Integration eines kontextsensitiven Interaktionsmodells in ein virtuelles, multimediales Schulungssystem

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom-Wirtschaftsinformatiker (FH)
an der Fachhochschule für Technik und Wirtschaft Berlin im
Studiengang Wirtschaftsinformatik
des Fachbereichs Wirtschaftswissenschaften II

Erstellt von: Torsten Rack im SS 2001
Betreuer: Prof. G. Bannert
Dr. Th. Schmidt

Berlin, 26. Oktober 2001

Inhaltsverzeichnis

1. Schulungssysteme

1.1. Einleitung

Im Laufe der letzten Jahre ist die Lehre in einer nie zuvor dagewesenen Weise revolutioniert worden. Durch immer leistungsstärkere Hardware und Software ist es heute möglich und darüber hinaus sogar selbstverständlich geworden, die bisher in konventionellen Formen der Lehre eingesetzten Medien elektronisch zu verarbeiten. Solche Medien können beispielsweise Texte, Bilder, Filmmaterial oder Tonaufzeichnungen sein.

Die Verarbeitung von multimedialen Daten erfolgt in elektronischer Form inzwischen sogar häufig viel effizienter als auf herkömmlichem Wege. Dazu beigetragen hat beispielsweise die leichte Wiederverwendbarkeit von Lehrmaterialien. Hinzu kommt, daß heutige Software mit ihren modernen und komfortablen graphischen Benutzerschnittstellen (GUI[1]) den Autor bei der Spezifikation der Lehrmaterialien in vielfältiger Weise unterstützen kann. So können viele Vorgänge mit repetitivem Charakter automatisiert werden, wodurch sich deren Bearbeitungsaufwand auf ein Minimum reduzieren läßt.

[1] Graphical User Interface

Durch die Verwendung von elektronischen Medien entstehen Möglichkeiten, die sich bei herkömmlichen Medien in dieser Qualität nicht erzielen ließen. Ebenso wäre der Aufwand, den ein Lehrer betreiben müßte, wenn er seinen Schülern mit konventionellen, nicht-elektronischen Methoden einen multimedialen Lehrablauf, bestehend aus Bild, Ton, Schrift und Videomaterial, präsentieren möchte, bei konventionellen Systemen meistens höher. Er würde dazu als Hilfsmittel wahrscheinlich einen Overhead-Projektor, ein Tonbandgerät, einen Fernseher mit Videorekorder und eine Tafel benötigen. Ebenso wäre der Lehrer selbst für eine entsprechende Synchronisation und zeitliche Steuerung der einzelnen Medien verantwortlich. Alle diese Geräte können in der elektronischen Lehre durch einen einzigen Computer ersetzt werden. Die räumliche und zeitliche Synchronisation zwischen den Medien übernimmt dann der Computer, der dem Schulungsteilnehmer von Autoren erstellte Lerneinheiten interaktiv präsentieren kann.

Zudem wird ein komplexer interaktiver Lehrablauf, an dem viele verschiedenartige Medientypen beteiligt sind, erst durch den Einsatz von Software in der Praxis ermöglicht. Schließlich kann ein gedruckter Text oder ein gemaltes Bild als Printmedium in technischer Hinsicht kaum sinnvolle interaktive Fähigkeiten bieten, da Printmedien zumeist linear rezeptiv sind. In konventionellen Formen der Lehre sind nicht lineare, interaktive und multimediale Lehrabläufe lediglich durch räumliche Präsenz und entsprechend koordiniertes Handeln des Lehrers zu realisieren.

In der elektronischen Lehre ist es bei den meisten in der Praxis existierenden Systemen die Regel, daß ein Schüler eine Schulung individuell und unabhängig von den anderen Kursteilnehmern am Computer verfolgt. Er kann daher den Präsentationsverlauf in für sein persönliches Profil optimaler Weise anpassen, wodurch sich der Wirkungsgrad der Schulung verbessern kann. So ist es ein großer Vorteil, daß der Schüler theoretisch beispielsweise die Ablaufgeschwindigkeit einer Schulung auf die für ihn individuell beste Geschwindigkeit einstellen kann. Die Präsenz einer Lehrperson während des Ablaufs einer interaktiven Lerneinheit ist hier nicht nötig.

Ein sehr bedeutsamer Faktor in der elektronischen Lehre ist auch das Internet. Es ermöglicht einerseits die Nutzung angebotener elektronischer Lehrinhalte unabhängig von Ort und Entfernung von sowohl Lehrer als auch Schüler. Andererseits bringt es auch in der Erstellung von Lerneinheiten durch die Möglichkeit des kooperativen und verteilten Bearbeitens der Inhalte große Vorteile. Auch hier werden räumliche Aspekte, wie die geographische Entfernung zweier Lehrer, durch das Internet zu einem vergleichsweise unbedeutenden Faktor.

Es ist leicht zu erkennen, daß die Vorteile der elektronischen Lehre in ihrer Summe groß sind. Daher hat in den letzten Jahren, sicher auch durch die im folgenden Abschnitt beschriebenen wirtschaftlichen Faktoren, die Forschungsaktivität auf diesem Gebiet stark zugenommen. Der wichtige Themenkomplex der Interaktion ist jedoch erst in jüngster Zeit stärker fokussiert wor-

den. Daher besteht in diesem Sektor noch ein ausreichend großes Potential für Verbesserungen und Innovationen.

1.2. Schulungssysteme als wirtschaftlicher Faktor

Pauschal können zunächst alle in der Einleitung genannten Vorteile direkt oder indirekt zu nicht unerheblichen Kostenersparnissen führen. Die gesamte Thematik ist also durchaus auch unter wirtschaftlichen Gesichtspunkten interessant, wenn auch in Unternehmen eher von Schulungen als von Lehre gesprochen wird. Dieses ist jedoch aus Sicht eines Schulungssystems nur ein Aspekt der Begriffswahl, da sowohl Lehre als auch Schulung im eigentlichen Wortsinn dort die selben Mechanismen und Fähigkeiten nutzen. In diesem Abschnitt sollen daher die einzelnen Aspekte unter wirtschaftlichen Gesichtspunkten untersucht werden.

1.2.1. Räumliche Aspekte

Ein wesentlicher Kostenfaktor bei der Durchführung von Schulungen sind die räumlichen Bedingungen. Schulungen werden außer vielleicht bei sehr großen Unternehmen mit eigener Schulungsabteilung im Regelfall extern durchgeführt. Die zu schulenden Mitarbeiter müssen daher meist weite Anreisewege in Kauf nehmen. Das ist zeitaufwendig und kostenintensiv. Zudem entsteht durch die zeitintensive Anreise ein längerer und häufig vermeidbarer Arbeitskraft- und damit unter Umständen Know-How-Ausfall der betroffenen Mitarbeiter.

Auch für die Anbieter von Schulungen spielt der räumliche Aspekt eine große Rolle. Sie müssen bei konventionellen Schulungen entsprechend ausreichend dimensionierte Räumlichkeiten bereithalten, wodurch hohe Mietaufwendungen entstehen können.

1.2.2. Förderung der Angebotsvielfalt

Dieser Punkt ist in gewisser Weise verwandt mit den räumlichen Aspekten. Durch Wegfall des Raumangebots auf Anbieterseite kann die Vielfalt der gleichzeitig angebotenen Schulungen maximiert werden. In einem Schulungsraum können schließlich nicht zeitgleich mehrere Schulungen stattfinden, ohne daß sich diese gegenseitig störend beeinflussen würden. Durch Wegfall der räumlichen Faktoren kann natürlich theoretisch die durch die verwendete Technologie beschränkte maximale Anzahl an Schulungen gleichzeitig zum Abrufen angeboten werden.

1.2.3. Personaleinsparungen auf Anbieterseite

Auf Anbieterseite ist es ferner möglich, in gewissem Umfang Personal einzusparen. Außer, daß man durch den Wegfall der Schulungen vor Ort Lehrpersonal einsparen kann, besteht auch in anderer Hinsicht das Potential, mit geringerem Personalbestand fahren zu können. Mußte der Autor einer Schulung diese nach dem Erstellen unzählige Male abhalten, kann er nun nach dem Erstellungsprozeß direkt weitere Lehrangebote erstellen. Die Schulungsarbeit übernimmt letztendlich das Softwaresystem für ihn. Damit wandelt sich die Rolle des Lehrers zu der eines Autors von Lerneinheiten. Ein einzelner Autor kann innerhalb relativ kurzer Zeit viele Lehrmaterialien erstellen, wodurch mit verhältnismäßig geringem Personalstand eine hohe Zahl bereitgestellter Präsentationen denkbar ist.

1.2.4. Infrastruktur

Die elektronische Verarbeitung von Lehrmaterialien erfordert natürlich eine entsprechend dafür geeignete IT-Infrastruktur im betreffenden Unternehmen. Es kann jedoch davon ausgegangen werden, daß mittlerweile zumindest in Büros jeder in Frage kommende Arbeitsplatz mit einem Computer ausgestattet ist. Diese sind mitunter zwar nicht zur Wiedergabe multimedialer Inhalte, speziell Video- und Tonmaterial geeignet, können aber mit recht geringem Kostenaufwand umgerüstet werden. Ebenso verfügt die große Mehrheit von Unternehmen über eine für diesen Anwendungsfall ausreichend dimensionierte Internetanbindung, die auch von den Arbeitsplatzcomputern aus nutzbar wäre.

Zusammenfassend läßt sich erkennen, daß der kostenmäßige Aufwand für die Einführung elektronischer Schulungssysteme, verglichen mit dem aus ihnen resultierenden Nutzen, pauschal gesehen nicht allzu hoch ist. Wie weit es für ein bestimmtes Unternehmen attraktiv ist, elektronische Schulungen einzuführen, muß natürlich im Einzelfall genauestens errechnet werden. Es ist zu erwarten, daß die elektronischen Schulungen in den nächsten Jahren in Unternehmen einen noch größeren Stellenwert erlangen werden, als bisher bereits geschehen.

2. Verwandte Ansätze

2.1. Dexter-Modell

Das Dexter-Modell[1] ist ein Referenzmodell für Hypermedia-Systeme. Es dient als Bewertungsmaßstab beim Vergleich verschiedener Hypermedia-Systeme sowie als Grundlage für die Entwicklung von Standards für den Datenaustausch und die Interoperabilität zwischen solchen Systemen. Ferner definiert das Modell standardisierte Bezeichnungen für die im Modell existierenden Mechanismen und Entitäten.

Das Modell sieht in einem Hypermedia-System eine Abgrenzung von drei Schichten (Layers) vor:

[1] siehe [3]

Storage Layer Der Storage Layer beschreibt eine Datenbank, die mit relationalen Links verknüpfte Komponenten enthält. Komponenten im Sinne des Modells sind die Knoten dieser Strukturen. Im Vordergrund stehen dabei die Mechanismen der Verlinkung von Komponenten.

Within-Component Layer Der Within-Component Layer beschäftigt sich mit dem Inhalt und der inneren Struktur von Komponenten. Das Modell macht in dieser Schicht jedoch bewußt keine klaren Vorgaben, sondern delegiert dieses an andere, auf die Strukturierung spezialisierte Referenzmodelle.

Runtime Layer Der Runtime Layer beschreibt die Laufzeitschicht eines Systems. Er ist verantwortlich für die Instanziierung und Präsentation von Komponenten.

Besonders zu erwähnen ist die Lösung der Verlinkung auch zu Zielen innerhalb des Inhalts oder der Struktur der Komponenten. Dazu wird ein Mechanismus benötigt, der auch eine Adressierung von Orten innerhalb einer Komponente unterstützt. Dieses wird im Dexter-Modell mit sogenannten **Ankern** realisiert. Anker im Sinne des Modells entsprechen Bezeichnern, die eine Stelle innerhalt der Komponente beschreiben können, wie etwa eine Position innerhalb eines Textes. Durch die Verwendung von Ankern bleibt die klare Trennung zwischen Within-Component und Storage Layer erhalten.

2.2. SMIL

SMIL[2] - Synchronized Multimedia Integration Language - ist eine Empfehlung des WWW Consortiums[3] (W3C Recommendation) für eine Beschreibungssprache für multimediale Präsentationen. Es ist dabei aber keine eigenständige Sprache im wörtlichen Sinne, sondern als XML-Anwendung realisiert.

SMIL unterstützt die Medientypen Audio, Video, Text und Bild.

2.2.1. Grundgedanken

SMIL ist als universelles und platformunabhängiges Präsentationsformat gedacht, welches sich auch und ganz besonders für den Einsatz im World Wide Web eignet. Alle Elemente einer Präsentation werden daher über einen Unified Resource Locator (URL) identifiziert.

[2] siehe [15, zuletzt besucht: 16.10.2001]
[3] siehe [14, zuletzt besucht: 16.10.2001]

Die Zeitsynchronisation bei SMIL erfolgt nicht durch Angabe absoluter Zeitpunkte, sondern durch Angaben relativ zu anderen Präsentationselementen.

Ein Vorteil von SMIL ist es, daß nicht zwingend ein spezielles Autorenwerkzeug benötigt wird. Da die Beschreibungssprache auf XML basiert, reicht zur Bearbeitung von Präsentationen theoretisch ein einfacher Texteditor aus.

2.2.2. Version 2

Im Jahre 2001 wurde SMIL 2.0 als Empfehlung vom W3 Consortium herausgegeben. Die neue Version bringt zunächst eine struktuelle Erweiterung des Systems mit sich. SMIL 2.0 setzt auf einer Modulstruktur auf. Es existieren Module beispielsweise zur Unterstützung von Layout, Animationen oder Verlinkung.

Des weiteren wurde ein Mechanismus zur ereignisgesteuerten Aktivierung (Event-based Activation) eingeführt, der neue Möglichkeiten der Interaktion schafft.

2.2.3. Nachteile

Den vielen positiven Eigenschaften von SMIL stehen jedoch auch einige negative gegenüber. Gravierend ist zum Beispiel die eingeschränkte Modularität der Präsentationen. Eine einfache Wiederverwendung kompletter Teilbäume einer Präsentation ist nicht ohne weiteres möglich. Es befinden sich nämlich sämtliche Strukturinformationen einer Präsentation innerhalb einer einzigen XML-Datei. Eine externe Wiederverwendung einzelner Elemente auf der Medienobjekt-Ebene ist deshalb nicht möglich. Lediglich die extern referenzierten reinen Daten können problemlos wiederverwendet werden.

SMIL Präsentationen benötigen, obwohl für das World Wide Web gedacht, momentan noch ein separates Abspielprogramm bzw. ein Browserplugin. Eine direkte Unterstützung durch die Browser gibt es bei den meisten Produkten nicht.

2.2.4. Beispiel

Ein kleines Beispiel soll die Spezifizierung einer relativ einfachen Interaktion in SMIL 2.0 demonstrieren (Quelle: W3C, http://www.w3.org/TR/smil20/smil-timing.html). Im Beispielszenario sollen drei (an anderer Stelle definierte) Knöpfe bei Betätigung ein Video mit einer dem jeweiligen Knopf zugeordneten Tonspur starten. Der Start der Bild- und Tonspur soll synchron erfolgen.

Listing: SMIL Beispiel

```
<smil ...>
...
<par>
   <video id="vid1" .../>
   <excl>
      <par begin="englishBtn.activateEvent" >
         <audio begin="vid1.begin" src="english.au" />
      </par>
      <par begin="frenchBtn.activateEvent" >
         <audio begin="vid1.begin" src="french.au" />
      </par>
      <par begin="swahiliBtn.activateEvent" >
         <audio begin="vid1.begin" src="swahili.au" />
      </par>
   </excl>
</par>
...
</smil>
```

2.3. HTML+TIME

HTML+TIME[4] ist eine Erweiterung von HTML um die Fähigkeiten zur Zeitsteuerung und Synchronisation. Ziel von HTML+TIME ist es, die SMIL[5] Konzepte in HTML, einen weit verbreiteten und weitläufig unterstützten Standard, zu integrieren. Als Grundlage dafür diente die SMIL Version 1.0.

Zur Integration der Zeitsteuerung in HTML sind beinahe ausschließlich bestehende HTML-Tags verwendet worden, die lediglich mit zusätzlichen Attributen, wie z.B. Zeit oder Dauer, dekoriert wurden. Ein großer Vorteil von HTML+TIME ist daher die für HTML-Autoren sehr leichte Erlernbarkeit.

HTML+TIME besteht im Kern aus einem Objektmodell, welches das Modell von DHTML erweitert. Komplexere Abläufe laßen sich somit über eingebettete Script-Bereiche (z.B. JavaScript) realisieren.

[4] TIME=Timed Interactive Multimedia Extension
[5] siehe [15, -]

Auch hier gilt natürlich, daß die Bearbeitung einer Präsentation keinen speziellen Editor benötigt, sondern mit einem einfachen Texteditor erfolgen kann.

HTML+TIME unterstützt die Gruppierung von Elementen. Es ist dann möglich, Operationen auf eine ganze Gruppe von Elementen anzuwenden, z.B. die Elementen synchron anzuzeigen.

2.3.1. Zeitsteuerung und Interaktion

Es existieren parallel zwei Modelle für die Zeitsteuerung in HTML+TIME.

- Zeitleiste - Alle Elemente sind an einer Zeitleiste angeordnet.
- Ereignissteuerung - Es werden Events und Event-Listener verwendet. Als Event-Typen kommen DOM-Events[6], wie z.B. element.onClick(), und Timer-Events zum Einsatz. So ist es möglich, Interaktivität in einer HTML+TIME-Präsentation anzubieten.

2.3.2. Beispiel

Zur Erläuterung soll ein kleines Beispiel dienen. Es wird eine 300 mal 200 Bildpunkte große Gruppe gebildet. In dieser Gruppe befindet sich eine Slideshow aus drei Bildern, die sequenziell jeweils 5 Sekunden angezeigt werden sollen. Ein Mausklick soll jeweils das nächste Bild visualisieren.

Die Slideshow selbst kann durch den Klick auf den Start-Link gestartet werden.

Der dazu notwendige Quelltext könnte folgendermaßen aussehen (Quelle: W3 Consortium, http://www.w3c.org/TR/NOTE-HTMLplusTIME):

```
1  <div height=200 width=300>
2
3      <t:seq id="SLIDESHOW" t:beginEvent="none">
4          <!-- This begins when a trigger is sent.
5               This will sequence the three images.
6               If the user clicks on an image before the assigned
7               duration, it will advance to the next image.
8          -->
9          <img src="image1.gif" t:dur="5" t:timeAction="display"
10              onclick="this.endElement()">
11
```

[6] DOM=Document Object Model, siehe http://www.w3c.org/DOM/

```
12          <img src="image2.gif" t:dur="5" t:timeAction="display"
13               onclick="this.endElement()">
14
15          <!-- This uses endEvent syntax. Note that 'dur'
16               will override if there is no click by 5 seconds -->
17
18          <img src="image3.gif" t:dur="5" t:timeAction="display"
19               t:endEvent="onclick">
20
21      </t:seq>
22
23
24      <img src="showOver.gif" t:beginAfter="SLIDESHOW"
25               t:timeAction="display">
26
27
28      <p align=center onclick="SLIDESHOW.beginElement()">
29          Click here to begin the slideshow.
30      </p>
31
32      <p align=center>
33          If it advances too slowly for you, just click on an image
34          to advance it interactively.
35      </p>
36
37  </div>
```

Dieser Ansatz scheint sich jedoch nicht durchzusetzen. Es existiert kein Paper zum Standard sondern lediglich ein aus dem Jahre 1998 stammendes W3C-Paper mit dem Status „Note“.

2.4. Nested Context Model

Das Nested Context Model (NCM)[7] stellt einen auf dem Dexter-Modell (siehe 2.1, Seite 10) basierenden Ansatz dar, der an dieser Stelle einführend beschrieben werden soll.

Die Superklasse beim NCM heißt *Entity* (Entität). Sie besitzt einen eindeutigen Schlüssel, eine Zugriffsliste (Access Control List, ACL) und einen „entity descriptor“, der beschreibt, wie die Präsentation erfolgen soll.

[7] siehe [12, Kapitel 2]

Abgeleitet von *Entity* ist *Node* (Knoten), welcher einen Content (Inhalt) sowie eine Menge von Ankern für das Objekt enthält.

Die Anker sind gebunden an Paare aus Konditionen/Aktionen, die die Gültigkeit des Ankers beeinflußen.

Es werden zwei Arten von Knoten unterschieden:

- **Terminalknoten („Terminal Node“)**

  Terminalknoten sind Knoten, die applikationsspezifischen Inhalt und Anker aufweisen. Die Anker sind dabei Verweisziele (Sprungmarken), die eine Adressierung auch innerhalb des Knotens erlauben.

- **Kompositionsknoten („Composition Node“)**

  Kompositionsknoten enthalten eine Liste von referenzierten Entitäten. Mit ihnen kann eine Menge von Entitäten strukturiert werden.

2.5. Madeus

Madeus ist eine Autoren- und Präsentationsumgebung. Es werden verschiedene Bildformate, Text und MPEG Streams (Audio und Video) unterstützt. Die aktuelle Implementierung ist in der Sprache C erfolgt, so daß eine wirkliche Tauglichkeit für das World Wide Web aufgrund der fehlenden Plattformunabhängigkeit nicht unbedingt gegeben ist.

Madeus basiert auf einem objektorientierten Datenmodell und benutzt eine hierarchische Strukturierung für die Medienobjekte einer Präsentation. Es verfolgt bei der Spezifizierung der Präsentations-Szenerien einen constraint-basierten Ansatz. D.h. einzelne Medienobjekte werden nicht absolut, sondern relativ zueinander in Beziehung gesetzt. Ein Beispiel dafür wäre „Spiele Video2, wenn Video1 durchgelaufen ist!“.

3. Betrachtung des Problems

3.1. Der Begriff der Interaktion

Ziel dieser Diplomarbeit ist es, ein Interaktionsmodell in ein Schulungssystem zu integrieren. Daher soll zunächst einmal der Begriff der Interaktion beziehungsweise der Interaktivität definiert werden:

„Interaktion bedeutet, daß der Nutzer aktiv die Abarbeitung eines Programmes beeinflussen kann.“[1]

Im multimedialen Kontext ergeben sich nach *Soares et al.*[2] eine Reihe von Vorraussetzungen, die ein multimediales Autoren-/Präsentationssystem im Idealfall erfüllen sollte. Im folgenden sollen einige der grundlegenden sowie einige besonders im Hinblick auf Interaktivität wichtige Aspekte dieser Anforderungen diskutiert werden.

3.1.1. Strukturierung der Dokumente

Es wird gefordert, daß das System die Spezifikation semantischer und nicht-semantischer Objektstrukturen erlaubt. Als strukturelle Grundlage können beispielsweise räumliche und zeitliche Merkmale dienen.

3.1.2. Trennung von Struktur und Daten

Eine wichtige Anforderung ist die Trennung von Struktur beziehungsweise Logik der Medienobjekte und ihren eigentlichen Mediendaten. Zum besseren Verständnis dieser Anforderung seien hier die Termini Medienobjekt und Mediendaten, wie sie bei Soares et al.[3] Verwendung finden, erklärt. Ein *Medienobjekt* bildet dabei eine logische Einheit,

- der Daten zugeordnet sind,
- die in einem Strukturellen Kontext befindlich ist
- und möglicherweise das Medium oder Medienobjekt parameterisierende Attribute aufweist.

Die dem Medienobjekt zugeordneten Mediendaten sind dabei wirklich die binären Daten, wie beispielsweise ein Videostream oder eine Bilddatei.

Die binären Daten von den Strukturdaten zu trennen, bietet den Vorteil, sie in mehreren, eventuell sogar erheblich unterschiedlich strukturierten Medienobjekten verwenden zu können, ohne die Daten redundant zu speichern.

[1] siehe [2, zuletzt besucht: 16.10.2001]
[2] siehe [11, Seite 71]
[3] siehe [11, 71ff]

3.1.3. Wiederverwendbarkeit

Wie im vorigen Abschnitt bereits angesprochen, ist die Wiederverwendung von Mediendaten mitunter durchaus sinnvoll. Dieses trifft auch auf Medienobjekte zu. So können ganze Blöcke einer Präsentation, wenn einmal vorhanden, beliebig oft durch Referenzierung in andere Präsentationen eingebettet werden.

3.1.4. Komplexe Objektrelationen

Aus Sicht der Interaktion ist die Fähigkeit zu komplexen Objektbeziehungen eine fundamental wichtige Eigenschaft. Im eigentlichen Sinne werden dabei genau ein Quellobjekt und ein Zielobjekt in eine Relation gesetzt. Diese Relation zwischen den Objekten kann zeitlicher und räumlicher Natur sein. Der einfachste Fall der Relation kann zum Beispiel eine räumliche „Enthaltensein“-Beziehung darstellen.

Häufig ist es jedoch nötig, Relationen mit mehr als zwei beteiligten Komponenten zu definieren. Man unterscheidet dann zwischen Bedingungsrelationen (Constraints), wie beispielsweise „Bild 1 und Bild 2 gleich lange anzeigen!“, und kausalen Relationen, die von einer Menge von Konditionen bedingt sind. Im Falle der Interaktion sind die kausalen Relationen von besonderem Interesse. Sie ermöglichen eine konditionierte Ausführung von Aktionen auf in der Relation referenzierte Objekte. Ein Beispiel wäre „Wenn Mausklick erfolgt und Bild Nr. 3 angezeigt ist, dann zeige Bild Nr. 4!“.

3.2. Zeitbasierte Abläufe

Zeitbasierte Abläufe weisen als zentralen und bestimmenden Parameter die Zeit auf. Zur Verdeutlichung kann sich der Leser dabei einen virtuellen Zeitstrahl vorstellen. An diesem Zeitstrahl sind nun zu bestimmten Zeitpunkten Vorgänge mit einer bestimmten oder unbestimmten Dauer angeordnet. Läßt man nun einen Zeitgeber sequentiell über diesen Zeitstrahl laufen, so finden zu jedem Zeitpunkt genau die Vorgänge statt, die an der jeweiligen Zeitgeberposition entlang des Strahls angeordnet sind.

Es kann grundsätzlich zwischen zwei Typen von zeitbasierten Abläufen unterschieden werden.

Sequentielle Abläufe

Die einzelnen Vorgänge sind überschneidungsfrei nebeneinander angeordnet. So ist niemals mehr als ein Vorgang gleichzeitig aktiv.

Parallele Abläufe

Bei parallelen Abläufen können an einer Position des Zeitstrahls mehrere gleichzeitige Vorgänge vorhanden sein. Diese Vorgänge sind dann zur gleichen Zeit parallel aktiv.

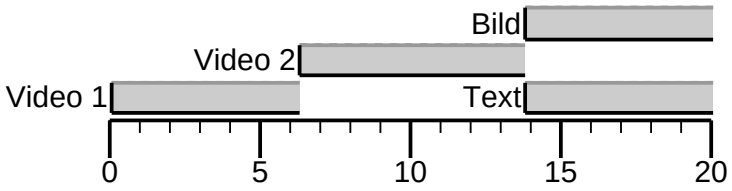

Abbildung 3.1.: Zeitbasierter Ablauf

3.3. Nichtlineare Abläufe

Wie der vorangegangene Abschnitt gezeigt hat, bieten zeitbasierte Abläufe eine Möglichkeit, einzelne Vorgänge untereinander zu synchronisieren. Für die Abarbeitung dieser Abläufe wurde dort zunächst ein streng linearer Verlauf angenommen. Bei der Abbildung zeitbasierter Abläufe jedoch wird man in einem System, in dem die Abläufe zwingend linear verlaufen müssen, recht schnell an die Grenzen stoßen. Interaktive Abläufe ließen sich mit solch einem System nicht abbilden, da Interaktivität beinahe immer zu einem Verlassen der zeitlichen oder strukturellen Linearität führt. Innerhalb einer Präsentation kann zum Beispiel ein Mausklick auf einen Knopf den linearen Verlauf unterbrechen, indem er ein Springen zu einer völlig anderen Stelle der Präsentation bewirkt.

Häufig sind nichtlineare Abläufe nicht vorhersagbar, da der Weg eines Ablaufes durch vielfältige, teils unbestimmte Faktoren, beispielsweise interaktive Einflußnahme durch den Benutzer, beeinflußt werden kann.

3.4. Kontextsensitivität

Wie für nichtlineare Abläufe festgestellt wurde, sind diese meistens nicht vorhersagbar. Dieser Umstand läßt sich aus der hohen Abhängigkeit des Verlaufs vom jeweiligen Kontext ableiten. Gerade Interaktionen sind meistens kontextabhängig und sind daher nicht jederzeit global möglich. Zur Vedeutlichung soll hier ein Beispiel dienen, welches in einer Präsentation durchaus denkbar wäre:

Ein Ablauf setzt sich aus zwei parallel verlaufenden, sequentiellen Abfolgen von Bildern zusammen. Nach jeweils einer definierten Zeit werden die nächsten Bilder der Sequenz angezeigt. Die Szenerie soll Interaktion mittels durch den Benutzer durchgeführten Mausklicks erlauben. Ein Klick in eines der Bilder soll eine zu dem angeklickten Bild passende Erklärung anzeigen. Dadurch liegen kontextuelle Abhängigkeiten zeitlicher und räumlicher Art vor.

Es lassen sich verschiedene generische Arten der kontextuellen Abhängigkeit formulieren:

- räumlich
- zeitlich
- (Nicht-)Vorhandensein bestimmter Objekte
- Zustand bestimmter Objekte

Dies sei hier einmal an einem Beispiel erläutert: Bei einem nicht kontextsensitiven System müssten im Falle einer interaktiven Präsentation vom Autor zur Spezifikationszeit alle in Frage kommenden Präsentationsverläufe explizit spezifiziert werden. Es wird exemplarisch angenommen, daß im Präsentationsverlauf an mehreren Stellen das gleiche Objekt präsentiert werden soll, welches auf Mausklicks eine Information zum aktuellen Präsentationsverlauf einblendet. Ohne Kontextsensitivität müsste das Objekt für jede dieser Stellen explizit und redundant spezifiziert werden, da eine dynamische Anpassung an den momentanen Zustand der Präsentation nicht möglich ist.

Es erwächst die Forderung an ein multimediales Präsentationssystem, kontextsensitiv zu arbeiten. Das System muß in der Lage sein, Interaktionen auf Grundlage mindestens der vier oben genannten Aspekte konditionieren zu können.

3.5. Problemfall Authoring

Der Autorenumgebung eines Präsentationssystemes gilt besonderes Augenmerk. Ihre Qualität beeinflußt den praktischen Wert eines Präsentationssystems mitunter erheblich. Autorensysteme können dabei nach Jourdan, Layaïda und Roisin [4] bezüglich zweier wesentlicher Merkmale beurteilt werden:

- **Flexibilität bei Erstellung („Authoring Capabilities“)**

[4] siehe [4, 3.+4.]

Die „Authoring Capabilities“ beschreiben den Grad der Spezifizierbarkeit von Szenarios. Im Idealfall, in einem System mit den angenommen größten Authoring Capabilities, ist es möglich, jede erdenkliche Szenerie in dem System abzubilden.

- **Ausdrucksmöglichkeiten („Expressive Power“)**

  Die Ausdrucksmöglichkeiten beschreiben das Potential eines Systems, dem Autor zu ermöglichen, jede von ihm gewünschte Szenerie mit dem System auf intuitive Weise und mit angemessenem Verhältnis zwischen Aufwand und Resultat spezifizieren zu können.

Ebenso ist eine Kategorisierung nach dem gewählten Autorenansatz möglich. Man unterscheidet zwischen zwei üblichen Ansätzen:

- **Operationale Systeme („Operational Systems“)**

  Bei operationalen Systemen werden Objekte und deren Beziehungen meist absolut spezifiziert, zum Beispiel „Zeige bei Zeiteinheit 7 für 3 Zeiteinheiten das Bild Nr. 3!“.

- **Beschränkungsorientierte Systeme („Constraint-based Systems“)**

  Bei beschränkungsorientierten Systemen ist eine absolute Spezifikation der Objektabhängigkeiten nicht zwingend notwendig. Synchronisation kann durch Auferlegung von Beschränkungen erreicht werden. Sollen beispielsweise zwei Videos synchronisiert werden, so daß sie sequentiell ablaufen, könnte man dieses durch Angabe relativer Beziehungen der beiden Videos zueinander spezifizieren. Beispiel: „Spiele Video 1 bei Zeiteinheit 3!“ und „Spiele Video 2 bei Ende von Video 1!“.

Für den Bereich Authoring läßt sich feststellen, daß eine der Hauptschwierigkeiten darin besteht, ein ausgewogenes Verhältnis zwischen „Authoring Capabilities“ und „Expressive Power“ zu erreichen. Ein ideales System würde maximale Authoring Capabilities bieten und so die Spezifikation jeder vorstellbaren Szenerie ermöglichen. Außerdem würde es danach streben, die Ausdrucksmöglichkeiten zu maximieren und dabei gleichzeitig den dafür nötigen Werkzeugsatz zu minimieren. Dadurch kann die Komplexität für Autoren und Entwickler des Systems reduziert werden, ohne Funktionalität einzubüßen.

3.6. Verschiedene Formen der Interaktion

In diesem Abschnitt sollen die verschiedenen möglichen Formen von Interaktion beschrieben werden, die im Kontext einer Lehr- und Präsentationssoftware denkbar wären.

3.6.1. Globale Interaktion

Die globale Interaktion ist eine Form der Interaktion, bei der die Abarbeitung des gesamten Präsentationsablaufes betroffen ist. Dies können beispielsweise Aktionen wie das Starten, Pausieren oder Stoppen des gesamten Abspielvorgangs sein. Allen globalen Interaktionen gemeinsam ist, daß diese in der Regel nicht vom Autor einer Präsentation berücksichtigt werden müssen, da sie vom Abspielprogramm autonom behandelt werden.

3.6.2. Lokale Interaktion

Im Gegensatz zur globalen Interaktion gibt es die lokale Interaktion, die sich innerhalb eines Präsentationsablaufes abspielt und sich auf diesen auswirkt. Die lokale Interaktion ist kontextsensitiv und muß immer durch den Autor einer Präsentation spezifiziert werden.

3.7. Ereignisgesteuerte Systeme

In Analogie zu vielen heute verbreiteten APIs und Programmiersprachen hat sich des weiteren auch ein an die Methodik der GUI-Programmierung angelehnter Ansatz verbreitet, die ereignisgesteuerten Systeme. Ihre Funktionsweise basiert im Kern darauf, daß durch bestimmte Konditionen Ereignisse (Events) ausgelöst werden. Solche Konditionen können beispielsweise Mausklicks, Tastatureingaben oder gar Medienereignisse, wie das Erreichen des Endes eines Videos sein. Der Autor kann zur Erstellungszeit der Präsentation Behandlungsroutinen (Event Handler) hinterlegen, die bei Eintreten bestimmter Ereignisse aktiviert werden und durch den Autor spezifizierte Aktionen ausführen. Ereignisgesteuerte Systeme können sowohl sprachenorientierter als auch graphischer Natur sein.

Ein Beispiel für einen ereignisgesteuerten Ansatz liefert HTML+TIME, wo die Ereignissteuerung mit JavaScript Event Handlern realisiert werden kann.

3.8. Technische Möglichkeiten für Interaktion

Wenn ein Präsentationssystem mit interaktiven Fähigkeiten ausgestattet werden soll, müssen im Vorfeld zunächst einmal auch die dafür einsetzbaren technologischen Ansätze beleuchtet werden. Heute haben sich zwei breite Felder herauskristallisiert, in die sich im Grunde alle heute existierenden Präsentationssysteme einordnen lassen.

3.8.1. Sprachenorientierte Systeme

Die erste Gruppe von Systemen ist die der sprachenorientierten Systeme. Diese Systeme basieren auf einer meistens als Skriptsprache realisierten Programmiersprache beziehungsweise einer Beschreibungssprache. Bekannte Beispiele sind zum Beispiel SMIL als Beschreibungssprache oder LINGO als Programmiersprache. Ebenso dient bei HTML+TIME JavaScript als Programmiersprache. HTML+TIME ist mit HTML als Beschreibungssprache und JavaScript als Programmiersprache ein Hybrid unter den sprachenorientierten Modellen.

3.8.2. Graphisch orientierte Systeme

Im Gegensatz dazu existieren graphisch orientierte Systeme, die auf den Einsatz einer Skriptsprache verzichten, beziehungsweise deren eventuelle Existenz gezielt vor dem Autor und Benutzer verbergen. Die Spezifikation der Szenarios geschieht hier zumeist vollkommen GUI-basiert. Die räumlichen Bearbeitungsmöglichkeiten orientieren sich häufig am WYSIWYG-Prinzip[5], wohingegen beispielsweise zeitbasierte Abläufe oft mit abstrahierenden Editiermöglichkeiten, wie graphischen Zeitstrahl-Repräsentattoren arbeiten.

3.8.3. Gegenüberstellung

Es ist von Wichtigkeit, die Zielgruppe speziell der Autorenumgebung eines Systems genauer zu untersuchen. Die Gruppe der Konsumenten der Präsentation gerät im Normalfall nicht mit der technischen Realisierung der Interaktion in Kontakt, da diese ihre Auswirkungen fast ausschließlich beim Autorenwerkzeug zeigt.

Die Nutzer solcher Software sind in der Regel nicht technisch orientiert. Dieser Nutzerkreis empfindet es als intuitiver, Szenarios graphisch zu spezifizieren. Graphisch orientierte Systeme bieten jedoch aufgrund von verschiedenen Limitationen von Software mit graphischen Oberflächen, wie Resourcenverbrauch oder Ergonomie (Komplexität, Überschaubarkeit) meistens, wenn auch nicht zwangsläufig, eine geringere Flexibilität als sprachenorientierte Systeme. Im Sinne der in Abschnitt 3.5 (Seite 21) beschriebenen Kategorisierung der Autorenwerkzeuge gilt für graphische Systeme, daß ihre Ausdrucksmöglichkeiten höher sind, ihre Flexibilität meistens geringer.

Durch den Einsatz einer Programmiersprache hingegen erreichen Systeme im Normalfall eine sehr große Flexibilität. Dieses wird aber durch den Nachteil einer „Technisierung“ des Erstellungsprozesses einzelner Medienobjekte erkauft. Damit wird der Autor gezwungen, in eventuell

[5] WYSIWYG=What You See Is What You Get

nicht von ihm zu erwartender Weise wie ein Programmierer zu abstrahieren. Diese Systeme erreichen eine höhere Flexibilität, bieten aber geringere Ausdrucksmöglichkeiten.

In der Praxis finden sich oft hybride Systeme, die sowohl GUI-basiertes Editieren ermöglichen, als auch über eine Skriptsprache verfügen, die bei Bedarf zur Spezifikation komplexer Vorgänge herangezogen werden kann. Häufig bieten diese Systeme zusätzlich auch Ereignissteuerung.

4. Analyse des bestehenden Systems

Im Rahmen dieser Diplomarbeit soll ein kontextsensitives Interaktionsmodell in ein bestehendes multimediales Schulungssystem integriert werden. Bei dem zu erweiternden System handelt es sich um MobIT[1], welches die Technologie des Multimedia Information Repository (MIR)[2] nutzt. In diesem Kapitel wird eine Analyse dieser beiden Systeme durchgeführt, da Kenntnis von MobIT und MIR für das Verständnis der folgenden Kapitel dieser Diplomarbeit benötigt wird.

4.1. MIR

Das MIR (Multimedia Information Repository) System ist ein Repository zur strukturierten Speicherung von Medienobjekten und multimedialen Dokumenten. Zum System gehören zudem einige Basis-Anwendungen zur Verwaltung des Systems. Da dieses System eine wesentliche Grundlage dieser Diplomarbeit darstellt, soll in diesem Kapitel zunächst eine Einführung

[1] siehe [6]
[2] siehe [7]

in das MIR System gegeben werden. Für vertiefende, über eine Einführung hinausgehende Informationen zu MIR sei der interessierte Leser auf das zu diesem Thema veröffentlichte Papier „An Environment for Processing Compound Media Streams“[3] verwiesen.

4.1.1. MIR Philosophie

In Zeiten des Telelearning bestehen Lerneinheiten unter Umständen aus einer Vielzahl verschiedenartiger Medienobjekte und Medientypen. Da diese verschiedenartigen Objekte nicht unabhängig voneinander sind, sondern untereinander in definierten räumlichen und zeitlichen Beziehungen stehen, muß ein geeignetes System zeitliche sowie räumliche Relationen zwischen den Objekten eines Dokumentes abbilden können. Besonderes Augenmerk wurde dabei darauf gelegt, daß einzelne Medienobjekte sowie auch komplex strukturierte Einheiten aus Medienobjekten in beliebigen anderen Lerneinheiten wiederverwendet werden können [4].

Ein wichtiger Aspekt ist die Notwendigkeit der Fähigkeit des Systems, kooperatives und netzwerkverteiltes Arbeiten an Medienobjekten und Multimedia-Dokumenten zu ermöglichen.

Alle Medienobjekte und Dateien werden in ein virtuelles Dateisystem abgebildet, welches als Schlüssel Dateinamen, ähnlich dem UNIX-Dateisystem verwendet (zum Beispiel „/MobIT-/Praesentation1/index.mob“). Durch diese Analogie zu regulären Dateisystemen wird der Zugriff auf die Objekte für die Nutzer erleichtert und die Übersicht im Repository erhöht.

Eine Kernanforderung war die Unterstützung beliebiger Medientypen, um auch zukünftige Formate ohne Änderungen am System selbst verarbeiten zu können. Für die komfortable Benutzung von Medien mit großen Binärdaten ist die Möglichkeit des Streamings vorgesehen.

Wie bereits erwähnt, spielt der Aspekt des verteilten Arbeitens bei MIR eine große Rolle. Damit ist speziell auch die Nutzung über das Internet gemeint. MIR setzt daher auf die Verwendung von standardisierten Zugriffsmethoden und Protokollen, wie JDBC, CORBA oder HTTP.

Das MIR System ist der Versuch, eine Umgebung zu schaffen, die es ermöglicht, komplexe multimediale Anwendungen mit vergleichsweise geringem Implementierungsaufwand zu erstellen. Das System soll den Entwickler dabei neben dem eigentlichen Repository auch mit verschiedenen anderen Werkzeugen, wie beispielsweise Frameworks, unterstützen.

Als letzter und wichtiger Punkt sei hier die geforderte Plattformunabhängigkeit zu nennen, um in der heterogenen Betriebssystemwelt des Internet das Spektrum der unterstützten Plattformen zu maximieren.

[3] siehe [8]
[4] siehe auch [6]

4.1.2. Architektur von MIR

Das MIR System verwendet eine mehrschichtige Client-Server Architektur. Abbildung 4.1 (Seite 29) stellt die Zusammenhänge der einzelnen Teile des Systems im Überblick dar.

- **Datenbankschicht**

  Die Datenbankschicht besteht aus einer Relationalen Datenbank (Sybase DataServer). Der Zugriff auf die MIR-Datenbank erfolgt jedoch nicht direkt mittels SQL sondern über ein API aus Stored Procedures.

- **Middlewareschicht**

  Die Middlewareschicht besteht aus einem Java Applikationsserver (Sybase Jaguar). Es existieren Komponenten für das Klassen-, Objekt-, Nutzer-/Gruppen-Management sowie für den Zugriff auf das virtuelle Dateisystem. Alle MIR Komponenten sind auf CORBA-Basis implementiert und können von Client-Applikationen verwendet werden.

- **MIR Application Framework**

  Das MIR Application Framework (MAF) ist ein Java API für MIR Client-Applikationen, die eine Swing/JFC[5]-Oberfläche besitzen. Es bietet Unterstützung unter andererem für das Session-Management, den Repositoryzugriff sowie Nutzer- und Gruppenverwaltung. Zudem sind diverse Utility-Klassen in MAF enthalten, die sonstige, häufig benötigte Funktionalitäten zur Verfügung stellen. Mit dem MAF kann die Entwicklung dieser Applikationen vereinfacht werden und so Entwicklungszeit eingespart werden. Eine gemeinsame Basis für Client-Applikationen kann außerdem den Wartungsaufwand reduzieren. MAF-konforme Applikationen können mittels Java Web Start[6] aus dem WWW-Browser heraus direkt über das Internet gestartet werden.

- **XML MIR Taglib**

  Das neuste, gerade in der Entwicklung befindliche API im MIR System ist die XML MIR Taglib. Sie stellt XML-Applikationen eine komfortable Schnittstelle zum MIR System bereit. Die XML-Anwendungen werden über eine XML-Erweiterung (Cocoon[7]) in einem Servlet Runner ausgeführt.

- **Client-Applikationen**

[5] JFC=Java Foundation Classes
[6] siehe [13]
[7] siehe [1, zuletzt besucht: 25.10.2001]

Das MIR System macht keine verbindlichen Vorgaben, in welcher Weise Anwendungen implementiert werden sollten. Der Phantasie der Entwickler sind hier also keine Grenzen gesetzt. Dennoch ist die Benutzung einer der Hilfs-APIs (MAF, XML MIR Taglib) aufgrund der dadurch entstehenden Vorteile ratsam.

- **MIR Management**

  Es sind mehrere Managementwerkzeuge für das MIR System vorhanden beziehungsweise werden zur Zeit implementiert. So stehen derzeit ein Klassenmanager, ein Nutzer-/Gruppenmanager sowie ein Repository-Browser zur Verfügung, die zur Administration des Systems genutzt werden können. Diese Applikationen sind als MAF-Applikationen konzipiert.

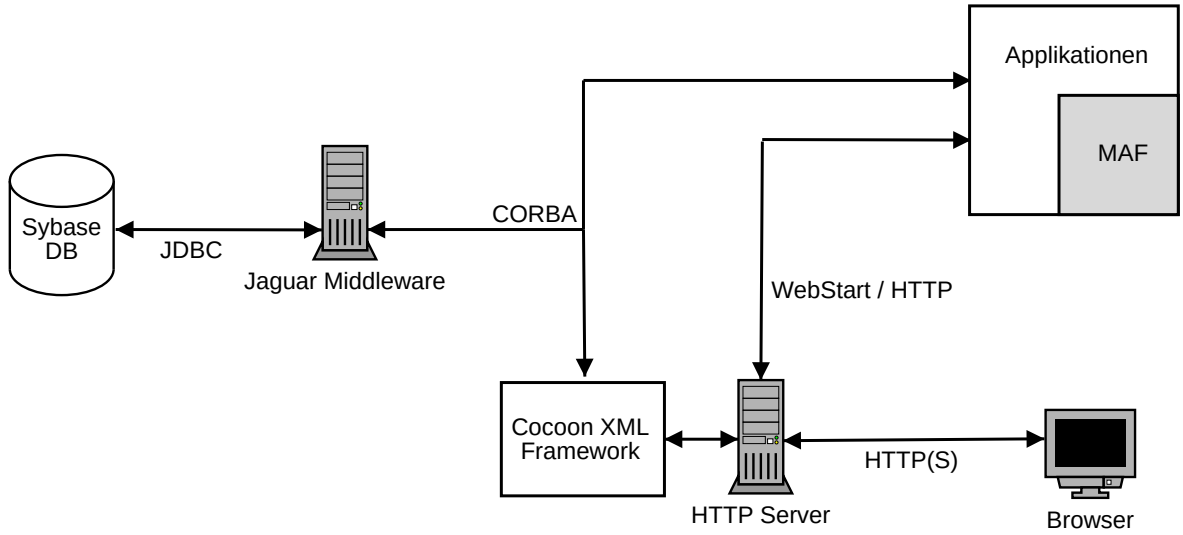

Abbildung 4.1.: MIR Architektur

4.1.3. Das MIR Repository-Konzept

- **Klassen**

  Alle im Repository gespeicherten Objekte gehören einer definierten Objektklasse an. Eine Klasse in diesem Sinne besitzt einen eindeutigen Namen und stellt ein Schema für die Erzeugung neuer Objekte dar. Das MIR System unterstützt Einfachvererbung und erlaubt so den Aufbau einer Klassenhierarchie durch die Applikationsentwickler. Alle Klassen, die direkt oder indirekt von der Klasse ComplexObject abgeleitet sind, können Attribute definieren, die Instanzen dieser Klasse besitzen sollen. Im Unterschied zu Klassendefinitionen in objektorientierten Programmiersprachen besitzen Klassen und Objekte im MIR

System keine eigenen Methoden. Abbildung 4.2 (Seite 31) zeigt einen Überblick über die vordefinierten MIR Basisklassen.

- **Datenobjekte (DOBs)**

  Datenobjekte enthalten die tatsächlichen Mediendaten im MIR Repository. Sie entsprechen weitestgehend normalen Dateien eines Dateisystems. Alle Datenobjekte sind Objekte der Klasse GenericDob und besitzen keinerlei Objektattribute. Jedem Datenobjekt ist ein MIME-Type zugeordnet, der das Format der Daten kennzeichnet.

- **Medienobjekte (MOBs)**

  Medienobjekte im Sinne von MIR erfüllen zwei verschiedene Funktionen. Einerseits dienen sie dazu, Mediendaten in Form von Datenobjekten mit beschreibenden Attributen zu versehen, indem sie sie als Hülle kapseln. Andererseits lassen sich in einem Medienobjekt auch andere Medienobjekte referenzieren. Auf diese Weise werden Relationen zwischen Medienobjekten im MIR System abgebildet. Diese Art der Referenzierung wird in erster Linie zur Realisierung von räumlichen Beziehungen verwendet werden.

  Technisch werden andere Objekte in einem Medienobjekt jedoch nicht direkt referenziert, sondern über sogenannte Target-Listen. Eine Target-Liste enthält dabei einen Vektor von Target-Objekten, die eine Referenz (den Dateinamen) auf das Zielobjekt enthalten.

  Medienobjekte sind Instanzen der Klasse GenericMob oder einer von ihr abgeleiteten Klasse.

- **Ereignisse (Events)**

  Events sind Objekte der Klasse GenericEvent oder einer davon abgeleiteten Klasse. Sie sind wesentlicher Bestandteil der zeitlichen Komponente des MIR-Systems.

  Events erweitern Medienobjekte über die in jedem Medienobjekt vorhandene Event-Liste um eine zeitliche Komponente. Diese Event-Liste ist als lokale Zeitachse eines Medienobjekts zu betrachten. Events haben daher immer einen festgelegten Zeitpunkt relativ zur Zeitachse des Medienobjektes. Außerdem sind zumeist applikationsspezifisch weitere Attribute enthalten. Viele Events beziehen sich außerdem auf ein Zielobjekt. Dieses Zielobjekt muß sich in der Target-Liste des Medienobjektes befinden, muß also von ihm referenziert werden. Im Dokumentenkontext bedeutet dies, daß Events bei MIR also nur für einem Medienobjekt strukturell untergeordnete Objekte definiert werden können.

- **Attributtypen**

  Von den im MIR verfügbaren Attributtypen entsprechen viele den von herkömmlichen Programmiersprachen wie beispielsweise Java angebotenen Datentypen. So existieren in

MIR derzeit die Typen Integer, Float, Double, String, Enumeration und Target. Enumeration ist ein Typ, der genau einen Wert aus einer fest hinterlegten Liste von möglichen Werten erlaubt. Ein Attribut vom Typ Target kann als Wert einen Verweis auf eine Referenz in der Target-Liste eines Objektes annehmen. Mit diesem Typ können applikationsspezifische Objektrelationen über Attribute realisiert werden.

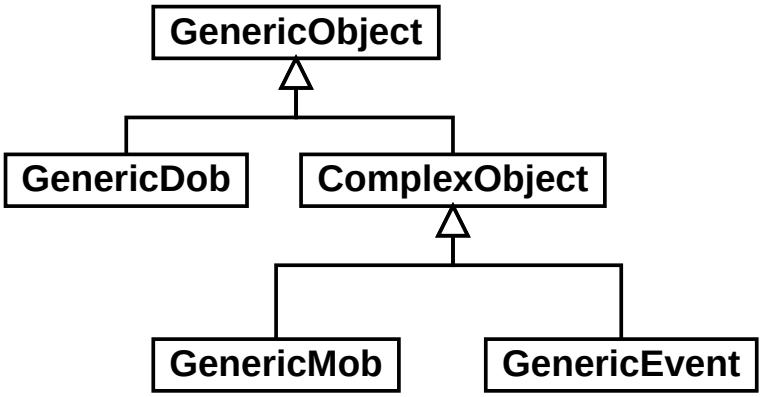

Abbildung 4.2.: MIR Basisklassen

4.2. MobIT

MobIT[8] (Media Objects In Time) ist eine als MIR-Applikation realisierte Lehr- und Präsentationssoftware. MobIT ist wie MIR selbst vollständig in JAVA implementiert, ist aber nicht interaktionsfähig.

MobIT wurde ursprünglich von Björn Feustel im Rahmen seiner Diplomarbeit[9] als eigenständige Software zur Verarbeitung von multimedialen Präsentationen entwickelt. Die ursprüngliche Version benutzte XML zur Datenspeicherung. Seine modulare Architektur ermöglichte die spätere Portierung auf das MIR-System.

Im Verlauf dieser Diplomarbeit wird diese MIR-Applikation auf Basis des in Kapitel 5.2 (Seite 38) beschriebenen Interaktionsmodells um Interaktionsfähigkeit erweitert.

4.2.1. MobIT Konzepte

Ziel von MobIT ist es, eine internetbasierte Plattform für das Telelearning bereitzustellen.

[8] siehe [6, zuletzt besucht: 16.10.2001], [5]
[9] siehe [5]

Eine Präsentation setzt sich aus Daten- und Medienobjekten zusammen. Deren Funktionsweise ist äquivalent zu Medienobjekt (siehe S.30) beziehungsweise Datenobjekt (siehe S.30) des MIR-Systems. Diese Medien- und Datenobjekte können dann durch Spezifikation zeitlicher und räumlicher Beziehungen hierarchisch zu Lerneinheiten strukturiert werden. Ein Autorenwerkzeug unterstützt den Autor bei der Erstellung der Lerneinheiten.

Als Grundsatz von MobIT bleibt zu erwähnen, daß besonderer Wert auf die Wiederverwendbarkeit von Objekten und ganzen Lerneinheiten gelegt wurde. Ebenso wird eine breite Palette von Medien in Präsentationen unterstützt, zum Beispiel Text, Bild oder Video. Ein JAVA-Applet ermöglicht das plattformunabhängige Abspielen der Präsentationen über das Internet. Ein Subserver-Mechanismus bietet dabei die Möglichkeit, streamingfähige Medien zur Abspielzeit abzurufen.

4.2.2. Das MobIT Datenmodell

Eine MobIT-Präsentation besteht aus einer nach räumlichen und zeitlichen Kriterien hierarchisch strukturierten Menge von Medienobjekten (Mobs) und Elementen. Die Knoten in dieser Struktur sind Mobs, während die Blätter typischerweise durch Elemente repräsentiert werden.

Die Elemente bilden die kleinste Einheit in diesem Datenmodell. Sie enthalten die binären Mediendaten, wie zum Beispiel ein JPEG-Bild, jedoch keine Attribute, welche die Daten beschreiben. Elemente sind äquivalent zu den Datenobjekten im MIR.

Medienobjekte (Mobs) hingegen enthalten selber keine Mediendaten. Sie gehören einer definierten Objektklasse an und besitzen die für die entsprechende Klasse festgelegte Menge an Attributen. Mobs erfüllen grundsätzlich eine von zwei möglichen Funktionen:

1. Kapselung von Elementen

   Mobs werden genutzt, um Elemente mit beschreibenden Attributen zu versehen, wie beispielsweise Schriftgröße oder Hintergrundfarbe. Das wird erreicht, indem sie die Elemente kapseln, d.h. referenzieren.

2. Strukturbildung

   Durch Mobs wird die Struktur einer Präsentation definiert. Sie können können beliebig viele andere Mobs referenzieren. Diese Möglichkeit wird zur räumlichen Strukturierung von Präsentationen genutzt.

Ein Mob kann also entweder nur andere Mobs oder nur Elemente referenzieren. Mobs sind die kleinste von der Laufzeitumgebung autark darstellbare Einheit einer MobIT-Präsentation. Die

Funktionalität der Mobs bei MobIT wird von den MIR-Medienobjekten abdeckt. Alle MobIT-Mobs werden im MIR als Objekte der Klasse *PlayableMob* gespeichert.

Abbildung 4.3 (Seite 33) gibt einen Überblick über die hierarchische Struktur einer Präsentation.

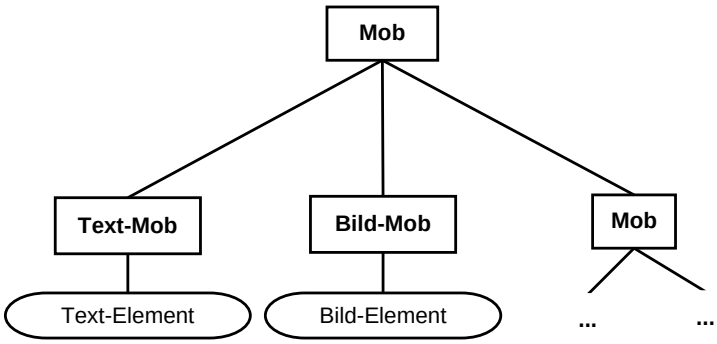

Abbildung 4.3.: Präsentationsstruktur

4.2.3. Zeitsteuerung in MobIT

Die Zeitsteuerung wird bei MobIT dadurch realisiert, daß jedes Mob mit einer Zeitleiste ausgestattet ist. An dieser Zeitleiste sind zu bestimmten Zeitpunkten Aktionen angeordnet, wie zum Beispiel „init“, „show“ oder „hide“. Jedes Mob enthält eine Liste solcher Aktionen, die sogenannte Play-List (Abspielliste). Die Aktionen können, müssen sich jedoch nicht auf andere (Ziel-)Objekte beziehen. Mögliche Zielobjekte müssen sich direkt in der nächsten Hierarchiestufe der Präsentation befinden.

Technisch wird für die Zeitleiste der Mobs der Eventmechanismus der MIR Medienobjekte verwendet.

Die Eventbehandlung ist so gelöst, daß zu jedem Typ von Events („start“, „stop“, etc.) jeweils eine korrespondierende

1. MIR-Klasse für die Speicherung im Repository sowie
2. Methode in der Klasse Mob mit abgeleitetem Namen (z.B. doInit())

existiert. Die Behandlungsmethode wird zur Laufzeit vom Timer-Thread des Players aufgerufen, wenn das Event abgearbeitet wird.

4.2.4. Architektur von MobIT

MobIT verfügt über eine modulare Client-Server Architektur. Eine saubere Trennung der Schichten der Architektur mit definierten Schnittstellen bringt Flexibilität durch die leichte Anpaßbarkeit an spezifische Umgebungen, wie zum Beispiel MIR. Abbildung 4.4 (Seite 34) gibt einen Überblick über das MobIT-System.

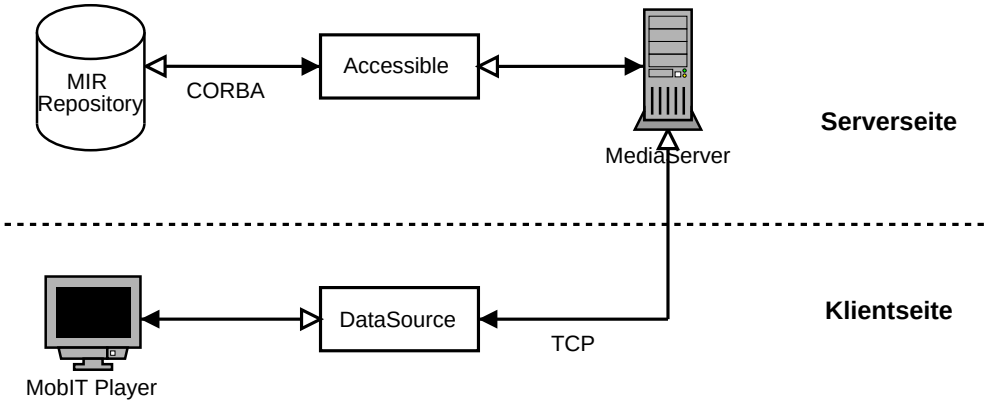

Abbildung 4.4.: MobIT Architektur

Player

Der MobIT Player ist ein Abspielprogramm für die MobIT Lerneinheiten beziehungsweise Präsentationen. Er ist als Java Applet implementiert und daher unabhängig von der Plattform des Client-Computers.

Datenquellen (Datasources)

Datenquellen (Datasources) sind der Mechanismus, über den der MobIT Player die Präsentationsdaten lädt. Eine Datenquelle enthält Funktionen zum Laden von Datenobjekten, Medienobjekten und Play-Listen. Die Art der Datenquelle ist bei MobIT nicht festgelegt. Es existiert

derzeit aber nur eine Referenzimplementierung für eine Datenquelle, die ihre Daten vom MediaServer bezieht, und TCP zur Kommunikation mit dem MediaServer einsetzt.

Datenquellen implementieren zudem einen clientseitigen Caching-Mechanismus über die Methode cachePresentation(), um die Ladezeiten insbesondere bei Fernverbindungen mit geringer Bandbreite zu verbessern.

MediaServer

Der MediaServer dient dazu, der Datenquelle des MobIT Players die benötigten Strukturinformationen beziehungsweise Objekte einer Präsentation zu liefern. Er verfügt dazu über eine über das Internet zugängliche Schnittstelle (TCP-Port). Klienten (Player-Datenquellen) stellen eine TCP-Verbindung zu ihm her und können über festgelegte Kommandos Objekte einer Präsentation sowie Informationen zu ihnen abrufen, wie beispielsweise Play-Lists oder Meta-Informationen (Attribute) der Objekte einer Präsentation.

Der MediaServer ist so konzipiert, daß er unabhängig von der tatsächlich für die Speicherung verwendeten Technologie arbeitet. Für den Zugriff auf die gespeicherten Präsentationsdaten wird ein Interface *Accessible* verwendet, für das für jede unterstützte Speicherungstechnologie eine Implementierung vorliegen muß. *Accessible* enthält den Kommandos des MediaServers entsprechende Methodenrümpfe.

Zum besseren Verständnis sei hier noch kurz die Kommunikation zwischen der Datenquelle des Players (*MobDataSourceTCP*) und dem MediaServer mittels TCP erläutert. Die Datenquelle öffnet eine TCP-Verbindung zu dem MediaServer. Auf dieser Verbindung werden Text-Kommandos von der Datenquelle an den MediaServer gesendet (zum Beispiel „GET /MobIT/v1/7000/7000.mob mi“). Der MediaServer erstellt als Resultat der Anfrage ein Java-Objekt, welches die abgefragten Daten enthält. Dieses wird aber nicht direkt zum Klienten übermittelt, sondern vorher in ein Übertragungsobjekt (*TransferObject*) gekapselt. Übertragungsobjekte verfügen zusätzlich über eine Liste von *Beschreibungsobjekten* (Klasse *DescriptionObject*, bestehend aus Status-ID und Nachricht), die Statusinformationen und Informationen über bei der Abfrage eventuell aufgetretene Fehler enthält. Diese Übertragungsobjekte werden dann über die TCP-Verbindung mittels Java-Serialisierung zur Datenquelle übertragen.

Subserver

Die Subserver sind der Streaming-Mechanismus von MobIT. Bei streamingfähigen Medientypen kann die Auslieferung der Binärdaten an den Klienten außer in einem monolithischen

Block auch kontinuierlich zur Abspielzeit erfolgen. Auch eine Übertragung in Echtzeit wäre (bei Implementierung von Echtzeit-Protokollen wie beispielsweise RTP[10]) theoretisch möglich.

Der Subserver-Mechanismus ist im MediaServer integriert. Jedem Medientyp kann dabei eine zuständige Subserver-Klasse zugeordnet werden, die für die Durchführung der eigentlichen Übertragung verantwortlich ist.

Autorenwerkzeug

Die ursprüngliche MobIT-Version konnte ohne ein spezielles Autorenwerkzeug auskommen. Da sie XML als Format für die Datenspeicherung verwendete, war ein Texteditor für die Bearbeitung von Präsentationen ausreichend. Im Kontext des MIR-Systems ist eine Bearbeitung von Objekten mit einem sochen Werkzeug konzeptbedingt nicht möglich. In der letzten Zeit ist daher begonnen worden, ein graphisches Autorenwerkzeug für MobIT zu entwickeln. Dieses Werkzeug unterstützt in der momentanen Version jedoch lediglich die rudimentäre Bearbeitung räumlicher Beziehungen. Insgesamt ist das Autorenwerkzeug derzeit größtenteils nicht funktionsfähig oder nicht implementiert. Noch vollkommen offen ist die Implementierung der Unterstützung zeitlicher Relationen.

[10] **R**eal-Time **T**ransport **P**rotocol

5. Konzeption

5.1. Diskussion

In Abschnitt 3.8 (Seite 23) wurden bereits die technischen Möglichkeiten zur Realisierung interaktiver Fähigkeiten erläutert. Ergänzend wurden in Kapitel 3 (Seite 17 ff) einige wichtige Punkte bezüglich Autorenwerkzeugen diskutiert.

Bei dem vorliegenden, auf MIR basierenden System erscheint die Realisierung auf Basis einer (Skript-)Sprache nicht ratsam. Bei der Verwendung einer Skriptsprache würde der große Nachteil entstehen, daß eine Integritätsprüfung in der Datenbankschicht nicht angemessen realisierbar wäre. Zudem würde die Implementierung einer Sprache nicht trivial sein und einen erheblichen Konzeptions- und Implementierungsaufwand bedeuten, der den Rahmen dieser Diplomarbeit sprengen würde.

Stattdessen soll im Rahmen dieser Arbeit versucht werden, einen modellbasierten Ansatz umzusetzen. Das dafür verwendete Modell, welches in Kapitel 5.2 (Seite 38) erläutert werden soll, soll größtmögliche Flexibilität bei kleinstmöglicher Werkzeugpalette bieten.

5.2. Interaktionsmodell

Als Modell liegt dieser Arbeit das Interaktionsmodell von Dr. Thomas Schmidt und Björn Feustel[1] zugrunde, welches ich in diesem Abschnitt näher erläutern werde.

5.2.1. Zustandsautomat

Das Modell geht von der Tatsache aus, daß sich der Zustand einer Präsentation zu jedem Zeitpunkt durch einen überschaubaren Parametersatz mit einer Zustandsliste beschreiben läßt. Die in der Zustandsliste enthaltenen Einträge ergeben sich aus dem Karthesischen Produkt aller Zustandsparameter (Objekteigenschaften) aller Medien- und Datenobjekte einer Präsentation.

Ein Eintrag in der Zustandsliste könnte wie folgt aussehen:

Instruktion	t	dt	x	y	Scale x	Scale y	ID	Parameter
start	0	2	10	10	1.0	1.0	4711	font=„Arial“

Änderungen an der Präsentationsstruktur wären in der Zustandsliste sichtbar: So würde das Einfügen neuer Objekte in eine Präsentation zu einer Erweiterung der Zustandsliste um den Zustand des neuen Objektes führen. Äquivalent würden beim Entfernen eines Objektes aus der Präsentation auch alle Zustandseinträge entfernt werden.

[1] siehe [10]

Ungleich schwieriger gestaltet sich der Fall von Änderungen an bestehenden Objekten. Hierfür wird eine Möglichkeit benötigt, Zustandseinträge modifizieren zu können. Das Modell sieht den Einsatz von Aktionen vor, um Zustände zu modifizieren.

5.2.2. Aktionen

Auf eine Interaktion in einer Präsentation folgt immer eine Aktion. So könnte zum Beispiel das Aktivieren eines Hyperlinks oder Buttons der Auslöser für eine Aktion sein.

Aktionen im Sinne des Modells sind Verknüpfungs-Masken für Zustände. Beim Anwenden einer solchen Aktion auf einen Zustand werden alle Felder der Aktion mit dem Zustand verknüpft, wobei leere Felder übergangen werden. Es ergibt sich dann der neue Zustand.

Beispiel für Aktion:

Instruktion	t	dt	x	y	Scale x	Scale y	ID	Parameter
				1.0	1.0			background=„white“

Das obige Beispiel-Aktion würde bei ihrer Auslösung die Hintergrundfarbe von Objekten, die die Skalierungsfaktoren 1.0/1.0 besitzen, auf weiß ändern.

5.2.3. Konditionierung

Die Interaktion erfolgt aber allein auf der Grundlage von Aktionen nicht zu vorbestimmten Zeitpunkten, sondern kann zunächst einmal theoretisch zu einer beliebigen, beim Erstellen der Präsentation nicht bekannten Zeit stattfinden. Ebenso wären vielfältige andere Bedingungen, wie zum Beispiel ein bestimmter Zustand eines anderen Objektes, als Gültigkeitskriterien denkbar. Es ist also eine Art von Konditionierung nötig, um kontextsensitive Interaktionen realisieren zu können.

Für diese Zwecke sieht das Modell die Verwendung von Konditionen vor. Eine Kondition deckt alle Parameter eines Objektzustandes ab, so daß die größtmögliche Flexibilität erreicht wird. Konditionen werden bei ihrer Prüfung gegen die Einträge der Zustandsliste geprüft. Es wäre unpraktikabel, zur Konditionierung immer einzelne, genau spezifizierte Werte für die einzelnen Kriterien vorzugeben. In einem solchen Fall müssten alle möglichen Ablaufpfade einer Präsentation zu ihrem Erstellzeitpunkt explizit spezifiziert werden. Es wird eine unscharfe Spezifikation der Kondition benötigt. Das Modell sieht vor, Bereiche gültiger Werte für die Kriterien der Konditionen vorzugeben.

Ein weiterer wichtiger Aspekt ist die Festlegung der Interaktionsziele, d.h. der Objekte auf die eine Interaktion / Aktion wirken soll. Eine feste Vorgabe der Zielobjekte zum Zeitpunkt

des Erstellens der Präsentation ist nicht möglich, da der jeweilige Zustand der Präsentation dort noch nicht bekannt ist. Die Zielobjekte müssen daher dynamisch zur Laufzeit bestimmt werden.

Dies wird dadurch gelöst, daß eine Kondition für jeden Eintrag der Zustandsliste evaluiert wird. Alle Zustände, für die die Evaluierung den Wert „wahr“ ergibt, bilden die Zielmenge für die Interaktion.

Jeder Kondition können Aktionen zugeordnet werden, die im Fall des Aktivierens der Interaktion gemäß der Beschreibung im vorigen Abschnitt auf die Ergebnismenge der Evaluierung angewendet werden.

Die folgende Beispielkondition würde von der siebten bis zur achten Zeiteinheit für alle Objekte evaluieren, deren momentan ausgeführtes Kommando „start“ ist:

Instruktion	t	dt	x	y	dx	dy	Scale x	Scale y	ID	Param.	Aktion	Folge
start	7	1										

Eine Verknüpfung von Konditionen im Sinne des UND- oder ODER-Operators ist ebenso möglich und wichtig. Verkettete Konditionen entsprechen dabei dem UND-Operator. Mengen als Werte der Konditionsparameter ergeben eine ODER-Verknüpfung.

Eine Interaktion ist genau dann möglich, wenn alle ihre verketteten Konditionen „wahr“ evaluieren.

5.2.4. Klick-Bereiche

Für eine Interaktion (mit der Maus) ist es nötig, eine Fläche zu definieren, die als klickbarer Bereich fungiert. Ein Klick in solch einen Bereich führt dann letztendlich zum Auslösen der Aktion. Die folgende Tabelle verdeutlicht die Parameter für klicksensitive Bereiche:

Mob-ID	x	y	Breite	Höhe	Kondition
#47	10	10	50	25	#11

5.3. MobIT Repositoryschicht

5.3.1. Abbildung der Datenstrukturen

Die Datenstrukturen werden wenn möglich direkt in MIR Klassenspezifikationen umgesetzt. Alle Klassen mit Ausnahme von *ClickInteraction* sind Subklassen der Klasse *GenericMob*.

5.3.2. Mengen

Eine Sonderrolle nimmt die zumindest bei Konditionen erforderliche Unterstützung für Mengen als Attributwerte ein. MIR sieht jedoch keine Mengen als Werte vor. Daher wird der Weg gewählt, Mengen als Zeichenketten darzustellen. Die Analyse der Zeichenketten muß dabei allerdings von der Applikation erfolgen. Es wird eine Hilfsklasse geschaffen, die Methoden bereitstellt, um mit diesen Mengen komfortabel operieren zu können. Damit einher geht allerdings der Verzicht auf mögliche Integritätsprüfungen für Mengen in der Datenbankschicht.

5.3.3. Masken

Da Aktionen im Interaktionsmodell eine zu verknüpfende Maske darstellen, muß die Möglichkeit bestehen, einzelne Attribute bei der Verknüpfung der Maske zu übergehen. MIR unterstützt für alle Datentypen den Wert *null*, welches soviel wie „kein Wert gesetzt“ bedeutet. Dies muß in den Java-Klassen Java ebenso über den Wert *null* realisiert werden. Es dürfen daher keine fundamentalen Java-Datentypen verwendet werden, da diese nicht den Wert *null* annehmen könnten. Abhilfe schaffen hier die Klassen aus dem Package java.lang (Float, Integer, etc.).

Es bietet sich also an, diese Möglichkeit für zu ignorierende Masken-Felder zu verwenden.

5.3.4. Aktionen und Konditionen

Nachdem nun die Klassen MobITAction und MobITCondition analog zu den Datenstrukturen des Interaktionsmodells definiert werden können, fehlen noch die Beziehungen zwischen beiden Typen von Objekten. Vom Modell ausgehend, müssen einer Kondition eine oder mehrere Aktionen zugeordnet werden können. Daher wird nun festgelegt, daß die Target-Liste einer MobITCondition Referenzen auf die entsprechenden MobITAction-Objekte enthält. Abbildung 5.1 (Seite 41) zeigt eine UML-Darstellung der Aktionen und Konditionen.

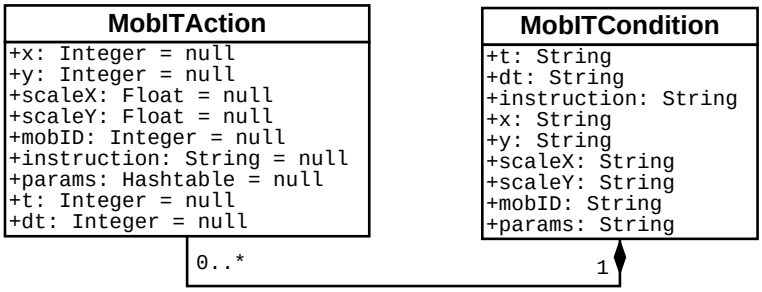

Abbildung 5.1.: Aktionen und Konditionen

5.3.5. Zuordnung zu Medienobjekten

Die Objekthierarchie ausgehend von den Konditionsobjekten muß nun noch einem Medienobjekt zugeordnet werden können. Es wird zunächst eine (abstrakte) Klasse *MobITInteraction* eingeführt, deren abgeleitete Klassen die beschreibenden Attribute einer möglichen Interaktion definieren können. Solche Attribute könnten beispielsweise begrenzende Koordinaten für einen räumlichen Bereich sein, in den geklickt werden kann. Durch Einführung einer Klassenhierarchie an dieser Stelle wird Flexibilität im Hinblick auf die Form der Interaktion (Mausklick, Tastendruck, akustisch) ermöglicht. Diese Diplomarbeit wird den Fokus jedoch einschränkend auf die Form der Interaktion durch Mausklick richten. Andere Formen könnten aber in der Zukunft problemlos implementiert werden.

Diese Interaktions-Objekte können dann über die Target-Liste beliebig viele Konditionsobjekte referenzieren. Auf diese Weise wird der UND-Operator für Konditionen realisiert.

Jedes Medienobjekt erhält eine Liste von MobITInteraction-Objekten, die die einzelnen Interaktionsmöglichkeiten des Objektes spezifizieren. Da kein Datentyp für Objektlisten zur Verfügung steht, muß dabei entweder ein Containerobjekt verwendet werden, oder alle Interaktionsobjekte müssen gemischt mit den untergeordneten Medien- bzw. Datenobjekten in der Target-Liste des Objektes referenziert werden. Letzteres enthält in seiner Target-Liste alle zugeordneten Interaktionsobjekte.

Die Verwendung eines separaten Containerobjektes würde die strukturelle Komplexität eines von MobIT verwalteten Medienobjektes um eine Hierarchiestufe mehr erweitern als die direkte Referenzierung. Andererseits ergibt sich so ein Problem bei der Verarbeitung der Objekte. Da ein Target-Objekt lediglich den Objektschlüssel jedoch nicht die MIR-Klasse des referenzierten Objekts enthält, müßte zur Ermittlung der Objektklasse (Medien-/Daten- oder Interaktionsobjekt) jeweils das referenzierte Objekt selbst geladen werden. Aus Laufzeitgründen wird der zweite Ansatz, die direkte Referenzierung, gewählt. Dazu wird jedoch aus Effizienzgründen eine Erweiterung der Targets empfehlenswert: Sie werden um die Angabe der MIR-Klasse erweitert. Bei Medienobjekten steht dort der voll qualifizierte Klassenname, bei Datenobjekten der Wert **null**.

5.3.6. Erweiterung der Klasse PlayableMob

Die Klasse PlayableMob, die Basisklasse aller MobIT-Medienobjekte, muß, wie im vorherigen Abschnitt bereits verdeutlicht, erweitert werden. Bei der MIR Klasse beschränkt sich die Änderung auf das Hinzufügen eines neuen Attributes „name“ vom Datentyp String. Dieses Attribut soll dem Autor eine benutzerdefinierte Kennzeichnung seiner Objekte unabhängig vom Dateinamen ermöglichen.

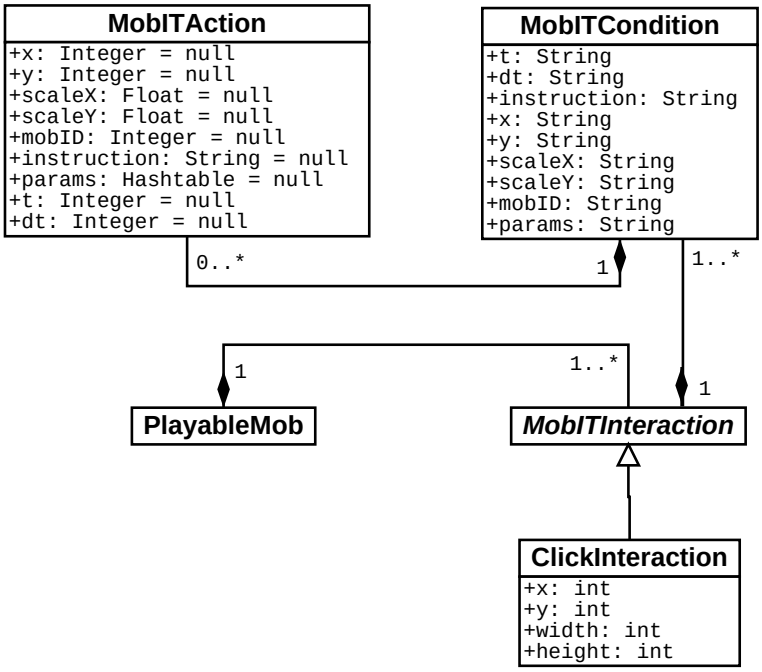

Abbildung 5.2.: Erweiterung PlayableMob

Eine Berücksichtigung des Attributes „name“ ist vorerst nur in der MIR-Klasse und im Autorenwerkzeug vorgesehen, da diese Information lediglich dort benötigt wird.

Die Java-Klasse *PlayableMob* wird so erweitert, daß bei der Initialisierung von Instanzen dieser Klasse die Interaktionsobjekte aus der Target-Liste entfernt und in einer separaten Interaktions-Liste im Objekt ablegt werden. Dadurch wird der spätere Zugriff auf die Interaktionsstrukturen erleichtert.

5.4. MobIT Laufzeitumgebung

5.4.1. MediaServer

Der MediaServer und in direkter Abhängigkeit davon die MIR-Accessible-Klasse (*DBaccess*) erhalten eine zusätzliche Abfragemöglichkeit für die Interaktionsstruktur eines Objektes.

Eine Erweiterung des MediaServers bedingt natürlich auch eine äquivalente Erweiterung des *Accessible*-Interfaces und implementierender Klassen (im Moment *DBaccess*), da der Media-Server im Grunde genommen nur ein Interface darstellt. Die tatsächliche Verarbeitungslogik übernimmt bei MobIT die *Accessible*-Implementierung.

Die *Accessible*-Implementierung liest bei Ausführung der Interaktionsabfrage die Interaktionsobjekte aus dem Repository aus und erstellt daraufhin eine die Interaktion spezifizierende Hier-

archie aus Java-Objekten. Dazu werden Java-Klassen benötigt, die als Datenwert im zurückgelieferten Transferobjekt enthalten sind.

5.4.2. Objektklassen

Es werden, analog zu den MIR-Klassen, auch Java-Repräsentationen der durch das Interaktionsmodell definierten Klickbereiche, Konditionen und Aktionen erstellt. Die Attribute dieser Klassen entsprechen den vom Interaktionsmodell vorgegebenen Parametern.

Die Klasse für Interaktionen erhält ein zusätzliches Flag, welches anzeigt, ob die Interaktion gültig ist. Dieses Flag ist nur in Bezug auf den Player relevant und wird bei der Evaluierung der Konditionen der Interaktion entsprechend gesetzt.

Aktionen erhalten zusätzlich eine Liste von Mob-Verweisen, die zur Laufzeit bei der Evaluierung ermittelt werden.

Die Klassen sind serialisierbar, so daß Instanzen direkt vom MediaServer an den Klienten serialisiert und dort weiterverwendet werden können. Dadurch entfallen zusätzliche Konvertierungen der Daten beziehungsweise Objekte im MediaServer und Player.

5.4.3. Player

Der Player ist die komplexeste Komponente der Laufzeitumgebung. Um das Interaktionsmodell zu implementieren, muß die auf Play-Listen und Ereignissen basierende Mob-Verarbeitung erweitert werden. Es muß beispielsweise die Zustandsliste des Interaktionsmodells realisiert werden, die für die bisherige Verarbeitung von Präsentationen nicht benötigt wurde.

Umwandlung des Player-Applets

Die bisher verwendete Applet-Technologie eignet sich nicht zur Realisierung einer universell verwendbaren Player-Komponente. Der ursprüngliche Zustand startete den Player aus dem MIR-System heraus über eine Applet-Wrapperklasse. Diese Lösung bewährte sich in der Praxis jedoch nicht, da im Zusammenhang mit dem Swing-basierten MIR-Klienten öfter Darstellungs- und Stabilitätsprobleme auftraten.

Das Threadingkonzept des ursprünglichen Players ist zudem nicht für die Verwendung in einer Komponente geeignet. Es existiert keine Möglichkeit, die vom Player gestarteten Threads nach dem Abspielen wieder zuverlässig zu stoppen und die von ihnen belegten Resourcen wieder zu deallozieren.

Der Player muß daher zunächst in eine wiederverwendbare Swing-Komponente transformiert werden. Das beinhaltet neben der Umwandlung des Player-Applets auch eine Änderung der Threading-Implementierung des Players.

Globale Interaktion

Die Realisierung globaler Interaktion (siehe Abschnitt 3.6.1, Seite 23) erfordert lediglich Änderungen am Player des MobIT-Systems.

Aktionen wie „stop“, „pause“ oder „resume“ können durch Modifikation der Threadingimplementierung implementiert werden. Zusätzlich werden die *Runnable*-Methoden der Player-Komponente und die Datenquelle geändert, um die Aktionen „start“ und „stop“ von außerhalb des Players kontrollieren zu können.

Eine „pause“-Funktion beispielsweise ließe sich ähnlich realisieren. Hierzu müßte zusätzlich jedoch noch eine Methode für den Zugriff auf die Aktion von außen implementiert werden.

Datenquelle

Die Datenquelle *MobDataSourceTCP* wird in Analogie zum MediaServer um eine Abfragemöglichkeit der Interaktionsstrukturen erweitert. Im Falle der Interaktion übernimmt die Datenquelle die Funktion eines Proxies zum MediaServer. Die vom MediaServer abgefragten Interaktionsobjekte werden ohne Konvertierung weitergereicht.

Initialisierung der Medienobjekte

Bei der Initialisierung der Medienobjekte (Klasse *mobit.client.Mob*) aufgrund zugehöriger init-Events im Player muß die Interaktionsstruktur des Mobs berücksichtigt werden. Über die Datenquelle wird zusätzlich noch die Interaktionsstruktur des Objektes abgefragt und im Objekt zwischengespeichert. Damit stehen zur Laufzeit einer Präsentation die Interaktionsstrukturen in den Mobs bereit.

Evaluierung der Konditionen

Die Evaluierung der Konditionen soll nebenläufig in einem eigenen Thread erfolgen. Der Thread wird beim Initialisieren des Players erzeugt und beim Threadmanager des Players registriert.

Das Modell sieht vor, die Konditionen der Interaktionen gegen die Zustandsliste der Präsentation zu prüfen. Der Player bietet explizit keine solche Zustandsliste an. Er hält jedoch zur Laufzeit bereits eine hierarchische Struktur von Mobs der gesamten Präsentation vor. Da die Mobs

mit den gegebenenfalls in ihnen enthaltenen Elementdaten den Zustand einer Präsentation wiedergeben, entspricht die Zustandsliste im Grunde allen Objekteigenschaften der linearisierten Mob-Hierarchie.

Der Evaluations-Thread läuft zur Laufzeit über die gesamte im Speicher befindliche Hierarchie einer Präsentation und evaluiert alle an die Medienobjekte gebundenen Interaktionen. Dabei auftretende Mengen von Verweisen auf Ziel-MedienObjekte für die Aktionen werden aus Effizienzgründen in den Aktionen zwischengespeichert, um spätere Evaluationsläufe einzusparen. Ergibt die Evaluierung der Konditionen einen *true* entsprechenden Wert, also keine leere Menge, wird die Interaktionsmöglichkeit durch Setzen des zuvor besprochenen Flags aktiviert und gegebenenfalls dem Betrachter der Präsentation extra kenntlich gemacht.

Auslösen der Interaktion

Um ein Auslösen der Interaktion durch den Betrachter zu ermöglichen, wird je Interaktionsform (Maus, Tastatur, etc.) ein Event-Listener benötigt. Die Event-Listener werden bei jedem Mob mit Interaktionsstruktur registriert. Erfolgt eine Interaktion durch den Betrachter, wird die Interaktionsstruktur des Mobs untersucht, ob ein gültiges und passendes (Koordinaten, Taste, etc.) Interaktionsobjekt in diesem Mob existiert. Wird ein solches Interaktionsobjekt gefunden, werden alle Aktionen des Interaktionsobjektes auf die in den Aktionen temporär verwiesenen Zielobjekte angewendet.

5.5. Autorenwerkzeug

Das Autorenwerkzeug müsste zunächst um die Fähigkeit zur Bearbeitung zeitlicher Zusammenhänge erweitert werden. Danach könnte eine entsprechende graphische Bearbeitungsmöglichkeit für die Interaktionsdaten der Objekte bereitgestellt werden. Bei Betrachtung des momentanen Zustandes des Autorenwerkzeugs ist eine Neuimplementierung beinahe unausweichlich und wird daher hier auch angestrebt. Einzig der bereits ansatzweise funktionsfähige räumliche Editor sowie die Objekteditoren für Bild und Text können übernommen werden.

Das neue Autorenwerkzeug soll dann folgende Merkmale aufweisen:

- Es wird, begründet auf den MIR- und MobIT-Systemen, ein operationaler Ansatz verfolgt[2].

[2] siehe Abschnitt 3.5 auf Seite 21

- Das neue Autorenwerkzeug wird im Gegensatz zum alten Ansatz immer nur jeweils eine Hierarchiestufe von Präsentationsobjekten darstellen und bearbeiten. Soll ein Objekt einer anderen Hierarchiestufe bearbeitet werden, so muß es in einer separaten Editor-Instanz geladen werden.

- Das räumliche Editieren soll vollständig graphisch und WYSIWYG-orientiert erfolgen können. Diesem Umstand wird dadurch Rechnung getragen, daß der alte, graphisch orientierte räumliche Editor als Basis dafür wiederverwendet wird.

- Das zeitliche Editieren der Abspiellisten (Play-Lists) wird in Listenform geschehen. Zur Bearbeitung einzelner Events dient ein neuer Bearbeitungsdialog für Ereignisse.

- Die Interaktionen werden teilweise graphisch und teilweise formularbasiert erfaßt und bearbeitet. Klickbereiche werden in den räumlichen Editor integriert und können mit der Maus erstellt und verändert werden. Ein neuer Bearbeitungsdialog für diese Objekte bietet formularbasiertes Editieren der Konditionen und daran gebundener Aktionen der Klickbereiche. In dem Dialog werden Eingabefelder für die einzelnen Attribute der Konditions- und Aktionsobjekte angeboten.

- Das Abspielen des bearbeiteten Objektes kann direkt aus dem Autorenwerkzeug initiiert werden (Preview-Funktion). Dafür wird ein minimaler MAF-Klient erschaffen, der die neue Player-Komponente in MAF-Klienten komfortabel nutzbar macht. Es werden Buttons für (globales) „start“ und „stop“ angeboten, sowie eine Statusleiste, die den Status der Player-Komponente anzeigt.

Dem Leser wird aufgefallen sein, daß die beiden Editoren für die zeitlichen und interaktiven Eigenschaften nicht graphisch, sondern, für den Autor einer Präsentation relativ abstrakt, in Listen- bzw. Formularweise realisiert werden. Der Aufwand, eine komplett graphisch orientierte Bearbeitungsmöglichkeit dieser Eigenschaften anzubieten, wäre jedoch zu hoch, um ihn im Rahmen dieser Diplomarbeit erbringen zu können.

6. Implementierung

In diesem Abschnitt soll die Implementierung einiger der im konzeptionellen Teil beschriebenen Mechanismen dargestellt werden.

6.1. Repositoryschicht

In der Repositoryschicht wurden keine Veränderungen an Quellcodes vorgenommen. Es wurden die in Abschnitt 5.3.1 (Seite 40) besprochenen Klassen erstellt. Dazu wurde das vom MIR System bereitgestellte KlassenEditor-Applet verwendet (siehe Abbildung 6.1, Seite 49).

6.2. Laufzeitumgebung

6.2.1. MediaServer

Der MediaServer wird wie besprochen um eine Variante des get-Kommandos erweitert, die die Interaktionsstruktur eines Objektes zurückliefert. Der Syntax der neuen get-Variante lautet folgendermaßen:

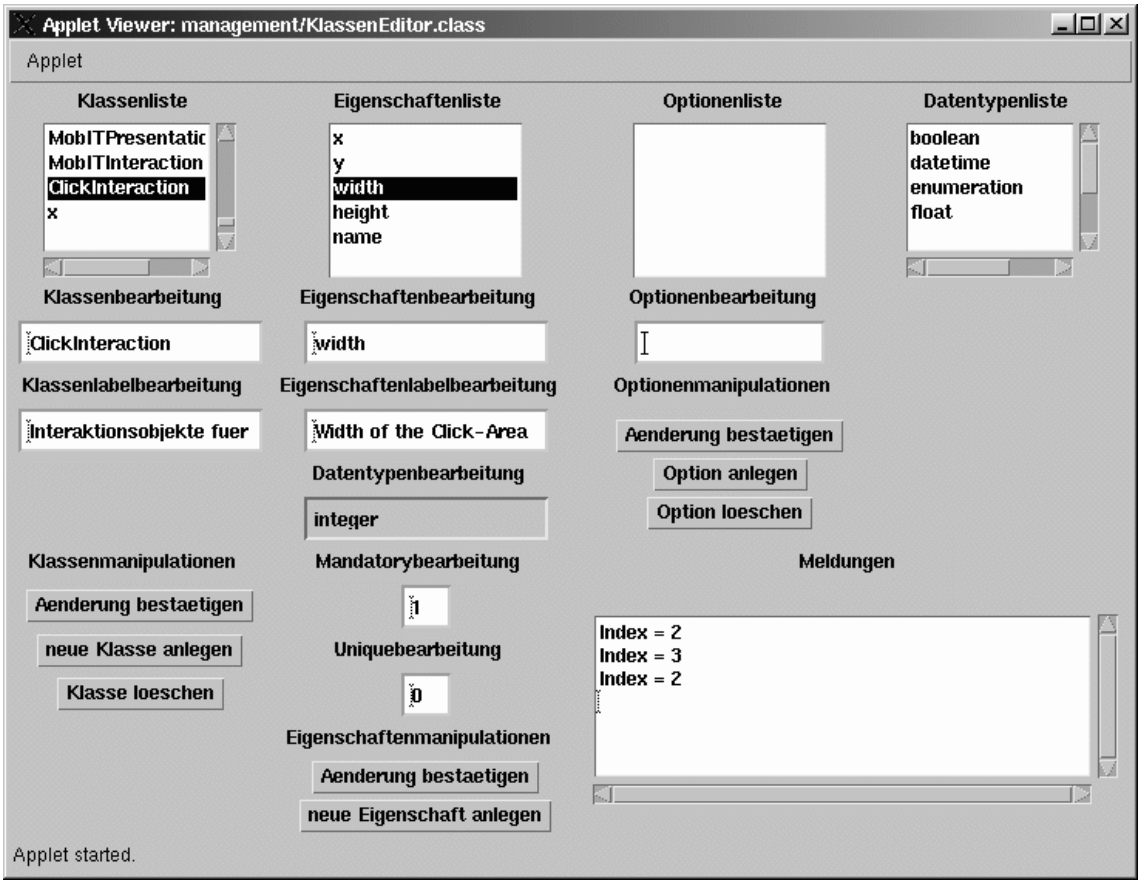

Abbildung 6.1.: MIR KlassenEditor-Applet

get <Repository-Pfad> **int**

Um die neue get-Syntax zu implementieren, die in den Folgenden Abschnitten erläutert werden.

Erweiterung des Accessible-Interfaces

Der MediaServer greift für alle Zugriffe auf eine Instanz einer das Accessible-Interface implementierenden Klasse zu. Daher wird das Interface um eine Methodendefinition erweitert, die die Interaktionsstruktur eines durch das get-Kommando spezifizierten Objektes zurückliefert. Listing 6.1 auf Seite 50 zeigt die neue Methodendefinition.

Listing 6.1: Erweiterung des *Accessible*-Interfaces

```
1 public interface Accessible {
2   ...
3
4   // ermittelt die interaktionsstruktur eines Mobs
5   public TransferObject getINT(String path);
6
7   ...
8 }
```

Erweiterung des MIR-Accessibles

Die Änderung des *Accessible*-Interfaces zieht direkt nach sich, daß alle implementierenden Klassen die neue getINT-Methode implementieren müssen. Im Rahmen dieser Arbeit wird die MIR-Implementierung des Interfaces, die Klasse *DBaccess* mit der neuen getINT-Methode ausgestattet. Die Änderungen seien hier an Listing 6.2 (Seite 51) erläutert:

Zeile 5 to referenziert das im konzeptionellen Kapitel beschriebene *TransferObject*, welches den Returnwert der Methode kapselt.

Zeile 6 Ein *DescriptionObjekt* wird benötigt, um das *TransferObject* mit Statusinformationen zu versehen.

Zeile 10 Das über den Parameter path spezifizierte MedienObjekt wird aus dem MIR Repository geladen. objectManager wurde im Konstruktor von *DBaccess* initialisiert und referenziert eine Instanz der CORBA Objekt-Manager-Komponente der MIR Middleware.

Zeilen 12-19 Im Fehlerfalle wird ein präpariertes *TransferObject* zurückgeliefert, das dem Klienten Informationen über den Fehlerzustand liefert.

Zeilen 21-23 Bei Erfolg wird im *TransferObject* die Interaktionsstruktur des MedienObjekts und der Status des Kommandos („OK“) gesetzt.

Zeile 24 Die Klasse *PlayableMob* analysiert ein MIR MedienObjekt und kann für MobIT präparierte Datenstrukturen, wie Play-Lists, etc. liefern. *PlayableMob* dient an vielen Stellen des MIR-Accessibles als Utility-Klasse zur Konvertierung der MIR-Datenstrukturen in MobIT-Datenstrukturen. Die neue getInteraction()-Methode von *PlayableMob* liefert die Interaktionsstruktur des MedienObjekts.

Listing 6.2: getINT()-Methode des MIR-*Accessible*

```
1  public class DBaccess implements Accessible {
2    ...
3
4    public TransferObject getINT(String path) {
5      TransferObject to=new TransferObject();
6      DescriptionObject des=null;
7
8      GenericMob mob=null;
9      try {
10       mob=objectManager.getMob (path);
11     } catch(Exception e) {
12       des = new DescriptionObject();
13       des.msg = "‘Error␣while␣getting␣interaction"’;
14       des.desc = "‘Can’t␣fetch␣mob␣from␣DB."’;
15       des.src = path;
16       des.status = DescriptionObject.TO_ERROR;
17       to.msgs.add (des);
18       to.data = null;
19       return to;
20     }
21     des=new DescriptionObject(DescriptionObject.TO_OK, "‘OK"’);
22     des.src=path;
23     to.msgs.add (des);
24     to.data=(new PlayableMob(mob)).getInteraction();
25     return to;
26   }
27
28   ...
29 }
```

Implementierung der neuen get-Variante

Die Implementierung wird durch Modifikation der Klasse *MediaServer$MobServer* umgesetzt. Die Änderungen werde ich an Listing 6.3 (Seite 52). erklären.

Zeile 2 für jeden get-Subtyp ist eine int-Konstante definiert

Zeile 12 selectionString enthält das letzte Token des Kommandos. Im Falle des Interaktionskommandos wird das Token „int“ verwendet.

Zeile 14 der Rückgabewert der get-Methode wird durch Aufruf der neuen getINT()-Methode des Accessible-Objekts gewonnen. globalPath enthält das zweite Token des Kommandos, beim get-Kommando also den Objektpfad im Repository.

Listing 6.3: Neues MediaServer-Kommando in *mobit.server.MediaServer$MobServer*

```
1  ...
2  static final int getINT    = 7; // interaction structure
3  ...
4  public Object get(String inputLine) {
5    ...
6
7    if (selectionString.equals(....)) {
8      ...
9    }
10
11   // fetch interaction structure
12   else if (selectionString.equals("‘int"’)) {
13     selectionCode=getINT;
14     returnObject=accessible.getINT(globalPath);
15   }
16
17   ...
18   return returnObject;
19 }
```

6.2.2. Player

Umwandlung des Player-Applets

Das Player-Applet *mobit.client.MobIT* wurde in eine von JComponent abgeleitete Klasse *EmbeddedPlayer* überführt. Eine minimale MAF-Player-Applikation (Klasse *MobITv1Player*) de-

monstriert die Verwendung der Playerkomponente und bietet dem Betrachter einfache globale Interaktion („play“,„stop“) über zwei Buttons. Abbildung 6.2 (Seite 53) zeigt einen Screenshot der Player-Applikation.

Abbildung 6.2.: Screenshot der Player-Applikation

Erweiterung der Datenquelle

In der Datenquelle *MobDataSourceTCP* wurde eine neue Methode getInteraction(String) implementiert. Listing 6.4 auf Seite 53 zeigt die neue Methode, welche im Folgenden erläutert sein soll:

Zeile 3 ProcessInst-Instanzen führen bei MobIT die Kommunikation mit dem MediaServer über TCP durch.

Zeilen 7-12 Die *MobDataSourceTCP* enthält einen Caching-Mechanismus für Mobs. Es muß daher geprüft werden, ob das angeforderte Mob bereits im Cache enthalten ist, und ein erneuter Ladevorgang entfallen kann.

Zeilen 14-16 Die *ProcessInst*-Instanz wird initialisiert und (asynchron) gestartet.

Zeilen 17-41 Schließlich wird auf die Komplettierung der Anfrage gewartet und das Ergebnis wird geprüft.

Listing 6.4: neue getInteraction-Methode

```
1 public TransferObject getInteraction(String path) {
2   Globals.Debug ("DS: getInteraction("+path+")");
3   ProcessInst pi = new ProcessInst();
4   TransferObject to = new TransferObject();
5   DescriptionObject des = null;
6
7   if (cache.containsKey (path)) {
```

```
8       PlayableMob mob = cache.get(path);
9       des = new DescriptionObject (DescriptionObject.TO_OK, "Ok.");
10      to.msgs.add(des);
11      to.data = null;
12      return to;
13    } else {
14      pi.request = new String ("get " + path + " int");
15      pi.transObject = null;
16      sendRequest (pi);
17      while (pi.status == ProcessInst.PI_UNFINISHED) {
18        try {
19          Thread.sleep(100);
20        } catch (InterruptedException e) {}
21      }
22
23      if (pi.status != ProcessInst.PI_OK) {
24        des = new DescriptionObject (DescriptionObject.TO_ERROR, pi.msg);
25        to.msgs.add(des);
26        to.data = null;
27        return to;
28      }
29
30      if (pi.transObject.getStatus() == DescriptionObject.TO_OK) {
31        des = new DescriptionObject (DescriptionObject.TO_OK, "Ok.");
32        to.msgs.add(des);
33        to.data = (PlayList)pi.transObject.data;
34        return to;
35      } else {
36        des = new DescriptionObject (pi.transObject.getStatus(),
37          "Problem while getting int " + path);
38        to.msgs.add(des);
39        to.msgs.addAll(pi.transObject.msgs);
40        to.data = null;
41        return to;
42      }
43    }
44  }
```

Initialisierung der Mobs

Die Initialisierung der Mobs in Folge von init-Events wird so erweitert, daß die Interaktionsstruktur des Mobs über die Datenquelle abgefragt wird und zur späteren Verwendung in einer privaten Variablen zwischengespeichert wird. Listing 6.5 auf Seite 55 demonstriert die Änderungen an der doInit-Methode der Klasse *Mob*.

Listing 6.5: Erweiterung von Mob.doInit()

```
1  ...
2  private Vector interaction=null;
3  ...
4  public void doInit(Command _c) {
5    ...
6    // nach Initialisierung der Unter-Mobs
7    TransferObject trob=ds.getInteraction(path);
8    if (trob.getStatus()==DescriptionObject.TO_OK) {
9      interaction=(Vector)trob.data;
10   }
11   ...
12 }
13 ...
```

6.3. Autorenwerkzeug

Das neue Autorenwerkzeug wurde als Swing-basierter MAF-Klient implementiert. Die Implementierung ermöglicht die Bearbeitung jeweils einer Ebene der Präsentationshierarchie. Abbildung 6.3 auf Seite 56 zeigt einen Screenshot des GUI[1] des Autorenwerkzeugs.

Vom alten Autorenwerkzeug wurde lediglich der räumliche Editor übernommen. Im Screenshot ist er auf der Fläche unterhalb der Kennzeichnung „Spatial Pane“ sichtbar. Für die Unterstützung der graphischen Bearbeitung von Klickbereichen mussten jedoch einige Erweiterungen vorgenommen werden. Die Erweiterungen des räumlichen Editors, die in diesem Abschnitt beschrieben werden, ermöglichen eine komfortable graphische Bearbeitung der Klickbereiche für Interaktionen mit der Maus.

[1] GUI=Graphical User Interface

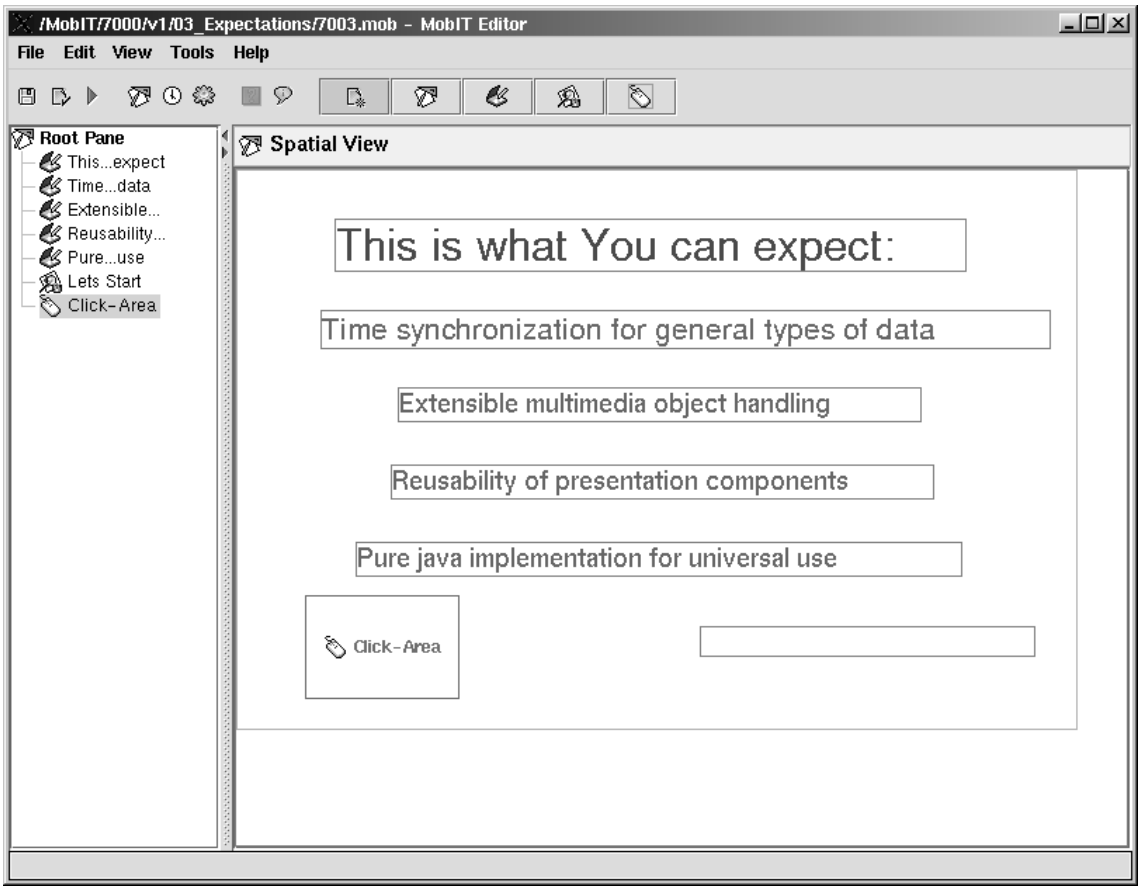

Abbildung 6.3.: Screenshot des neuen Autorenwerkzeugs

6.3.1. Elementklasse *ClickArea*

Für die Verarbeitung von Klickbereichen muß eine passende Elementklasse im Autorenwerkzeug zur Verfügung stehen. Alle Elemente im Autorenwerkzeug haben als Wurzelklasse *MobITElement*. Die Klasse *MobITElement* enthält Attribute für einen Namen (Label) und einen eindeutigen Schlüssel zur Identifizierung im Repository. Abbildung 6.4 (Seite 57) stellt die Klassenhierarchie der Elemente dar. Die Hierarchie ab *DisplayableElement* umfaßt graphisch darstellbare Elemente, die zusätzlich über räumlich beschreibende Attribute wie Breite und Höhe verfügen.

Die Struktur des im Autorenwerkzeug geladenen Medienobjektes wird durch *MutableTreeNode*-Implementierungen (beispielsweise *DisplayableNode* für darstellbare Knoten) abgebildet. Diese Knoten enthalten jeweils als Nutzdaten („UserObject“) ein gekapseltes Element.

Für Klickbereiche wird eine neue, von *DisplayableElement* abgeleitete Klasse *ClickArea* implementiert (in der Abbildung invertiert dargestellt). Die Klasse benötigt neben den geerbten keine weiteren Attribute (siehe Listing 6.6, Seite 57).

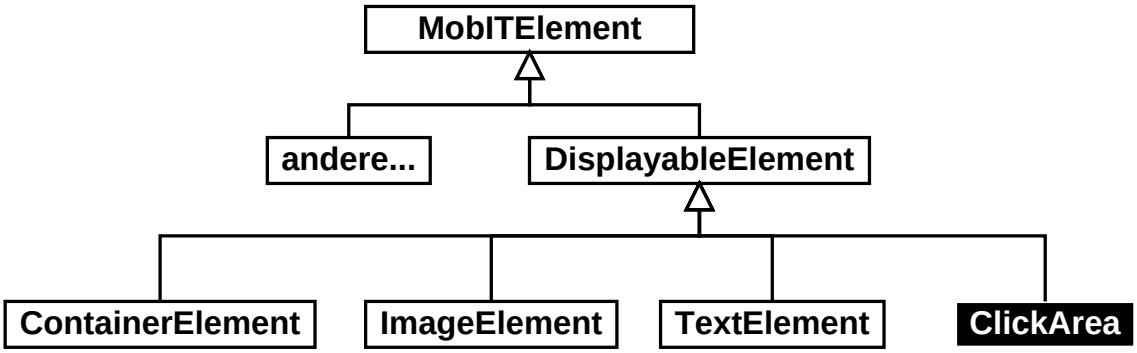

Abbildung 6.4.: Elementklassen des Autorenwerkzeugs

Listing 6.6: Klasse *mobit.authoring.ClickArea*

```
1 public class ClickArea extends DisplayableElement {
2 }
```

6.3.2. Element-Widget für Klickbereiche

Zur Bearbeitung mit dem räumlichen Editor muß je Elementklasse eine Widgetklasse verfügbar sein, welche für die Darstellung des Elementes im Editor verantwortlich ist. Abbildung 6.5 auf Seite 58 zeigt die Widget-Klassenhierarchie des Autorenwerkzeugs, welche sich stark an den Elementtypen orientiert.

Die für Klickbereiche neu geschaffene Widgetklasse *W_ClickArea* ist in der Abbildung invertiert dargestellt. Listing 6.7 (Seite 58) zeigt den Quellcode der Klasse *W_ClickArea*, der im Folgenden erläutert werden soll:

Zeilen 4-5 Der Konstruktor erwartet als Argument eine Referenz auf den zugehörigen Strukturknoten. Dieser wird an den Konstruktor der Superklasse zur Analyse weitergegeben.

Zeilen 6-10 Die Darstellung wird als Label mit dem Namen des Knotens sowie einem beschreibenden Icon (Maussymbol) realisiert.

Zeilen 12-29 Elemente und Strukturknoten sind im Autorenwerkzeug so implementiert, daß das Verändern eines Attributwertes zum Auslösen eines *PropertyChangeEvents* führt. Beliebige Objekte können sich als *PropertyChangeListener* bei den Elementen oder Knoten anmelden und Veränderungen ihrer Attribute überwachen. In der Klasse *W_ClickArea* werden der Name und die geometrischen Attribute überwacht, da diese die graphische Repräsentation des Klickbereichs beeinflussen.

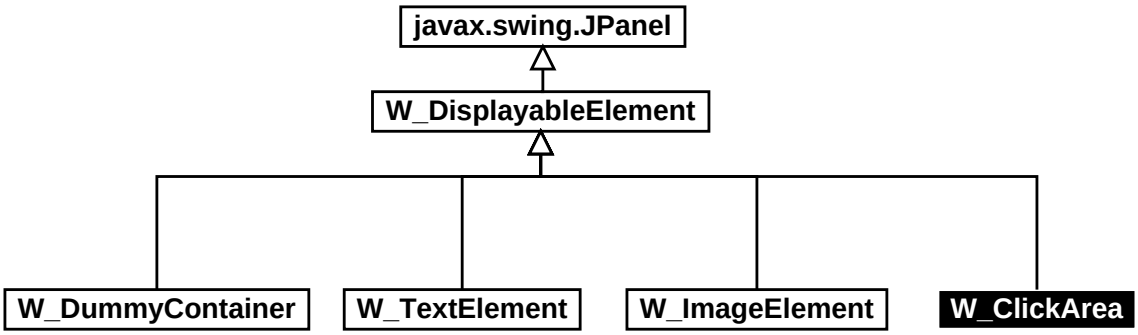

Abbildung 6.5.: Element-Widgets des Autorenwerkzeugs

Listing 6.7: Widget für Klickbereiche

```
1  public class W_ClickArea extends W_DisplayableElement {
2    private JLabel l;
3
4    public W_ClickArea(DisplayableNode node) {
5      super(node,true);
6      setLayout(new GridLayout(1,1));
7      add(l=new JLabel(el.getName()));
8      l.setHorizontalAlignment(SwingConstants.CENTER);
9      l.setVerticalAlignment(SwingConstants.CENTER);
10     l.setIcon(UIManager.getIcon("interaction.clickarea"));
11
```

```
12      el.addPropertyChangeListener("Name",new PropertyChangeListener() {
13        public void propertyChange(PropertyChangeEvent e) {
14          l.setText(el.getName());
15        }
16      });
17
18      PropertyChangeListener pcl=new PropertyChangeListener() {
19        public void propertyChange(PropertyChangeEvent e) {
20          Dimension d=new Dimension(
21            (int)(W_ClickArea.this.n.getScale()*el.getWidth()),
22            (int)(W_ClickArea.this.n.getScale()*el.getHeight()));
23          l.setSize(d);
24          l.setPreferredSize(d);
25        }
26      };
27      el.addPropertyChangeListener("Width",pcl);
28      el.addPropertyChangeListener("Height",pcl);
29      node.addPropertyChangeListener("Scale",pcl);
30    }
31
32  }
```

6.3.3. Einbindung der Klickbereiche

Die Unterstützung für Klickbereiche wurde schließlich dadurch in den räumlichen Editor eingebunden, daß

1. die Zeichenpalette (in der Abbildung 6.3, Seite 56 über der Beschriftung „Spatial View“ befindlich) um eine Anwahlmöglichkeit für den neuen Typ Klickbereich erweitert wurde.

2. die Zentrale Klasse des räumlichen Editors, *W_ContainerPane* analog zu den Änderungen an der Zeichenpalette erweitert wurde.

In der Abbildung 6.3 ist in der linken unteren Ecke der Mobfläche ein in dem editierten Mob enthaltener Klickbereich sichtbar.

7. Zusammenfassung

Das Ziel der Diplomarbeit war, ein kontextsensitives Interaktionsmodell in ein Schulungssystem zu integrieren.

Es wurden zunächst andere, bereits existierende Interaktions-Ansätze untersucht. Dabei hat sich gezeigt, daß zwar einige Ansätze existieren, diese jedoch zumeist entweder für die Verwendung in der Praxis zu komplex waren, oder die von ihren gebotenen Möglichkeiten waren nicht überzeugend. Zudem erwies sich keiner der betrachteten Ansätze zur Integration in das bestehende MobIT-/MIR-System als geeignet. Daher wurde in dieser Arbeit ein neues, innovatives Interaktionsmodell vorgestellt. Dieses Modell erlaubt es, interaktive Szenarios mit einem sehr überschaubaren Werkzeugsatz in eleganter Weise zu spezifizieren.

Im Verlaufe dieser Diplomarbeit wurde die Integration dieses Interaktionsmodells in das bestehende System konzipiert. Dabei wurden das Datenmodell, die Laufzeitumgebung und das Autorenwerkzeug fokussiert.

Schließlich wurde damit begonnen, die konzipierte Integration in das System implementatorisch durchzuführen. Aufgrund des grossen Umfangs der Änderungen und Erweiterungen sowie einiger vorübergehender technischer Probleme und Umstellungen des Systems konnte die Implementierung jedoch nicht in allen Bereichen abgeschlossen werden. Dieses muß zu einem späteren Zeitpunkt im Rahmen des MobIT-Projektes fortgesetzt werden.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] APACHE FOUNDATION: *The Apache XML Project*. http://xml.apache.org. - Oct. 2001 - mailto:apache@apache.org

[2] BRETSCHNEIDER, OLAF: *Allgemeine Informationen*. http://www.informatik.htw-dresden.de/~htw8940/ginf/allginf.htm. - Oct. 2001 - mailto:htw8940@htw-dresden.de. - HTW Dresden

[3] HALASZ, FRANK ; SCHWARTZ, MAYER: The Dexter Hypertext. In: *Communications of the ACM* Vol. 37 (February 1994). No. 2, S. 30-39

[4] JOURDAN, MURIEL ; LAYAÏDA, NABIL ; ROISIN, CÉCILE : Authoring Techniques for temporal Scenarios of multimedia Documents. In: *Handbook of Internet and Multimedia Systems and Applications* (1999), CRC Press LLC, Boca Raton, Florida

[5] FEUSTEL, BJÖRN: *Ein multimediales, zeitbasiertes Lehr- und Publikationssystem*. Berlin, Technische Fachhochschule Berlin, Fachbereich Informatik, Dipl-Arb., 2000

[6] FEUSTEL, BJÖRN: *MobIT! - Media Objects in Time!*. http://www.rz.fhtw-berlin.de/projekte/mobit/. - Oct. 2001 - mailto:feustel@fhtw-berlin.de. - FHTW Berlin

[7] FEUSTEL, B. ; KÁRPÁTI, A.; RACK, T. ; SCHMIDT, T.C. ; TSCHEPKI, V.: *MIR - Multimedia Information Repository*. http://www.rz.fhtw-berlin.de/MIR/. - Oct. 2001 - mailto:{feustel,karpati,rack,schmidt,tschepki}@fhtw-berlin.de. - FHTW Berlin

[8] FEUSTEL, B. ; KÁRPÁTI, A. ; RACK, T. ; SCHMIDT, T.C.: *An Environment for Processing Compound Media Streams (Work in progress paper V 2.0)*. The Changing Universities, The Role of Technology (Proceedings of the 7th International Conference of European University Information Systems). Berlin, 2001

[9] FEUSTEL, B. ; SCHMIDT, T.C. ; MARPE, D. ; PALKOW, M. ; CYCON, H.L.: *Compound Media Streaming in Time*. The 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2001 held at the University of West Bohemia, Campus Bory, Plzen, Czech Republic, 2001

[10] SCHMIDT, TH. ; FEUSTEL, BJÖRN: *The Interaction Model*. Berlin, FHTW Berlin, Hochschulrechenzentrum, Internes Konzeptpapier, 2001

[11] ANTONACCI, M.J. ; MUCHALUAT-SAADE, D.C. ; RODRIGUES, R.F. ; SOARES, L.F.G.: *Improving Expressiveness of XML-based Hypermedia Authoring Languages*. HASHIMOTO, SHUJI: *Multimedia Modeling: Modeling Multimedia Information and Systems (MMM2000)*. World Scientific, Singapore, 2000.

[12] SOARES, LUIZ FERNANDO G. ; RODRIGUEZ, NOEMI L. R. ; CASANOVA, MARCO ANTONIO: Nested Composite Nodes and Version Control in an Open Hypermedia System. In: *Information Systems Journal (Special Issue on Multimedia Information Systems)* Vol. 20 (September 1995). No. 6, Elsevier Science Ltd., Inglaterra

[13] SUN MICROSYSTEMS, INC.: *Java(TM) Web Start*. http://java.sun.com/products/javawebstart/index.html. - Oct. 2001

[14] W3 CONSORTIUM: *The World Wide Web Consortium.* http://www.w3c.org/. - Oct. 2001 - Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University

[15] W3 CONSORTIUM: *W3C Synchronized Multimedia Home page*. http://www.w3c.org/AudioVideo/. - Oct. 2001 - mailto:tmichel@w3.org. - Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University

A. Anlagen

Dieser Arbeit liegt eine CD-Rom mit folgenden Inhalten bei:

/Diplomarbeit/	Diese Arbeit im
arbeit.ps	Postscript-Format
arbeit.pdf	PDF-Format
arbeit.dvi	DVI-Format
/Quellcodes/	Quellcode des
MIR	MIR-Projekts
mobit	MobIT-Projekts