das Baum-Modell im Navigator-Fenster dar. Diese Darstellung ist mit der eines Datei-Browsers vergleichbar. In ihr können einzelne Teile des Dokumentes ein- und ausgeblendet werden. Des weiteren kann durch die Auswahl eines Baumknotens die Icon-Ansicht eines Teilbaums (genau genommen eines Containers) aufgerufen werden.

## 6.4 Dynamische Konfiguration

Neben dem Präsentationssystem muss natürlich auch das Autorenwerkzeug die dynamische Erweiterbarkeit unterstützen. In diesem Abschnitt werden die Bereiche des Autorenwerkzeugs besprochen, in denen die Erweiterbarkeit eine Rolle spielt.

Die aktuell verfügbare Funktionalität wird als Editor-Konfiguration des Autorenwerkzeugs bezeichnet. Unter der dynamischen Konfiguration wird das Verändern der Konfiguration während der Laufzeit des Autorenwerkzeugs verstanden.

Die aktuelle Editor-Konfiguration wird durch eine Konfigurationsbeschreibung festgelegt, die beim Starten des Autorenwerkzeugs gelesen wird. Die Benutzungsoberfläche wird anhand dieser Beschreibung angepasst. Um eine Erweiterung zu integrieren, kann zur Laufzeit eine weitere Konfigurationsbeschreibung dazugeladen werden, die alle Elemente der Erweiterung beschreibt. Die beiden Konfigurationsbeschreibungen werden vereinigt und die Benutzungsoberfläche an die vereinigte Konfigurationsbeschreibung angepasst.

Um die Editorkonfiguration zu speichern, wird die komplette Konfiguration in einem XML-basierten Format in eine Datei geschrieben. Diese Datei wird beim Start des Autorenwerkzeugs gelesen und die Benutzungsschnittstelle entsprechend konfiguriert. Bei dem XML-basierten Format handelt es sich um eine sequentielle Aufzählung aller Elemente der Editorkonfiguration.

In erster Linie stehen die zur Spezifikation benötigten Operatoren und Medienelemente an der Benutzungsoberfläche zur Verfügung. Möchte der Autor ein Dokument in einem Anwendungsgebiet erstellen, für das die Operatoren oder Medienelemente nicht zur Verfügung gestellt werden, so muss er die benötigten Operatoren und Medienelementen durch eine Erweiterung dem Autorenwerkzeug hinzufügen.

Während Medienelemente lediglich in der Konfigurationssprache beschrieben werden, muss für andere Erweiterungen zusätzlich der Programmcode mitgegeben werden, der ihre spezielle Funktionalität realisiert.

Das Autorensystem führt während des Editierens eine Konsistenzprüfung der Spezifikation durch. Die Überprüfung der syntaktischen Konsistenz wird direkt durch das Autorensystem für alle anwendungsspezifischen Erweiterungen durchgeführt. Da die Überprüfung der anwen-

dungsspezifischen Konsistenz von den Erweiterungen abhängig ist, muss dafür eine Implementierung mitgeliefert werden.

Die Benutzungsschnittstelle bietet zusätzlich konzeptspezifische Sichten auf einen Container an, die, wie in Kapitel 6.2.4 beschrieben, auf den Operatoren eines Konzepts basieren. Deshalb muss es auch möglich sein, dem Autorenwerkzeug neben neuen Operatoren einen Konsistenz-Prüfer und neue konzeptspezifische Sichten hinzuzufügen.

Für Medienelemente bietet der Parameter-Dialog (Kapitel 6.1.1) einen Medieninspektor an, der Parameter automatisch setzen kann. Medieninspektoren können ebenfalls dem Autorenwerkzeug hinzufügen werden.

Die Editorkonfiguration ist eine Datenstruktur, die Beschreibungen für alle Elemente enthält, die nicht festgelegt sind, also durch die Erweiterbarkeit hinzugefügt werden können. Dies trifft auf die Dokumentelemente, die Konsistenz-Prüfer für ein Anwendungskonzept, die Filter in der Containeransicht, die Medieninspektoren und die konzeptspezifischen Sichten zu. Ein Teil der Elemente benötigt zusätzlich Programmcode-Pakete, um seine Funktionalität zu realisieren.

Abbildung 6-13 zeigt exemplarisch an einer XML-basierten Beschreibung eines Medienelements, wie diese Beschreibung aufgebaut ist und welche Informationen sie umfasst. In dieser Beschreibung wird jedes Medienelement durch ein `Element`-Tag definiert. Dieses Tag enthält alle Informationen, die zur Erzeugung des XML-basierten Transferformats und zur Darstellung der Elemente im Autorenwerkzeug notwendig sind. Das erste Attribut `icon` bestimmt, welches Icon für die Darstellung verwendet wird. Das zweite Attribut `ImplementedInterfaces` beschreibt die Eigenschaften des Medienelements anhand der implementierten Schnittstellen. Die Schnittstellen werden während des Editierens zur Überprüfung der syntaktischen Konsistenz verwendet.

Anschließend werden die Attribute aufgelistet, welche die für das Speichern notwendigen Informationen beinhalten. Dazu gehören die Angaben, welches Tag für diese Element verwendet werden soll, in welchem Namensraum das Tag liegt, wie das Programmcode-Paket heißt, das die Klasse enthält und welches der Klassenname ist, der die Klasse implementiert.

```
<Element  icon="images/mediaitem.gif"
          ImplementedInterfaces="mava.documentmodel.
                interfaces.BasicMedia;mava.documentmodel.
                interfaces.VisibleMedia"
          xmlTag="Image" nameSpace="de.uni-stuttgart.mava"
          className="mavax.baseelements.Image"
          package="image.jar"
          url="http://ftp.mava.XX/extensions/image.jar" >

  <AttributeList>

    <Attribute  name="Name"
                type="user"
                visName="Media Item Name" />

    <Attribute  name="Type"
                value="Image"
                type="fixed" />

    <Attribute  name="Width"
                type="user"
                visName="Media Item Width"/>

    <Attribute  name="Height"
                type="user"
                visName="Media Item Height" />

    <AttributeURL name="URL"
                  type="user"
                  visName="Media Data URL" />

    <Attribute  name="Duration"
                type="user"
                visName="Presentation Duration"/>

  </AttributeList>

</Element>
```

**Abb. 6-13 :** Beispiel der Beschreibung anhand des Bild-Medienelements

Das `Element`-Tag beinhaltet ein weiteres Tag, und zwar das `AttributeList`-Tag. Es beschreibt alle Parameter des Medienelements in einer Liste. Die Parameter werden in verschiedene Klassen eingeteilt, die unterschiedlich im Autorenwerkzeug visualisiert werden können (vgl. Abbildung 6-2). Dazu enthält das `AttributeList`-Tag weitere Tags aus der folgenden Liste:

- `Attribute`

  Attribute dieser Klasse stellen eine Zeichenkette dar. Ihre Eingabe erfolgt über ein Textfeld.

- `AttributeURL`

  Attribute dieser Klasse stehen für URLs. Sie können in ein Textfeld eingegeben oder durch einen Dateiauswahldialog bestimmt werden.

- `AttributeBoolean`

  Attribute dieser Klasse stehen für einen Wahrheitswert. Sie werden durch ein Kontrollkästchen eingegeben.

- `AttributeColor`

  Damit können Farbwerte dargestellt werden. Es kann ein Farbauswahldialog aufgerufen werden, der den Benutzer bei der Eingabe unterstützt.

- `AttributeIntRange`

  Bei dieser Klasse können Zahlenwerte aus einem bestimmten Intervall ausgewählt werden. An der Benutzungsoberfläche erscheint ein Schieberegler. Beispielsweise kann damit die Lautstärke beim Abspielen eines Audiomediums zwischen 0% und 100% festgelegt werden.

Jedes Attribut hat einen Typ (das `Type`-Attribut des Attribut-Elements). Er kann entweder vom Typ `user` sein, was heißt, dass der Benutzer den Wert verändern darf oder vom Typ `fixed`, dann kann der Wert nicht verändert werden.

Jeder Parameter hat einen Namen, dieser wird durch das `name`-Attribut festgelegt. Der Name dient nur der internen Verarbeitung. Er wird später dazu verwendet, im Objektmodell den entsprechenden Parameter zu setzen. Dazu wird eine Methode aufgerufen, die sich aus dem Namen und einem vorangestellten "`set`" ergibt. Für den ersten Parameter mit dem Namen "`Width`" wird eine Methode "`setWidth()`" des *Image*-Medienelements aufgerufen. Auf Grund der Abbildung des name-Attributs auf eine Java-Methode darf der Name keine Zeichen enthalten,

die nicht für einen Methoden-Namen in Java verwendet werden dürfen (beispielsweise das Leerzeichen).

Aus diesem Grund gibt es ein zusätzliches `visName`-Attribut. Es beinhaltet eine Zeichenkette, die von der Benutzungsoberfläche dazu verwendet werden kann, den Namen dieses Parameters darzustellen. So zeigt der Parameter-Dialog vor den Eingabefeldern den Visualisierungsnamen und nicht den Parameternamen an (vgl. Abbildung 6-2).

Durch diese Trennung ist eine Anpassung an unterschiedliche Sprachen möglich. Um die Benutzungsoberfläche in einer anderen Sprache anzuzeigen, muss jeweils nur der Visualisierungsname geändert werden. In diesem Zusammenhang wird von der Lokalisierung einer Anwendung gesprochen. Das `name`-Attribut kann unverändert bleiben und die entsprechenden Methoden werden immer noch gefunden.

Für einen Teil der Erweiterungen des Autorenwerkzeugs ist neben der Beschreibung ein Programmcode-Paket notwendig, das die entsprechende Funktionalität umsetzt. Dies ist bei den Konsistenz-Prüfern für die Operatoren eines Anwendungskonzepts, bei den Medieninspektoren und den konzeptspezifischen Sichten der Fall. Zum Laden der Programmcode-Pakete wird der gleiche Klassenlademechanismus verwendet wie für das Laden der Programmcode-Pakete im Präsentationssystem.

Ähnlich der Beschreibung der verfügbaren Medienelemente ist auch die Beschreibung der Konsistenz-Prüfer, Medieninspektoren und konzeptspezifischen Sichten aufgebaut. Die Vorgehensweise wird exemplarisch anhand eines Medieninspektors gezeigt. Für die beiden anderen Erweiterungen ist die Vorgehensweise analog.

Die Beschreibung eines Medieninspektors legt fest, für welchen Medientyp er die Werte von Attributen bestimmen kann (beispielsweise anhand des MIME-Typs oder der Erweiterung des Dateinamens). Außerdem bestimmt sie den Namen und die URL für ein Programmcode-Paket, das alle für die Implementierung benötigten Klassen enthält. Zusätzlich wird der Name einer Klasse angegeben, die die Medieninspektor-Schnittstelle implementiert. Die Beschreibung für einen Inspektor, der für MPEG-Videos die Attribute bestimmt, sieht beispielsweise wie folgt aus.

```
<Inspector extension="mpeg" className="medx.MovieInspector"
                package="movieinspector.jar" />
```

Abbildung 6-14 zeigt die Medieninspektoren-Schnittstelle. Die Schnittstelle legt drei Methoden fest, die von jeder Medieninspektoren-Klasse implementiert werden müssen. Die Methode `inspect()` beauftragt den Medieninspektor, die zuletzt mittels `setURL()` mitgeteilte Adresse für Mediendaten zu analysieren. Durch den Aufruf der Methode `getAttributes()` kann eine Liste von Attributen angefordert werden. Diese Liste enthält alle Attribute, die vom Medieninspektor mit Werten, basierend auf den entsprechenden Mediendaten, gesetzt werden konnten. Die Benutzungsoberfläche kann diese Werte in den Parameter-Dialog übernehmen.

```
public interface MediaInspektor
{
  public boolean inspect();
  public void setURL(String URL);
  public AttributeList getAttributes();
}
```

**Abb. 6-14 :** Schnittstellen-Definition eines Medieninspektors

## 6.5 Bewertung und Vergleich

Unter visueller Programmierung wird die graphisch interaktive Erstellung von Programmen verstanden [GoR90]. Wichtige Bereiche der visuellen Programmierung sind die graphische Darstellung von (Programm-)Strukturen und deren interaktive Manipulation. Oft liegt hier das Prinzip der direkten Manipulation zu Grunde [Shn83]. Darunter wird eine graphische Darstellung des Programms verstanden, die durch den Autor verändert werden kann. Über den Erfolg oder Misserfolg der Änderung bekommt er sofort eine Rückmeldung.

Ein weiterer wichtiger Punkt ist die Komplexität der visuellen Sprache und die damit zusammenhängende Komplexität der Visualisierung und Manipulation.

Die Darstellung von visuellen Sprachen kann durch einen Objekt-Graphen erfolgen [GoR90]. Die Graphstruktur dient dabei als abstrakte Syntax der visuellen Sprache. Mayers [May90] betrachtet die Dimensionalität der visuellen Sprache. Visuelle Sprachen sind demnach mehrdimensional im Gegensatz zu textuellen Sprachen, die als eindimensionaler Strom verarbeitet

werden. Ein Beziehungsgraph zwischen Objekten ist demnach mehrdimensional. Für eine äquivalente textuelle Darstellung muss diese Mehrdimensionalität aufgebrochen werden. Aus diesem Grund unterscheidet Burnett [Bur00] zwischen natürlichen visuellen Sprachen, die mehrdimensional sind, und solchen, die lediglich graphisch repräsentiert werden. Damit kann die Dokumentenspezifikationssprache des erweiterbaren Ansatzes als eine mehrdimensionale visuelle Sprache bezeichnen werden.

Werden Multimedia-Dokumente als interaktive Anwendungen betrachtet, so kann der Begriff visuelle Programmierung auf die Erstellung von Multimedia-Dokumenten übertragen werden. Üblicherweise wird statt des Begriffs Programmierung die Bezeichnung Authoring verwendet.

Laut Definition der visuellen Programmierung darf keine textuelle Eingabe von Programmcode erfolgen. Bieten Autorenwerkzeuge eine Skriptsprache an, die zur Erstellung von Dokumenten (in diesem Fall Programmierung) herangezogen werden muss, so verletzen diese das Prinzip der visuellen Programmierung. Dies ist z.B. bei Macromedia Director der Fall. Aus dem gleichen Grund werden Entwicklungsumgebungen, die es ermöglichen, die Benutzungsoberfläche durch ein Werkzeug graphisch interaktiv zu erzeugen, aber in denen die Funktionalität weiterhin programmiert werden muss, nicht als visuelle Programmiersprachen bzw. Werkzeuge bezeichnet. Beispiele hierfür sind Borland JBuilder und Microsoft Visual Basic.

Dem hier vorgestellten Autorensystem liegt die visuelle Programmierung zu Grunde. Das Dokument, bestehend aus den vier Klassen von Dokumentelementen (Medienelement, Operator, Effektoperator und Container) und den Verbindungen von (Effekt-)Operatoren mit Medienelementen, wird graphisch dargestellt. Der Autor nimmt in dieser graphischen Darstellung Veränderungen am Dokument vor. Er bekommt auch sofort eine Rückmeldung, ob seine Aktion erfolgreich war, z.B. ob ein Operator mit einem Medienelement verbunden werden konnte oder nicht.

Die Erstellung von Multimedia-Dokumenten mittels visueller Programmierung betrachtet nur einen Teilbereich dessen, was Gegenstand der Forschung in diesem Themenbereich war und ist. Weiterführende Informationen zur visuellen Programmierung finden sich unter [Bur00].

## 6.6 Zusammenfassung

In diesem Kapitel wurde ein Autorenwerkzeug zur graphisch interaktiven Spezifikation von Multimedia-Dokumenten, basierend auf dem erweiterbaren Ansatz, konzipiert und die prototypische Implementierung anhand von Bildschirmabzügen gezeigt. Das Autorenwerkzeug ist das wichtigste Werkzeug für einen Autor. Es dient als Schnittstelle zum Dokumentenmodell und bestimmt zusammen mit dem Dokumentenmodell den notwendigen Aufwand bei der Spezifikation eines Dokuments.

In dem hier vorgestellten Ansatz ist das Meta-Dokumentenmodell direkt an der Benutzungsschnittstelle sichtbar. Deshalb ist die geringe Komplexität des Meta-Dokumentenmodells wichtig für die Einfachheit der Benutzungsschnittstelle. Um den Aufwand weiter zu reduzieren, wurde das Autorenwerkzeug um zusätzliche Mechanismen ergänzt, die für einfache, aber häufig wiederkehrende Aufgaben die Arbeit erleichtern.

Neben der Konzeption wurde die Architektur eines erweiterbaren Autorenwerkzeugs vorgestellt. Die Erweiterbarkeit des Autorensystems spiegelt sich in der dynamischen Konfiguration des Autorenwerkzeugs wieder. Je nach Anwendungsgebiet werden unterschiedliche Operatoren, Medienelemente, Konsistenz-Prüfer und konzeptspezifische Sichten oder Medieninspektoren angeboten. Für die Dokumentelemente ist es ausreichend, eine Beschreibung konkreter Dokumentelemente zu liefern. Für die Konsistenz-Prüfer, die konzeptspezifischen Sichten und die Medieninspektoren wird zusätzlicher Programmcode benötigt, der ihre Funktionalität realisiert.

# 7 ZUSAMMENFASSUNG UND AUSBLICK

Die Erstellung und Präsentation von Multimedia-Dokumenten unterscheidet sich grundlegend von der Erstellung und Präsentation statischer Dokumente. Die wichtigsten Unterscheidungsmerkmale sind die Zeitabhängigkeit und die Interaktionsmöglichkeiten im Verlauf von Multimedia-Präsentationen. Beide Merkmale müssen bei der Erstellung eines Multimedia-Dokuments beachtet und von den Autorenwerkzeugen und den Präsentationssystemen unterstützt werden.

Stand bei bisherigen Arbeiten im Bereich der Modellierung von Multimedia-Dokumenten der zeitliche Aspekt im Mittelpunkt, so beschäftigt sich diese Abhandlung in erster Linie mit Interaktionen. Für die Untersuchungen in dieser Abhandlung spielt die Anwendungsabhängigkeit eine maßgebliche Rolle. Anwendungsspezifische Interaktionen und Reaktionen müssen auf der Ebene der Benutzungsschnittstelle unterstützt werden, d.h. in der Spezifikationssprache reflektiert sein. Somit wird die Erstellung von Multimedia-Dokumenten in speziellen Anwendungsgebieten unterstützt.

Der hier vorgestellte Ansatz erfüllt diese Anforderungen. Besonders bei der Erstellung anwendungsspezifischer Multimedia-Dokumente stellt er sich als vorteilhaft heraus, da er die Verwendung spezieller Spezifikationssprachen ermöglicht. Doch auch bei diesem Ansatz wird ein systematisches Vorgehen vom Autor verlangt. So werden Dokumente in logische Teildokumente aufgeteilt, die jeweils durch einen Graphen spezifiziert werden. Die einzige technische Anforderung an einen Autor ist die Kenntnis über den Aufbau und die Funktionsweise des Dokumentengraphen, der auf dem Meta-Dokumentenmodell basiert. Aufgrund der Anwendungsunabhängigkeit des Meta-Dokumentenmodells ist es jedoch ausreichend, das Prinzip einmal zu verstehen, da die Vorgehensweise in den unterschiedlichen Anwendungsgebieten immer dieselbe bleibt.

Der erweiterbare Ansatz geht davon aus, dass sich alle Aspekte eines Multimedia-Dokuments mittels Beziehungen zwischen Medienelementen darstellen lassen. Diese Beziehungen werden mit Hilfe parametrisierter Operatoren dargestellt. Bei der Spezifikation eines Dokuments wird ein aus Operatoren und Medienelementen bestehender Graph schrittweise aufgebaut. Die

Unterstützung eines bestimmten Anwendungsgebiets wird durch das Anbieten einer Menge von spezifischen Operatoren und Medienelementen erreicht.

Das Autorensystem muss um die Dokumentelemente für neue Anwendungsgebiete erweiterbar sein. Das ist notwendig, da es eine Vielzahl unterschiedlicher Anwendungsgebiete gibt, die bei der Entwicklung eines Präsentations- und Autorensystems nicht bekannt sind. Um ein neues Anwendungsgebiet zu erschließen, müssen lediglich die entsprechenden Operatoren und Medienelemente bestimmt werden. Dafür wird ein Experte mit Programmierkenntnissen benötigt, der die Operatoren und Medienelemente implementiert. Ziel ist jedoch die Wiederverwendung von existierenden Spezifikationssprachen, so dass im Normalfall kein Programmierer zur Realisierung von Erweiterungen benötigt wird.

Die ad-hoc Erweiterbarkeit wird durch das Präsentationssystem realisiert. Wenn ein Dokument abgespielt werden soll, das eine dem Präsentationssystem unbekannte Spezifikationssprache enthält, muss davor dynamisch zur Laufzeit die Funktionalität des Präsentationssystems erweitert werden. Erst damit wird es möglich, das Dokument anzuzeigen. In dieser Abhandlung wurde die Architektur für ein komponentenbasiertes, erweiterbares Präsentationssystem konzipiert, das die ad-hoc Erweiterbarkeit unterstützt.

Der in dieser Abhandlung vorgestellte Ansatz ist ein offener Ansatz. Dies bedeutet, dass jeder Anwender Erweiterungen vornehmen kann. Es müssen lediglich die Operatoren und Medienelemente definiert und implementiert werden. Die restliche Funktionalität des Präsentationssystems und Autorenwerkzeugs wird dabei wiederverwendet. Insbesondere ist keine Standardisierung von Erweiterungen notwendig. Dies macht bei einem späteren Einsatz eine Qualitätssicherung der Erweiterungen nötig.

Zur Speicherung und Übertragung der Multimedia-Dokumente musste auf ein proprietäres, selbstentwickeltes Transferformat zurückgegriffen werden. Kein vorhandenes Format konnte den Anforderungen der Erweiterbarkeit genügen. Um den Anforderungen digitaler Bibliotheken gerecht zu werden, wurde bei der Entwicklung des Formats auf den offenen Standard XML zurückgegriffen. Dies ermöglicht die Weiterverarbeitung von Dokumenten durch andere Werkzeuge. Beispiele hierfür sind die Integration und Verarbeitung von Meta-Daten in bibliothekarischen Nachweissystemen.

Das WYSIWYG-Prinzip ist im Bereich Multimedia auf Grund von Zeitabhängigkeit und Interaktion nur schwer zu erreichen. Deshalb ist es wichtig, dass die Dokumentenmodelle, die zur Spezifikation von Multimedia-Dokumenten herangezogen werden, eine möglichst geringe Komplexität aufweisen. Für den erweiterbaren Ansatz wird dies durch nur vier unterschiedlichen Typen von Elementen des Meta-Dokumentenmodells erreicht. Dies führt zu einer geringen Anzahl von unterschiedlichen Editierschritten, die durch den Autor ausgeführt werden müssen. Zusätzlich bieten die anwendungsspezifischen Instanzen des Meta-Dokumentenmodells Abstraktionen aus den entsprechenden Anwendungsgebieten an.

Die Bewertung des notwendigen Aufwands für die Entwicklung einer Erweiterung lässt sich durchführen, sobald ein Anwendungsgebiet vollständig erschlossen und umgesetzt ist. Bei dieser Bewertung kann einen Vergleich mit den anwendungsspezifischen Ansätzen durchgeführt werden. Die prototypische Implementierung gab einen ersten Hinweis, dass Manager mit geringem Aufwand zu programmieren sind und aus wenig Programmcode bestehen (< 2.000 Programmzeilen).

Die Erstellung von Multimedia-Dokumenten muss in der Zukunft einfacher werden, so wie dies bei der Erstellung textueller Dokumente bereits geschehen ist. Diese Abhandlung hat eine Möglichkeit vorgestellt, wie diese Vereinfachung für Multimedia-Dokumente in unterschiedlichen Anwendungsgebieten aussehen kann. Dafür wurden Modelle und Konzepte entwickelt, deren Umsetzbarkeit in einer prototypischen Implementierung gezeigt wurde.

# Anhang A. BEISPIELSPEZIFIKATION

In diesem Anhang werden die ersten Schritte einer Spezifikation eines sehr einfachen Reiseführers, dem zweiten Beispiel dieser Abhandlung, gezeigt. Die Spezifikation zeigt die Anwendung der Spracherweiterungen und wie mit ihr eine Präsentation spezifiziert wird.

Abbildung A-1 zeigt die Spezifikation in einem *LocationArea*-Container. In ihm wird die komplette Logik zur Beschreibung der Reaktion auf eine Positionsänderung des Benutzers angegeben. Der *default*-Operator wird dazu verwendet um festzulegen, was präsentiert werden soll, wenn der Benutzer sich nicht in einem sensitiven Bereich für eine Sehenswürdigkeit befindet. Durch den *setLocation*-Operator werden einem *Location*-Container die GPS-basierte Koordinaten zugewiesen. Durch den *activationRadius*-Operator wird die Größe des sensitiven Bereichs (in diesem Fall ein Kreis) um die Sehenswürdigkeit festgelegt. Die Spezifikation der Teilpräsentation für eine Sehenswürdigkeit ist in den *Location*-Containern enthalten.

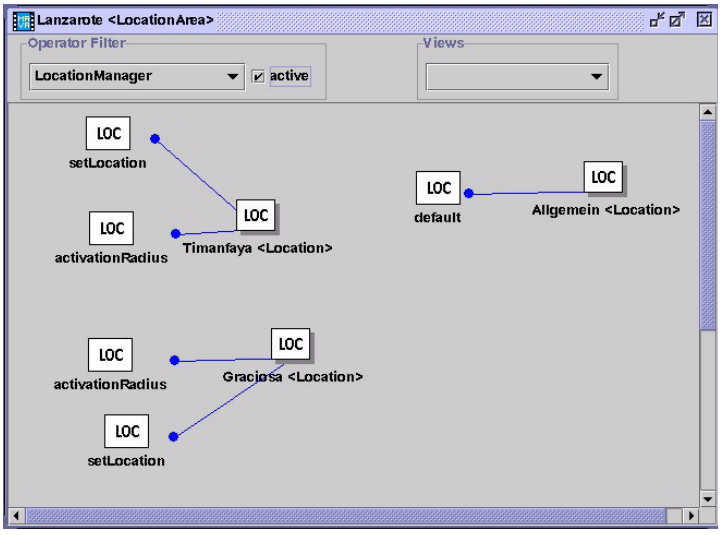


**Abb. A-1 :** Ausschnitt der Spezifikation, der das Reiseführer-Konzept verwendet

Neben dem interaktiven Teil des Reiseführers muss die räumliche Platzierung der Medienelemente bestimmt werden. Dazu ist in Abbildung A-2 die Spezifikation, basierend auf den Operatoren zur relativen räumlichen Positionierung von Medienelementen zu sehen. Es wird der *center*-Operator verwendet, der ein Medienelement in der Präsentationsfläche zentriert.

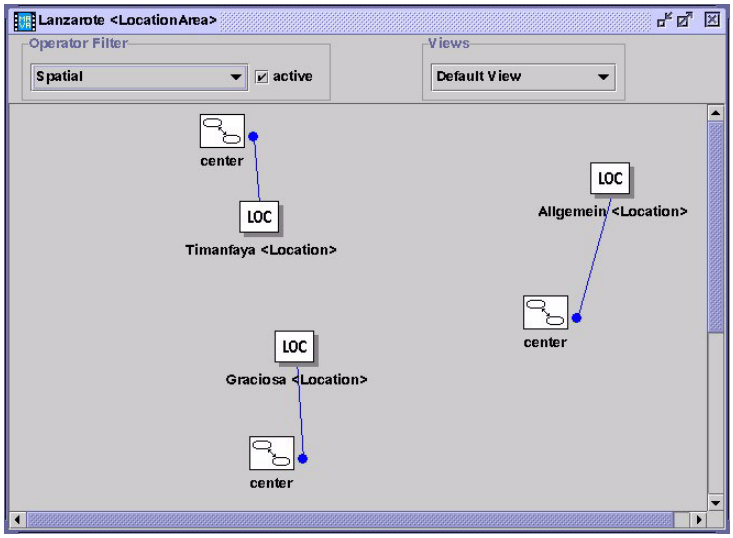


**Abb. A-2 :** Spezifikation des räumlichen Aspekts

Abbildung A-3 zeigt die komplette Spezifikation des Reiseführers ohne aktivierten Filter für ein Konzept. Damit verdeutlicht die Serie von drei Bildschirmabzügen nochmals die Anwendung von Filtern und den daraus resultierenden Nutzen für den Autor.

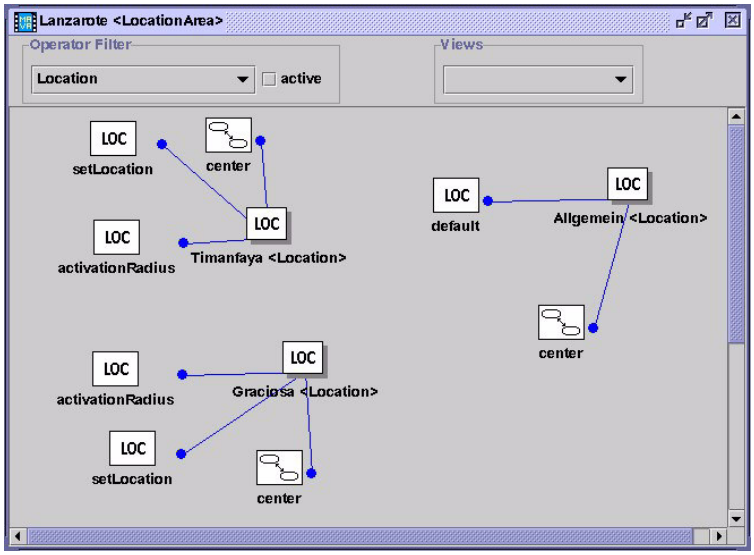


**Abb. A-3 :** Die gesamte Spezifikation

In Abbildung A-4 ist die Spezifikation einer Teilpräsentation zu sehen, die abgespielt wird, wenn der Benutzer sich nahe genug an einer Sehenswürdigkeit befindet. Die Spezifikation ver-

wendet ein räumliches und ein zeitliches Konzept. Mittels *center*-Operator werden die Medienelemente in der Präsentationsfläche positioniert. Der *cobegin*-Operator ist ein temporaler Intervall-Operator [WaR93]. Die Präsentation beginnt mir einem Text, der durch das Audiomedienelement “vorgelesen” wird. Anschließend wird ein Video abgespielt, welches das Farbspiel der Feuerberge im Timanfaya-Nationalpark auf der Kanareninsel Lanzarote zeigt.

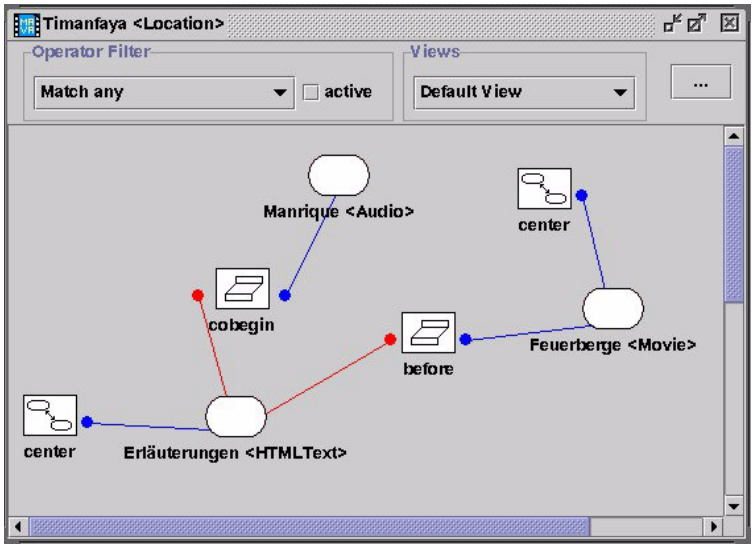


**Abb. A-4 :** Spezifikation eines *Location*-Containers

Abbildung A-5 zeigt einen Bildschirmabzug der Präsentation der Beispielspezifikation, die jedoch drei Bilder anstatt des in Abbildung A-4 angegebenen einzelnen Bildes enthält.

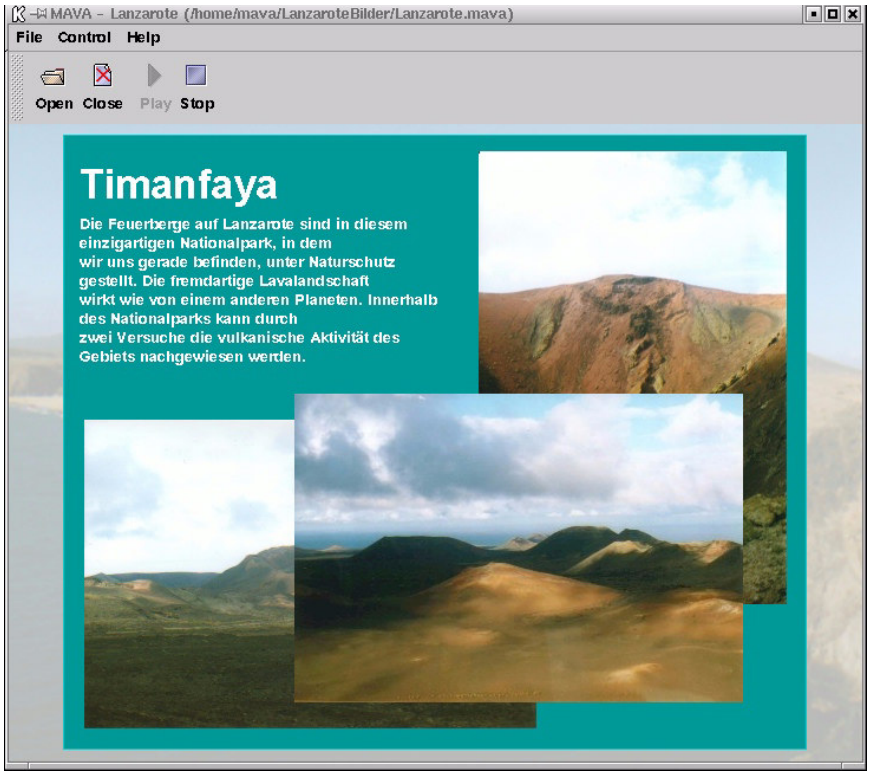


**Abb. A-5 :** Bildschirmabzug der Präsentation des Beispiels

# Anhang B. BEISPIELSPEZIFIKATION IM TRANSFERFORMAT

Im Folgenden ist das Transferformat des Lanzarote-Reiseführers aus Anhang A abgedruckt. Die Erläuterung des Transferformats findet sich in Kapitel 5.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- This DTD was automatically generated by the mava document
      editor med -->
<!-- Do not modify! -->
<!-- Modification of this file may result in an unpresentable
      document -->

<!DOCTYPE mava [
<!ELEMENT source EMPTY >
<!ATTLIST source idref IDREF #REQUIRED >
<!ELEMENT destination EMPTY >
<!ATTLIST destination idref IDREF #REQUIRED >
<!ELEMENT effect EMPTY >
<!ATTLIST effect idref IDREF #REQUIRED >
<!ELEMENT ns1:center ( source | destination )*>
<!ATTLIST ns1:center mava:class CDATA #FIXED "mavax.relposi-
tioning.Center" >
<!ELEMENT ns0:setLocation ( source | destination )*>
<!ATTLIST ns0:setLocation mava:class CDATA #FIXED "mavax.loca-
tion.SetLocation" >
<!ATTLIST ns0:setLocation LocationE CDATA #IMPLIED >
<!ATTLIST ns0:setLocation LocationN CDATA #IMPLIED >
<!ELEMENT ns0:activationRadius ( source | destination )*>
<!ATTLIST ns0:activationRadius mava:class CDATA #FIXED
"mavax.location.Circle" >
<!ATTLIST ns0:activationRadius Radius CDATA #IMPLIED >
<!ELEMENT ns0:default ( source | destination )*>
<!ATTLIST ns0:default mava:class CDATA
                          #FIXED "mavax.location.Default" >
<!ELEMENT ns0:Location ( ns1:center | ns0:setLocation |
                          ns0:activationRadius | ns0:default |
                          ns0:Location | ns0:LocationArea )*>
<!ATTLIST ns0:Location mava:class CDATA
                   #FIXED "mavax.location.LocationContainer" >
<!ATTLIST ns0:Location id ID #REQUIRED >
<!ATTLIST ns0:Location Name CDATA #IMPLIED >
<!ATTLIST ns0:Location Width CDATA #IMPLIED >
```

```
<!ATTLIST ns0:Location Height CDATA #IMPLIED >
<!ATTLIST ns0:Location Duration CDATA #IMPLIED >
<!ATTLIST ns0:Location Color CDATA #IMPLIED >
<!ATTLIST ns0:Location ShowBorder CDATA #IMPLIED >
<!ATTLIST ns0:Location BorderColor CDATA #IMPLIED >
<!ATTLIST ns0:Location Transparent CDATA #IMPLIED >
<!ATTLIST ns0:Location LocationE CDATA #IMPLIED >
<!ATTLIST ns0:Location LocationN CDATA #IMPLIED >
<!ELEMENT ns0:LocationArea ( ns1:center | ns0:setLocation |
                             ns0:circle | ns0:default |
                         ns0:Location | ns0:LocationArea )*>
<!ATTLIST ns0:LocationArea mava:class CDATA
              #FIXED "mavax.location.LocationAreaContainer" >
<!ATTLIST ns0:LocationArea id ID #REQUIRED >
<!ATTLIST ns0:LocationArea Name CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea Width CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea Height CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea Duration CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea Color CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea ShowBorder CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea BorderColor CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea Transparent CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea AreaE CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea AreaN CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea AreaWidth CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea AreaHeight CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea xmlns:mava CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:author CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:organization CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:presentation-width
                                              CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:presentation-height
                                              CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:created CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea mava:lastmodified CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea xmlns:ns1 CDATA #IMPLIED >
<!ATTLIST ns0:LocationArea xmlns:ns0 CDATA #IMPLIED >
<!ELEMENT mava ( ns1:center | ns0:setLocation |
                 ns0:activationRadius |
                 ns0:default |  ns0:Location |
                 ns0:LocationArea | requiredManager |
                 requiredMediaItem )*>
<!ELEMENT requiredManager EMPTY>
<!ATTLIST requiredManager name CDATA #REQUIRED>
<!ATTLIST requiredManager ns CDATA #REQUIRED>
<!ATTLIST requiredManager package CDATA #REQUIRED>
<!ATTLIST requiredManager url CDATA #REQUIRED>
```

```
<!ATTLIST requiredManager version CDATA #IMPLIED>
<!ATTLIST requiredManager testclass CDATA #IMPLIED>
<!ELEMENT requiredMediaItem EMPTY>
<!ATTLIST requiredMediaItem name CDATA #REQUIRED>
<!ATTLIST requiredMediaItem ns CDATA #REQUIRED>
<!ATTLIST requiredMediaItem package CDATA #REQUIRED>
<!ATTLIST requiredMediaItem url CDATA #REQUIRED>
<!ATTLIST requiredMediaItem version CDATA #IMPLIED>
<!ATTLIST requiredMediaItem testclass CDATA #IMPLIED>
]>

<mava>

<!-- Sector information of the document -->

<requiredExtension name="RelativPositioningManager"
            ns="de.uni-stuttgart.informatik.vs.relpositioning"
            package="relpositioning.jar"
            url="file:/d:/mava/implementierung/code/extensions/
                                    relpositioning.jar" />
<requiredExtension name="Location"
                  ns="de.uni-stuttgart.informatik.vs.location"
                  package="location.jar"
                  url="file:/d:/mava/implementierung/code/
                                  extensions/location.jar" />

<!-- Start of the document content -->

 <ns0:LocationArea id="i-1" xmlns:mava="http://www.mava.de"
            mava:author="Jürgen Hauser"
            mava:organization=”Uni Stuttgart"
            mava:presentation-width="640"
            mava:presentation-height="480"
            mava:created="2001-8-31"
            mava:lastmodified="2001-8-31"
      xmlns:ns1="de.uni-stuttgart.informatik.vs.relpositioning"
      xmlns:ns0="de.uni-stuttgart.informatik.vs.location"
      Name="Lanzarote"
      ShowBorder="false"
      Transparent="false" >
  <ns0:Location id="i1" Name="Allgemein" Width="620"
      Height="440" ShowBorder="false" Transparent="false" >
  </ns0:Location>
  <ns0:Location id="i2" Name="Graciosa" Width="620" Height="440"
          ShowBorder="false" Transparent="false" >
  </ns0:Location>
```

```
  <ns0:Location id="i3" Name="Timanfaya" Width="620"
      Height="440" ShowBorder="false" Transparent="false" >
  </ns0:Location>
  <ns0:default >
   <destination idref="i1" />
  </ns0:default>
  <ns0:activationRadius Radius="20" >
   <destination idref="i2" />
  </ns0:activationRadius>
  <ns0:activationRadius Radius="15" >
   <destination idref="i3" />
  </ns0:activationRadius>
  <ns1:center >
   <destination idref="i2" />
  </ns1:center>
  <ns1:center >
   <destination idref="i1" />
  </ns1:center>
  <ns1:center >
   <destination idref="i3" />
  </ns1:center>
  <ns0:setLocation LocationE="48" LocationN="49" >
   <destination idref="i3" />
  </ns0:setLocation>
  <ns0:setLocation LocationE="47" LocationN="50" >
   <destination idref="i2" />
  </ns0:setLocation>
 </ns0:LocationArea>
</mava>
```

# Anhang C. LITERATURVERZEICHNIS

[AdS99] Adobe Systems Inc. Postscript Language Reference (with CD-ROM). Addison Wesley, 5/1999.

[All83] J.F. Allen. Maintaining Knowledge about Temporal Intervals. In *Communication of the ACM*, 26(11):832-843. 11/1983.

[App+98] Vidur Apparao et. al. Document Object Model (DOM) Level 1 Specification (Version 1.0) .http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/. 10/1998.

[Arg96] Arnold, K., Gosling, J. The Java-Programming Language. Addison Wesley. 1996.

[Aut01] Macromedia. Authorware Professional Version 6, World Wide Web Seite http://www.macromedia.com/products/authorware. Macromedia, 2001.

[Bes01] David Beskeen. Microsoft PowerPoint 2000. Thomson Learning. 2001.

[BFM98] T. Berners-Lee, R Fieling und L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396. URL: http://www.ietf.org/rfc/rfc2396.txt. 1998.

[BHL99] Tim Bray, Dave Hollander und Andrew Layman. Namespaces in XML. W3C Recommendation. URL: http://www.w3.org/TR/REC-xml-names. 1/1999.

[Bra99] Karlheinz Brandenburg. MP3 and AAC Explained. AES 17. In Proc. *International Conference on High Quality Audio Coding*. Florence, Italy. 9/1999

[BrS98] T. Bray, J. Paoli und C.M. Sperberg-McQueen. Extensible Mark-up Language (XML) 1.0. Working Recommendation. W3C. 1998.

[Bur00] http://www.cs.orst.edu/<tilde>burnett/vpl.html

[Cal96] Michael Callery. Learning Lingo. Addison Wesley Publishing Company, 1996

[CaW01] Mary Campione und Kathy Walrath. The Java Tutorial. Addison Wesley, 2001

[Dir01] Macromedia. Director Version 8.5. World Wide Web Seite http://www.macromedia.com/products/director. Macromedia, 2001.

[DLm00] DLmeta Initiative. URL: http://www.dlmeta.de. 2000.

[Eck00] Robert Eckstein. XML. Kurz und gut. OReilly / VVA, 2000.

[EHV93] J. L. Encarnacao, W. Hübner, K. Väänänen. Autorenwerkzeuge für multimediale Informationssysteme. In *Informationstechnik und Technische Informatik*, 35(2):31-38, 1993.

[Fla99] "Flash 4.0". URL: http://www.macromedia.com/software/flash. Macromedia. 1999.

[Fla00] David Flanagan. Java in a Nutshell. Deutsche Ausgabe. OReilly/VVA. 2000.

[FrB96] N. Freed und N. Borenstein. Multipurpose Internet Mail Extensions - (MIME) Part One: Format of Internet Message Bodies, RFC 2045. URL: http://www.ietf.org/rfc/rfc2045.txt. 1996.

[Fre01] Alexander Frey. Konzeption und Realisierung der dynamischen Erweiterbarkeit des MAVA-Systems. Diplomarbeit Nr. 1920, IPVR, Universität Stuttgart, 2001

[Gal91] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. In *Communications of the ACM*, 34(4):46–58, 4 1991.

[GiS00] Marc Giradot und Neel Sundaresan. Millau: an encoding format for efficient representation and exchange of XML over the Web. In Proc *9. Internationalen Worl Wide Web Konferenz.* Amsterdam, Niederlande. 5/2000.

[GoR90] Eric Golin und Steven Reiss. The Specification of Visual Language Syntax. In *Journal Visual Languages and Computing*, 2(1):141-157, 1990.

[GRJ99] G. Booch, J. Rumbaugh und I. Jacobson. *Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[HaM01] Elliotte Rusty Harold und W. Scott Means. XML in a Nutshell. Deutsche Ausgabe. Oreilly / VVA, 2001

[HaR00] Jürgen Hauser und Kurt Rothermel. Specificiation and Implementation of an Extensible Multimedia System. Im Tagungsband *Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, Enschede, Holland, LNCS 1905, Springer, Seite 241-253, 10/2000

[HaR01] Jürgen Hauser und Kurt Rothermel. Specification and Implementation of an Extensible Multimedia System. Im Tagungsband *Kommunikation in Verteilten Systemen (KiVS)*, Hamburg, Deutschland, Seite 215-226, 2/2001

[HaS01] Jürgen Hauser und Frank Scholze. Integration und Verarbeitung multimedialer Dokumente in Digitalen Bibliotheken - der MAVA-Ansatz. In *Fachzeitschrift für Bibliotheken B.I.T. onlinem,* 1(4):35-41,1/2001

[HaT01] Jürgen Hauser und Jing Tian. Abstractions in Multimedia Authoring: The MAVA Approach. In Proc: 6. *Eurographics Workshop on Multimedia*, Seite (to appear), Manchester, UK, 9/2001

[Hau99] Jürgen Hauser. Realization of an Extensible Multimedia Document Model. In Proc. *5. Eurographics Workshop on Multimedia '99*, Mailand, Italien, Seite 113-122, 9/1999

[Hau00] Jürgen Hauser. Multimedia Authoring with MAVA. In Proc. *8. International Conference on Digital Documents and Electronic Publishing 2000 (DDEP00)*. Springer, München, Deutschland, 9/2000

[Hau01] Jürgen Hauser. A Component-based Extensible Multimedia System. In Proc. *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01*), Las Vegas, NV, USA, Seite 1243-1249, 6/2001

[HLC92] B. Hofmann-Wellenhof, H.Lichtenegger und J. Collins. Global Positioning System - Theroy and Practice. Springer-Verlag Wien. 1992.

[Hos01] P. Hoschka et al. (Eds.). Synchronized Multimedia Integration Language (SMIL) 2.0 Specification. W3C Proposed Recommendation. W3C. 6/2001.

[HRB93] Hardman, L., Van Rossum, G. und Bulterman, D.C.A. Structured Multimedia Authoring. In Proc. *ACM Multimedia '93 Conference*. 6/1993.

[HSA99] Jürgen Hauser, Frank Scholze und Uwe Albrecht. MAVA - Entwicklung und Integration eines erweiterbaren multimedialen Dokumentensystems. Im Tagungsband zum *89. Dt. Bibliothekartag*. Klostermann, Seite 57-69, 9/1999

[Http99] IETF. Hypertext Transfer Protocoll (HTTP) Working Group. URL: http://www.ics.uci.edu/pub/ietf/http/. 199?.

[HYS88] G. Hudson, H. Yasuda und I, Sebestyen. The International Standardization of a Still Picture Compression Technique. In Proc. *IEEE Global Telecommunications Conference*, S. 1016-1021. 11/1988.

[ISO93] ISO IEC JTC 1. Information technology - coding of moving pictures and associated audio for digital storage media, test model 4, MPEG 93/255b. 2/1993

[ISO96] ISO/IEC 10179:1996. Information technology -- Processing languages -- Document Sytle Semantics and Specification Language (DSSSL). URL: http://occam.sjf.novell.com:880/dsssl/dsssl96. 1996

[JLR98] Mouriel Jourdan, Nabil Layaida, Cecile Roisin, L. Sabry-ismail, und Laurant Tardif. Madeus, An Authoring Environment for Interactive Multimedia Documents. In Proc. *6. ACM international conference on Multimedia*, Bristol, UK, S. 267-272, 9/1998.

[JRT00] Mouriel Jourdan, Cecile Roisin, und Laurant Tardif. A scalable Toolkit for Designging Multimedia Authoring Environments. In *Multimedia Authoring and Presentation: Strategies, Tools and Experiences: Multimedia Tools and Applications Journal*. Kluwer Academic Publishers. 2/3(12):257-279, 11/2000

[KFB99] Yong-Moo Kwon, Elena Ferrari und Elisa Bertino. Modeling spatio-temporal constraints for multimeida objects. In *Elsevier Data & Knowledge Engineering,* 30(1999):217-238, 1999

[Klo00] Martin Kloss. Lingo objektorientiert. Director optimiert einsetzen. Galileo Press, 2000

[LiG90] T.D.C. Little und A. Ghafor. Synchronization and Storage Models for Multimedia Objects. In *IEEE Journal on Selected Areas in Communication*, 8(3):413-427, 1990.

[LiS99] Hartmut Liefke und Dan Suciu. XMill: an Efficient Compressor for XML Data. Technischer Bericht MS-CIS-99-26, At&t Research. 1999.

[May90] B.A. Mayers. “Taxonomies of Visual Programming and Program Visualization”. Journal Visual Languages and Computing, 1(1):97-123, 1990.

[Mheg95] ISO/IEC DIS 13522-5. Information Technology - Coding of Multimedia and Hypermedia Information - Part 5. 1995.

[Mheg97] ISO/IEC DIS 13522-6. Information Technology - Coding of Multimedia and Hypermedia Information - Part. 1997.

[NyM00] John R. Nyquist und Robert Martin. Director 8 Lingo Bible. Hungry Minds Inc, 11/ 2000.

[PAP95] D. Papadias, Y. Theodoridis. Spatial relations, Minimum Bounding Rectangles, and Spatial Data Structures. In *International Journal of Geographic Information Systems*, 1995.

[PeL96] Maria Jose Perez-Luque and Thomas D. C. Little. A Temporal Reference Framework for Multimedia Synchronization. In *IEEE Journal on Selected Areas in Communication*, 14(1), 1/1996.

[Pem+00] Steven Pemberton et al. XHTML 1.0 The Extensible Hyper Text Markup Language. W3C Recommendation. URL: http://www.w3.org/TR/xhtml1 . 1/2000.

[Png95] Internet Engineering Task Force (IETF). PNG (Portable Network Graphics) Specifiation Version 1.0. URL: http://www.ietf.org/rfc/rfc2083.txt. 1995.

[Qua98] Terry Quatrani. Visual Modelling with Rational Rose and UML. Addison Wesley, 1998

[RHJ99] Dave Ragget, Arnaud Le Hors und Ian Jacobs. HTML 4.0 Specification. W3C Recommendation. URL: http//www.w3.org/TR/html401. 1999.

[RJM+93] Guido van Rossum, Jack Jansen, K. Sjoerd Mullender and Dick C.A. Bulterman. CMIFed: A Presentation Environment for Portable Hypermedia Documents. In *Proc. ACM Multimedia*, S. 183-188. 1993.

[SDL+96] P. Senac, Michel Diaz, Alain Leger, and Pierre de Saqui-Sannes. Modeling Logical and Temporal Synchronization in Hypermedia Systems. In *IEEE Journal on Selected Areas in Communications*, 14(1):84-104, 1996.

[Shn83] Ben Shneiderman. “Direct Manipulation: A Step Beyond Programming Languages”, In Computer, Seite 57-69, 8/1983.

[SKD96] J. Schnepf, J. A. Konstan, and D. H.-C. Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. *In IEEE Jounal on Selected Areas in Communications*, 14(1):114-125, 1996.

[Ste99] Ralf Steinmetz. Multimedia Technologie: Grundlagen, Komponenten und Systeme. 2. Auflage, Springer Verlag. 1999.

[Sun97] Sun Microsystems. JavaBeans Specification 1.01. URL: http://java.sun.com/beans. 1997.

[Sun98] Sun Microsystems. Reflection. http://java.sun.com/products/jdk/1.1/docs/guide/reflection/index.html. 1998.

[Sun01a] Sun Microsystems. Java API for XML Processing. URL: “http://java.sun.com/xml/xml_jaxp.html”. 2001.

[Sun01b] Sun Microsystems. Java Architecture for XML Binding (JAXB v0.21). http://java.sun.com/xml/jaxb/index.htm, 9/2001.

[Sun01c] Sun Microsystems. Java Media Framework. URL: http://java.sun.com/products/java-media/jmf/index.html. 2001.

[V3D2] Deutsche Forschungsgemeinschaft (DFG). Verteilte Verarbeitung und Vermittlung Digitaler Dokumente. URL: http://www.cg.cs.tu-bs.de/dfgspp.VVVDD.

[VaS96] M. Vazirgiannis und T. Sellis. Event and Action Representation and Composition for Multimedia Application Scenario Modeling. In Proc. *ERCIM Workshop on Interactive Distributed Multimedia Systems and Services*. 3/1996.

[W3C] W3C - The World Wide Web Consortium. URL: http://www.w3c.org/

[WaR94] Thomas Wahl and Kurt Rothermel. Representing Time in Multimedia Systems. In *Proc. of IEEE 1st Intl. Conference on Multimedia Computing and Systems, Boston, USA*, S. 538–543, 1994.

[WeD99] Sanjiva Weerawarana und Mathew J. Duftler. Bean Markup Language (Version 2.3) User’s Guide, IBM TJ Watson Research Center, 9/1999.

[Wir97] Stefan Wirag. Modeling of Adaptable Multimedia Documents. In Proc. *Interactive Distributed Multimedia Systems and Telecommunication Services;* 4th International Workshop, IDMS'97, S. 220- 230, Darmstadt, Germany, September 1997.

[Wir99a] Stefan Wirag. Specification and Scheduling of Adaptive Multimedia Documents, Fakultätsbericht 1999/04, Universität Stuttgart, 1/1999.

[Wir99b] Stefan Wirag. Adaptive Scheduling of Multimedia Documents. In Proc. *Kommunikation in Verteilten Systemen (KiVS)*. 3/1999.

[Wir02] Stefan Wirag. Modellierung und Ablaufplanung adaptiver Multimedia-Dokumente. Dissertation. Universität Stuttgart. 2002.

[WWR95] Thomas Wahl, Stefan Wirag and Kurt Rothermel. TIEMPO: Temporal Modeling and Authoring of Interactive Multimedia. In *Proc. of IEEE 2nd Intl. Conference on Multimedia Computing and Systems, Washington DC*, S. 274-277, 5 1995