Herausgegeben von Norbert Szyperski, Udo Winand, Dietrich Seibt, Rainer Kuhlen und Rudolf Pospischil

Martin Engelien/Jens Homann (Hrsg.)

Virtuelle Organisation und Neue Medien

Workshop GeNeMe99
Gemeinschaften in Neuen Medien

TU Dresden, 28./29.10.1999

JOSEF EUL VERLAG
Lohmar · Köln

Reihe: Telekommunikation @ Mediendienste · Band 6

Herausgegeben von Prof. Dr. Dr. h. c. Norbert Szyperski, Köln, Prof. Dr. Udo Winand, Kassel, Prof. Dr. Dietrich Seibt, Köln, Prof. Dr. Rainer Kuhlen, Konstanz, und Dr. Rudolf Pospischil, Brüssel

PD Dr.-Ing. habil. Martin Engelien
Dipl.-Inform. (FH) Jens Homann (Hrsg.)

Virtuelle Organisation und Neue Medien

Workshop GeNeMe99
Gemeinschaften in Neuen Medien

TU Dresden, 28./29.10.1999

JOSEF EUL VERLAG
Lohmar · Köln

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

GeNeMe <1999 Dresden> :
GeNeMe 99 : Gemeinschaften in neuen Medien ; Dresden, 28./29.10.1999, an der Fakultät Informatik der Technischen Universität Dresden / Technische Universität Dresden, Fakultät Informatik, Institut für Informationssysteme, Forschungsgruppe "Entwurfsmethoden und Werkzeuge für Anwendungssysteme". Martin Engelien ; Jens Homann (Hrsg.). – Lohmar ; Köln : Eul, 1999
(Reihe: Telekommunikation @ Mediendienste ; Bd. 6)
ISBN 3-89012-710-X

Josef Eul Verlag GmbH
Brandsberg 6
53797 Lohmar
Tel.: 0 22 05 / 91 08 91
Fax: 0 22 05 / 91 08 92
http://www.eul-verlag.de
eul.verlag.gmbh@t-online.de

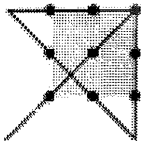
Printed in Germany
Druck: Rosch-Buch, Scheßlitz

Gedruckt auf säurefreiem, 100% chlorfrei gebleichtem, alterungsbeständigem Papier nach DIN 6738

Technische Universität Dresden
Fakultät Informatik • Institut für Informationssysteme
Forschungsgruppe „Entwurfsmethoden und Werkzeuge für Anwendungssysteme"

PD Dr.-Ing. habil. Martin Engelien
Dipl.-Inform. (FH) Jens Homann
(Hrsg.)

an der
Fakultät Informatik der Technischen Universität Dresden



Gefördert von der Klaus Tschira Stiftung,
gemeinnützige Gesellschaft mit beschränkter Haftung

sowie unter Mitwirkung der
GI-Regionalgruppe Dresden

am 28./29.10.1999
in Dresden

E.3. Ein Dokumentmodell für Kursdokumente in Webbasierten Virtuellen Lernumgebungen

Prof. Dr. K. Meißner
Dipl.-Inform. F. Wehner
Technische Universität Dresden
Heinz-Nixdorf Stiftungslehrstuhl für Multimediatechnik

1 Einführung

Virtuelle Lernumgebungen ermöglichen Aus- und Weiterbildung unabhängig von Zeit und Raum, z.B. können Berufstätige während wie auch nach der Arbeitszeit Kurse über ihren PC mit Internetzugang absolvieren und dabei verteilte Lerngruppen und Projektteams bilden. Eine Vielzahl anderer Szenerien virtueller Lernumgebungen sind denkbar. Faktoren, die diese Entwicklung massiv unterstützen, sind der Verbreitungsgrad multimedialer PCs – ca. 7 Mio. Personen über 14 Jahren hatten 1998 in Deutschland beruflich oder privat Zugriff auf Onlinedienste bzw. das Internet – und insbesondere die Dynamik, mit der sich das Wissen in bestimmten Fachgebieten verändert und damit ständige berufliche Weiterbildung erfordert. Die Informations- und Multimediatechnik sind hierfür typische Beispiele. Dieser Entwicklung kann mit klassischen Lehrmethoden und -medien, z.B. Fortbildungsseminaren und dem Lehrbuch, nicht adäquat Rechnung getragen werden. Virtuelle Lernumgebungen bieten deshalb eine zeitgemäße Möglichkeit der kostengünstigen und effizienten Aus- und Weiterbildung.

Unter dem Begriff "Virtuelle Lernumgebung" sollen Systemumgebungen verstanden werden, die alle wesentlichen, für die Wissensvermittlung notwendigen Vorgänge unterstützen und steuern, angefangen von administrativen Aufgaben, wie z.B. die Einschreibung und Abrechnung von Kursen, über Werkzeuge zur Kommunikation und Kooperation Lernender untereinander oder mit dem Tutor bis hin zur eigentlichen Vermittlung und Präsentation von in Form von Kursdokumenten zur Verfügung stehenden Lerninhalten.

Im Mittelpunkt dieser Arbeit stehen Fragen, wie Kursdokumente aufgebaut, modular gestaltet, dynamisch generiert und dadurch an Voraussetzungen, Ziele und den Lernerfolg der Lernenden angepaßt werden können. Folgende Anforderungen können aus der Sicht der Nutzer an virtuelle Lernumgebungen und deren Kursdokumente gestellt werden. Virtuelle Lernumgebungen sollten

- weitgehend unabhängig von spezifischen Eigenschaften des Präsentationssystems sein. Diese Forderung wird durch die Verwendung von (de facto) Standards erreicht.

- modular aufgebaut und dadurch einfach wartbar sein. Das betrifft insbesondere die Anpassung durch die Tutoren an neue Einsatzszenarien und die Aktualisierung der Inhalte.
- sowohl durch die Lernenden als auch durch die Tutoren konfigurierbar sein. So könnte beispielsweise die Navigation der Lernenden in der virtuellen Lernumgebung analysiert werden, um häufig verwendete Pfade in Form von Sichten zu identifizieren und künftigen Lernenden bevorzugt anzubieten. Dies sollte sowohl geführtes wie auch selbstbestimmtes (und evtl. durch Hinweise unterstütztes) Lernen ermöglichen.
- die Kommunikation zwischen Lernenden und Tutoren ermöglichen, die z.B. der Beantwortung von Fragen, der Kontrolle von Übungsaufgaben oder der fachlichen Diskussion dient.
- die Kommunikation und Kooperation zwischen den Studenten unterstützten, z.B. zur Bildung von (räumlich) verteilten Lerngruppen und Projektteams.
- administrative Vorgänge, wie z.B. die Einschreibung in Kurse, Prüfungszulassungen und Abrechnung von Gebühren unterstützen.

Die Entwicklung von Kursdokumenten multimedialer virtueller Lernumgebungen, die das Potential multimedialer Technologien optimal nutzen, ist anspruchsvoll und sehr arbeitsintensiv. Dies hat u.a. mit der Vielfalt der Tätigkeiten und Erfahrungen aus unterschiedlichen Wissensgebieten zu tun, z.B. der Mediengestaltung, Informatik, Didaktik, Lernpsychologie und dem eigentlichen Fachgebiet. Zunächst muß der zu vermittelnde Themenbereich inhaltlich gegliedert werden. Danach ist eine didaktische Strukturierung anhand pädagogischer Modelle und Regeln nötig. Die zu integrierenden Medien, z.B. Texte, Bilder, Audio- und Videoclips sowie interaktive Bestandteile wie Simulationen und virtuelle Praktika, sind zu erstellen und miteinander zu synchronisieren. Außerdem müssen Designer und Lernpsychologen an der Gestaltung, Präsentation und Optimierung des Kurses für unterschiedliche Ausgabeformate mitwirken.

Heutige Autorenwerkzeuge für Kursdokumente sind entweder nicht mächtig genug, um das vollständige Anforderungsspektrum abzudecken oder sie sind zu elementar und erfordern grundlegende Programmierkenntnisse. Multimediale Autorensysteme, z.B. Asymetrix Toolbook [Schult 99] oder Macromedia Director [Welsch 96], erzeugen i.a. Dokumente mit proprietärem Format, die nur durch die Entwickler angepaßt oder erweitert werden können und zu deren Präsentation plattformspezifische Laufzeitsysteme notwendig sind. Die Verwendung von Entwicklungsergebnissen in anderen Kursdokumenten ist meist nur eingeschränkt oder gar nicht möglich. Die dynamische, an die vorliegende Zielgruppe angepaßte Generierung von Kursdokumen-

ten und die Trennung der Inhalte von der Präsentationsform wird von diesen Autorensystemen nicht unterstützt. Gleiches gilt für die kooperative Zusammenarbeit mehrerer Autoren mit z.T. unterschiedlichen Rollen (Fachautoren, Programmierer, Pädagogen, Psychologen, Designer) bei der Entwicklung eines Kurses. Ein Überblick über aktuelle multimediale Autorenwerkzeuge gibt [Rieder 99].

Eine plattformübergreifende Nutzung von Kursdokumenten mit eingeschränkten multimedialen Ausdrucksmitteln ist dagegen im WWW mit der Markup Sprache HTML (*HyperText Markup Language*) [Raggett 98] möglich. Sie gestattet die Gliederung und Formatierung weitgehend beliebiger Hypertexte. Allerdings unterstützt sie nicht die in Kursdokumenten notwendige Definition semantischer Blöcke und Strukturen höherer Granularität, wie z.B. „Übung" oder „Lektion". Eine Lösung bietet hier die Meta-Markup Sprache XML (*eXtensible Markup Language*) [Bray 98], [Harold 98], mit der beliebige Elemente aus semantischer Sicht und Relationen zwischen diesen als Grammatik in Form einer DTD (*Document Type Declaration*) definiert werden können. Sie bildet die Basis für die im folgenden Kapitel vorgestellte, für Kursdokumente entwickelte Markup Sprache "TeachML".

Eine weitere Ursache für die Komplexität der Entwicklung virtueller Lernumgebungen ist die Verwaltung der Lernenden und insbesondere die notwendige Anpassung des Kursmaterials an individuelle Voraussetzungen, Lernziele und Lernfortschritte. Typischerweise sind dies Eigenschaften 'Intelligenter Tutorieller Systeme' (ITS). In diesen repräsentiert das Expertenmodul das zu vermittelnde Wissen sowie die Zusammenhänge zwischen den einzelnen Wissenseinheiten, im Studentenmodul werden Vorwissen, Ziele und aktueller Lernfortschritt der Lernenden abgebildet, das Unterrichtsmodul steuert den Lehr-/Lernprozeß und das Kommunikationsmodul dient i.w. der Kommunikation des Lernenden mit dem System [Lusti 92]. Diese Systeme, insbesondere das Expertenmodul, sind i.a. stark auf das Anwendungsgebiet abgestimmt und unterstützen deshalb die Wiederverwendbarkeit von Lerneinheiten in anderen Systemen nur bedingt. Für den in dieser Arbeit vorgestellten Ansatz sind ITS deshalb nur bez. der Grundlage für die dynamische Erzeugung und Adaption von Kursdokumenten von Interesse.

Das Projekt CHAMELEON (*Cooperative Hypermedia Adaptive MultimEdia LEarn Objects*) [Wehner 99] adressiert die zuvor beschriebenen Probleme und Anforderungen an virtuelle Lernumgebungen. Hauptziel ist die Entwicklung einer Architektur, eines (de-facto) Standards für Formate sowie Schnittstellen und der notwendigen Autorenumgebung zur Realisierung portabler virtueller Lernumgebungen auf der Grundlage wiederverwendbarer Kursbestandteile. Die Vision ist ein globaler Markt von Kurskomponenten und universellen Dokumentenrahmen, entwickelt und zur Verfügung

gestellt von unterschiedlichen Anbietern, z.B. Verlagen, Fachautoren und Lehrern. Unter Verwendung der Autorenumgebung soll es Lehrern und Tutoren ohne substantielle Medien- und Programmierkenntnisse möglich sein, Kursdokumente aus Dokumentenrahmen, Dokumentenbestandteilen und Lernkomponenten zusammenzustellen, indem sie diese erweitern und an die jeweilige Zielgruppe anpassen. Dokumentenrahmen stehen für eine bestimmte Klasse von Kurstypen und repräsentieren u.a. didaktische Strukturen, Dokumentenbestandteile und Komponenten stehen dagegen für bestimmte Aufgaben, z.B. eine Übung, und die Visualisierung bestimmter Inhalte, z.B. die Simulation eines physikalischen Gesetzes. Auf diese Weise erstelltes Kursmaterial soll abhängig vom Lernermodell die Möglichkeit der dynamischen Anpassung an konkrete Lernfortschritte, an die Voraussetzungen des Lernenden und an Lernziele bieten. Auf die Aggregation von Kursfragmenten und die Adaption der Kurse wird in Kapitel 3 eingegangen. Abschließend werden in Kapitel 4 die bisher erzielten Projektergebnisse zusammengefaßt und die noch zu untersuchenden Fragestellungen vorgestellt.

2 Kursdokumente virtueller Lernumgebungen

Kursdokumente in virtuellen Lernumgebungen sind alle Inhalte, die zu vermittelndes Wissen transportieren, der Strukturierung und Präsentation dieses Wissens dienen. Sie sind in einem Dokumentenrahmen abgelegt und lassen sich i.w. als aus passiven und (inter)aktiven Bestandteilen zusammengesetzt betrachten. Passive Dokumentbestandteile sind solche, die üblicherweise zur Vermittlung ihrer Inhalte keine Interaktion mit dem Lernenden erfordern, z.B. Texte, Bilder oder Grafiken. Abhängig vom zu vermittelnden Fachgebiet machen sie häufig den größten Anteil der Inhaltsbestandteile aus. (Inter)aktive Bestandteile, sog. Lernkomponenten, sind funktional geprägt, häufig aufwendig in der Erstellung und führen entweder verdeckte Aktionen aus, z.B. Benutzerbeobachtungs- oder Evaluationskomponenten, oder interagieren mit dem Lernenden um Inhalte und das in ihnen repräsentierte Wissen zu vermitteln, z.B. Simulationen oder virtuelle Praktika.

Die Grenzen zwischen passiven und (inter)aktiven Dokumentbestandteilen sind häufig fließend. So klassifizieren wir Hypertextstrukturen in z.B. HTML als passiv, obwohl die in ihnen enthaltenen Verweise eine Interaktion des Benutzers ermöglichen. Aktive Dokumententeile sind üblicherweise dadurch geprägt, daß sie komplexe Interaktions- und Präsentationsmöglichkeiten aufweisen, die durchaus mit Dokumentbeschreibungssprachen, z.B. SMIL (*Synchronized Multimedia Integration Language)* [Hoschka 98]) formuliert sein können.

2.1 Realisierung von Lernkomponenten mit Java Beans

Zur plattformunabhängigen Realisierung von Lernkomponenten bietet sich die Programmiersprache Java an. Da der Tutor bzw. Lehrende diese Komponenten an die jeweilige Zielgruppe und Lehrsituation anpassen oder um bestimmte Aspekte erweitern können sollte, müssen sie für einen generellen Anwendungsfall realisiert sein und spezielle Mechanismen oder Funktionen zur Spezialisierung enthalten. Das Projekt CHAMELEON verwendet hierfür Softwarekomponenten, konkret Java Beans [Piemont 99], die über eine einheitliche Schnittstelle verfügen und mit Entwicklungswerkzeugen visuell angepaßt und zu kompletten Anwendungen kombiniert werden können. [Sameting 97, S. 68] definiert Softwarekomponenten folgendermaßen:

> *„Reusable software components are self-contained, clearly identifiable artefacts that describe and/or perform specific functions and have clear interfaces, appropriate documentation and a defined reuse status."*

Java Beans lassen sich zur Entwicklungszeit der Anwendung durch Parametrisierung mit automatisch aus den Attributtypen erzeugten bzw. speziell implementierten Property-Editoren anpassen und über ein Ereignisprotokoll und dynamisch generierte Adapterklassen miteinander kombinieren. Damit jedoch ein globaler Markt von Softwarekomponenten entstehen kann, die recherchierbar sind, sich auch funktional einfach anpassen und zu komplexen Anwendungen verknüpfen lassen, müssen Komponentenarchitekturen die im folgenden beschriebenen Eigenschaften besitzen.

Java Beans verbergen ihre Implementierung, sichtbar ist nur ihre Schnittstelle, bestehend aus Attributen, Ereignissen und Methoden. Aus ihr lassen sich keine zuverlässigen Informationen über die Semantik dieser Komponente ableiten. Diese Informationen werden aber z.B. dann benötigt, wenn ein Tutor gezielt nach einer speziellen Simulationskomponente suchen möchte. Zur Zeit ist er dazu auf externe, d.h. nicht in der Komponente enthaltene Beschreibungen bzw. Kategorisierungen angewiesen. Aber auch für die Kombination von Komponenten sind semantische Informationen sinnvoll, z.B. um entscheiden zu können, welche Komponenten kompatibel sind. Nicht immer ergibt die Kombination von Komponenten mit kompatiblen Schnittstellen eine semantisch sinnvolle Einheit. In verschiedenen Arbeiten werden Lösungen vorgestellt. So schlägt [Weck 98] Verträge (*Contracts*) vor, die dazu benutzt werden, Verpflichtungen und Anforderungen zu formulieren, die Anbieter und Nutzer von Services haben. Komponenten erklären, welche Teile dieser Verträge sie erfüllen. Auf diese Weise läßt sich die Kombination von Komponenten unter semantischen Gesichtspunkten vereinfachen. In [Alencar 98] wird ein Modell vorgeschlagen, mit dem Aussagen zur Semantik der Kombination von Komponenten getroffen werden können. Die Verbindungen zwischen Komponenten werden als Designbeziehungen

beschrieben, die durch eine Menge typischer Eigenschaften charakterisiert sind. Damit kann die Komponentenkombination sowohl auf Design- als auch auf Implementierungsebene beschrieben werden. Die existierenden Ansätzen enthalten jedoch z.B. keine Aussagen über einheitliche semantische Kategorien von Services bzw. Eigenschaften von Komponentenverknüpfungen, die eine offene Plattform für die Kombination von Komponenten verschiedener Hersteller schaffen könnten.

Die *Black-Box* Eigenschaft von Komponenten birgt noch ein weiteres Problem. Wie bereits erwähnt, lassen sich Komponenten prinzipiell nur durch Parametrisierung, d.h. in dem vom Entwickler der Komponenten vorgesehenen Maße anpassen. Da jedoch kein Entwickler alle Einsatzszenarien seiner Komponente vorhersehen kann, kommt es vor, daß Komponenten noch funktional angepaßt oder erweitert werden müssen. Es existieren verschiedene Ansätze zur Lösung dieses Problems. Im ADAPT Framework [Heinem 98] wurde eine Konfigurationssprache zur Beschreibung von Komponenten und ihren möglichen Anpassungen entwickelt. Die Anpassung einer Komponente erfolgt hier durch die Einführung von *before*- und *after*-Methoden. In [Teege 97] wird der Ansatz der *Features* speziell für die Anpassung von Groupware Anwendungen eingeführt. *Features* werden in einem Feature-Set verwaltet. Beispiele für *Features* sind strukturelle Informationen über eine Komponente, z.B. das Vorhandensein bestimmter Attribute und Methoden oder auch konkrete Methoden, z.B. ein bestimmtes Synchronisationsverfahren. Realisiert wird dies durch ein auf *Slots* basierendes System, bei dem dynamisch *Features* hinzugefügt und entfernt werden können. Ein dritter Ansatz schließlich benutzt das Konzept der objektbasierten Vererbung [Noble 99] zur Anpassung von Komponenten. Dabei werden Objekte nicht durch Instanziierung von Klassen, sondern durch Kopieren und Anpassen von Musterobjekten, sog. Prototypen, erzeugt. Jedes Objekt besitzt eine Referenz auf sein Prototyp- (*Parent*-) Objekt. Auch hier werden Attribute und Methoden dynamisch in *Slots* abgelegt. Die Anpassung erfolgt neben der oben beschriebenen Möglichkeit des dynamischen Hinzufügens und Entfernens von Attributen und Methoden auch durch die dynamische Änderung des Prototyp-Objekts und somit der vererbten Attribute und Methoden. Eine erste Version einer objektbasierten Erweiterung von Java ist mit der Sprache LAVA [Kniesel 98] verfügbar. Alle erwähnten Ansätze basieren jedoch auf speziellen Programmiersprachen bzw. Spracherweiterungen, so daß sie nur schwer mit existierenden und verbreiteten Komponentenarchitekturen zu realisieren sind.

Zusammenfassend ist zu sagen, daß Java Beans in der aktuellen Form für die Realisierung von Lernkomponenten auf Grund der genannten Defizite nur bedingt geeignet sind, jedoch als Basis für die notwendigen Erweiterungen um semantische Informationen und erweiterte Anpaßbarkeit dienen können.

2.2 Realisierung passiver Dokumentbestandteile mit TeachML

Passive Dokumentbestandteile lassen sich in drei Gruppen einteilen: Inhalte, strukturierende Bestandteile und Benutzerinformationen. Diese Gruppen sollen im folgenden genauer vorgestellt und eine Möglichkeit zur Integration in ein Dokumentmodell aufgezeigt werden.

Wie bereits erwähnt, sind passive, inhaltsbasierte Dokumentbestandteile solche, die üblicherweise keine Interaktionen mit dem Benutzer zur Präsentation ihrer Inhalte benötigen. Einfache Inhaltsbestandteile sind z.B. Texte, Verknüpfungen, Tabellen, Listen und Bilder. Für Kursdokumente sind jedoch semantisch höherwertige Bestandteile denkbar, z.B. ein Bild mit dazugehörender Text- oder Audioerklärung oder ganze Lektionen, die aus einem vermittelnden und einem Übungsteil bestehen. Solche Elemente lassen sich in existierenden Dokumentformaten nicht explizit definieren, sondern nur durch Elemente geringerer Granularität zusammensetzen. Auch die Darstellung multimedialer Inhalte, deren zeitliche Beziehungen und Systemvoraussetzungen lassen sich mit Dokumentenbeschreibungssprachen formulieren (SMIL). Jedoch existiert kein Ansatz, in dem klassische und multimediale Dokumentenbestandteile sowie semantische Strukturen integriert werden können.

Strukturierende Bestandteile sind solche, die Inhaltselemente fachlich, didaktisch oder lernpsychologisch gliedern. Zur inhaltlichen Strukturierung dienen z.B. Container, ein Kurs stellt einen Container für alle Inhaltselemente dar. Den wichtigsten Container bilden die "didaktischen Pfade". Zu jeder Lektion kann es alternative Pfade geben, die sich z.B. durch ihren Schwierigkeits- und Detaillierungsgrad unterscheiden. Abhängig vom Vorwissen des Lernenden und seinem aktuellen Lernerfolg wird für diesen Lernenden der passende Pfad ausgewählt. Somit beinhaltet das Kursdokument stets alle vom Autor des Kurses vorgegebenen alternativen Varianten. Bei der Auswahl eines konkreten Kursdokuments wird abhängig vom Lernermodell eine dieser Alternativen ausgewählt und in ein Ausgabeformat konvertiert (s.u.). Im Projekt CHAMELEON wird untersucht, wie didaktische und lernpsychologische Theorien, Modelle und Regeln sich formalisieren und in einem Dokumentformat repräsentieren lassen.

Will man die Präsentation der Inhalte an die Voraussetzungen, Lernziele und den aktuellen Fortschritt des Lernenden anpassen, so wird ein Modell des Lernenden benötigt. Dieses kann in Form einer History-Liste auch Informationen über die absolvierten Lektionen und gelösten Übungsaufgaben beinhalten. Zudem können Sichten über den Lernfortschritt erzeugt oder häufig genutzte Pfade identifiziert werden. Das Lernermodell steht also in engem Bezug zum Kursdokument.

Für die Entwicklung eines Dokumentformat für Kursdokumente mit den zuvor beschriebenen Dokumentbestandteilen bieten sich Metasprachen wie SGML (*Standard Generalized Markup Language*) [SGML 86] an. SGML ermöglicht die Definition nahezu beliebiger Grammatiken für strukturierte Dokumente, die in Form von Markup Sprachen realisiert werden. SGML Grammatiken sind jedoch nicht immer in Backus-Naur Form darstellbar, was die Entwicklung von Werkzeugen erschwert. Deshalb wurde mit XML eine vereinfachte Form von SGML entwickelt. XML Dokumente lassen sich stets in Backus-Naur Form darstellen und mit XLink und XPointer sind spezielle Erweiterungen für die Realisierung von Verknüpfungen zwischen XML Dokumenten verfügbar. Ein detaillierter Vergleich zwischen SGML und XML ist in [Clark 97] zu finden. Auf Grund der gegenüber SGML einfacheren Struktur und der fast ebenso großen Ausdrucksfähigkeit von XML wurde es als Basis für die Entwicklung einer Grammatik von TeachML gewählt. In TeachML werden, wie in jeder XML Grammatik, Elemente, deren Beziehungen und Attribute definiert. Ein didaktischer Pfad sieht z.B. folgendermaßen aus:

```
<!ELEMENT didacticalPath (fact,fact?,fact?,fact?,exercise?)>
<!ATTLIST didacticalPath
      levelOfDetail CDATA #REQUIRED
      levelOfDifficulty CDATA #REQUIRED
...
>
```

Dieser besteht danach aus einem obligatorischen und vier optionalen (durch das „?“ gekennzeichneten) `fact`-Elementen sowie einem optionalen `exercise`-Element. Er besitzt u.a. die obligatorischen (`#REQUIRED`) Attribute `levelOfDetail` und `levelOfDifficulty`, die beliebige Zeichenketten sind (`CDATA`). An Stelle von Zeichenketten hätte auch ein diskreter Wertebereich definiert werden können, z.B. in folgender Form:

```
<!ATTLIST didacticalPath
      levelOfDetail (low | medium | high) „medium“
      levelOfDifficulty (easy | medium | difficult) „medium“
      ...
>
```

Hier werden in der Klammer die gültigen Werte und danach ein Standardwert angegeben. Dadurch wird jedoch der Wertebereich bereits in der Grammatik eingeschränkt. Ein Auszug der TeachML DTD ist im Anhang, die aktuelle Version der vollständigen DTD unter [TeachML 99] zu finden. In der aktuellen Version werden inhaltliche und strukturierende Dokumentteile sowie ein einfaches Lernermodell in

TeachML abgebildet. Die konkreten Inhaltselemente werden durch den Import der SMIL DTD in Form von SMIL Dokumenten repräsentiert. SMIL kann selbst wieder HTML Dokumente einbetten. Strukturierende Elemente in TeachML sind z.B. Course, LearnBlock, DidacticalPath, Exercise und SingleTest. Das Lernermodell enthält neben persönlichen Daten eine Liste der Zugriffsberechtigungen auf Kursteile, eine History-Liste der bereits absolvierten Lektionen und Übungen, die Lernziele, Vorkenntnisse und einen persönlichen Pfad als Referenzen auf einzelne Kursbestandteile.
TeachML Kursdokumente enthalten die strukturierten Inhalte sowie Lernermodelle und sind so unabhängig wie möglich von einem konkreten Ausgabeformat, d.h. sie enthalten keine die Präsentation der Inhalte betreffenden Informationen. Dadurch wird einerseits erreicht, daß die Kurse auf verschiedensten Plattformen präsentiert werden können, andererseits lassen sich von einem Kursdokument unterschiedliche Präsentationen erzeugen, z.B. eine akustische für sehbehinderte Lernende oder die Darstellung der Inhalte in Form eines gedruckten Skriptes, als Vortragsfolien für den Tutor oder als virtuelle Lernumgebung für den Schüler. Auf mögliche Realisierungsformen unterschiedlicher Präsentationsformen von Kursdokumenten wird im folgenden Kapitel eingegangen.

3 Dynamische Dokumenterzeugung

Um die Präsentation der Kursdokumente sowohl an die Voraussetzungen, Ziele und den aktuellen Lernfortschritt, als auch an die verfügbare Systemumgebung (z.B. Hardware, Betriebssystem und Netzverbindung) des Lernenden anpassen zu können, ist es notwendig, die Kursdokumente entsprechend den Anforderungen des Nutzers dynamisch zu erzeugen. Hierzu werden u.a. die im Lernermodell verfügbaren Informationen genutzt.

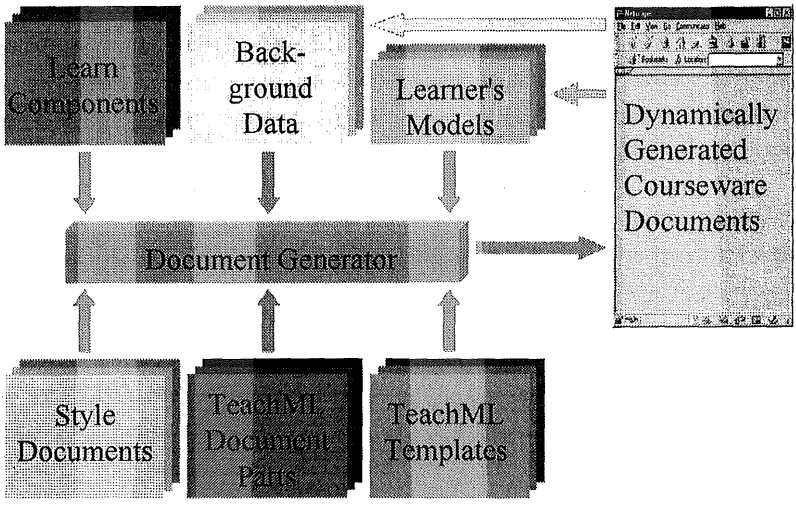


Abbildung 1: Dynamische Generierung von Kursdokumenten

Abbildung 1 zeigt, wie Kursdokumente in CHAMELEON generiert werden. Hierzu werden aus dem Lernermodell (Learner's Model) die für die Generierung relevanten Daten des Lernenden ermittelt. Hintergrunddaten (Background Data) ergänzen dies um technische Informationen, z.B. zur aktuellen Netzwerkbelastung oder bez. der Systemausstattung des Lernenden. Danach werden TeachML Templates als didaktisch und lernpsychologisch strukturierte Vorlagen mit den passenden TeachML Dokumentteilen und den benötigten Lernkomponenten komplettiert. Dazu muß z.B. aus einem TeachML Dokument ein passender didaktischer Pfad gewählt werden. Ein Problem ist dabei, daß die im didaktischen Pfad enthaltenen lokalen Referenzen, z.B. auf den Kurs, zu dem dieser Pfad gehört, verloren gehen. Verallgemeinert man dieses Problem, muß beim Heraustrennen eines Fragments aus einem XML Dokument der Kontext dieses Fragments, also z.B. das umgebende Element, erhalten bleiben. Eine Lösung stellt das vom W3 Konsortium spezifizierte Konzept der XML Fragmente [Grosso 99] dar. Dieser Working Draft beschreibt, wie XML Fragmente aus umfangreichen XML Dokumenten unter Wahrung ihres Kontexts herausgelöst und übertragen werden können.
Damit der Dokumentgenerator die passenden XML Fragmente und Lernkomponenten selektieren und zu einem homogenen Kursdokument zusammenfügen kann, muß ihm eine Beschreibung des Inhalts der Fragmente zur Verfügung stehen. Ein ähnliches Problem wurde bereits in Kapitel 2.1 mit der Semantik von Lernkomponenten angesprochen. Die Grammatik eines XML Dokuments sagt nur etwas über die syntaktische Korrektheit von XML Dokumenten, jedoch nichts über deren Semantik

aus. Z.B. könnte das Element *<img>* sowohl ein Bild als auch eine imaginäre Zahl enthalten. Ziel ist deshalb die Definition eines Schemas von Metainformationen für TeachML Dokumente bzw. Dokumentfragmente. Dadurch lassen sich Dokumente und Dokumentenbestandteile klassifizieren und recherchieren. Dies kann z.B. mit dem speziell für XML Dokumente entwickelten *Resource Description Framework* (RDF) [Brickley 99] geschehen. In ihm wird das Format der Metadaten spezifiziert, das um Kategorien bzw. Beschreibungsvorschriften für das jeweilige Anwendungsgebiet erweitert werden muß.

Ist das konkrete, an den Lernenden angepaßte TeachML Dokument erzeugt, muß es im nächsten Schritt in ein Präsentationsformat überführt werden. Dazu dienen XSL (*eXtensible Style-Sheet Language*) Dokumente [Deach 99]. Sie überführen anhand von Formatobjekten die XML Elemente in ein Ausgabeformat. Im einfachsten Fall wird ein XML Element auf HTML Elementen abgebildet. Andere Ausgabeformate, wie z.B. Postscript oder Audioausgabe sind möglich. Ein Ziel des CHAMELEON Projektes ist die Entwicklung von XSL *Style-sheets*, die TeachML Dokumente in unterschiedliche Präsentationsformate (z.B. Skript, Vortragsfolien oder Online Präsentation) überführen können.

Die Verwaltung von TeachML Dokumenten erfolgt in einer objektorientierten Datenbank unter Verwendung des *Document Object Model* (DOM) [Wood 98]. Dieses beschreibt, wie XML Dokumente in eine äquivalente Objekthierarchie überführt und dort manipuliert werden können. Obwohl DOM prinzipiell für die Entwicklung von XML Werkzeugen (z.B. Parser) konzipiert wurde, ist es auch für die dynamische Selektion und Aggregation von TeachML Fragmenten innerhalb des Dokumentgenerators und als Speicherformat für TeachML Dokumente in einer objektorientierten Datenbank geeignet.

4 Zusammenfassung und Ausblick

Es wurde ein Dokumentmodell für Kursdokumente virtueller Lernumgebungen vorgestellt. Zunächst wurden Kursdokumente in (inter)aktive und passive Bestandteile unterteilt. (Inter)aktive Bestandteile, sog. Lernkomponenten, werden mit Mechanismen der Komponentenarchitektur Java Beans realisiert. An dieser sind Erweiterungen nötig, um semantische Information für Auswahl und Kombination von Lernkomponenten zu integrieren und die funktionale Anpassung und Erweiterung dieser Komponenten zu ermöglichen. Passive Dokumentbestandteile werden in einem Dokumentmodell auf Basis von XML strukturiert. Ein Vorschlag für eine XML Grammatik, TeachML, wurde vorgestellt und auf eine Möglichkeit zur dynamischen Generierung von Kursdoku-

menten aus TeachML Templates und Fragmenten, abhängig von einem einfachen Lernermodell und unter Verwendung von XSL *Style-sheets* näher eingegangen.

Verbunden mit diesem Ansatz sind verschiedene noch zu klärende Fragen. Beispielsweise, wie Lernkomponenten so in TeachML Dokumente integriert werden können, daß sie auf Informationen des sie umgebenden Dokuments zugreifen können. Um die Selektion passender TeachML Fragmente, sowohl durch den Dokumentgenerator bei der Erzeugung von Kursdokumenten als auch durch den Tutor bei der Auswahl anhand spezifischer Kriterien der Zielgruppe und des gewählten didaktischen Konzeptes, zu ermöglichen, muß ein für Kursdokumente geeignetes RDF Schema für Metainformationen entwickelt werden, das mit den semantischen Informationen der Lernkomponenten abgestimmt ist. Für die Anpassung der Kursdokumente an den Lernenden ist eine permanente Aktualisierung des Lernermodells nötig. Diese Informationen können z.B. durch Beobachtung der Interaktionen des Lernenden und Auswertung der Lösung von Übungsaufgaben gewonnen werden. Zu untersuchen ist, anhand welcher Regeln die so erhaltenen Informationen in Parameter des Lernermodell überführt werden können und wie die Informationen des Lernermodells in die Generierung der Kursdokumente einfließen. Weitere interessante Fragen betreffen z.B. die Unterstützung kooperativer Lernszenarien und die Entwicklung von Autorenwerkzeugen für TeachML Dokumente.

5 Literatur

[Alencar 98] Alencar, P.S.C.; Cowan, D.D.; Lucena, C.J.P.; Nova, L.C.M.: A Model for Gluing Components. Proceedings of the 3rd International Workshop on Component-Oriented Programming, Brussels, Belgium, 21 July 1998, Turku Centre for Computer Science

[Bray 98] Bray, T.; Paoli, J.; Sperberg-McQueen, C.M. (Hrsg.): Extensible Markup Language (XML) 1.0 – W3C Recommendation 10-February-1998, http://www.w3.org/TR/REC-xml

[Brickley 99] Brickley, D.; Guha, R.V. (Hrsg.): Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation 03 March 1999, http://www.w3.org/TR/PR-rdf-schema/

[Clark 97] Clark, J.: Comparison of SGML and XML Version 1.0. World Wide Web Consortium Note 15-December-1997, http://www.w3.org/TR/NOTE-sgml-xml-971215

[Deach 99] Deach, S. (Hrsg.): Extensible Style Language (XSL) Specification. W3C Working Draft 21 Apr 1999, http://www.w3.org/TR/WD-xsl/

[Grosso 99] Grosso, P.; Veillard, D. (Hrsg.): XML Fragment Interchange. W3C Working Draft 1999 June 30, http://www.w3.org/TR/WD-xml-fragment

[Harold 98] Harold, E.R.: XML Extensible Markup Language. IDG Books Worldwide Inc., 1998

[Heinem 98] Heinemann, G.T.: Composing Software Systems from Adaptable Software Components. Worcester Polytechnic Institute, 1998

[Hoschka 98] Hoschka, P. (Hrsg.): Synchronized Multimedia Integration Language (SMIL) 1.0 – W3C Recommendation 15-June-1998, http://www.w3.org/TR/REC-smil/

[Kniesel 98] Kniesel, G.: Type-safe Delegation for Dynamic Component Adaptation. Position paper for ECOOP'98 Workshop on Component-Oriented Programming (WCOP'98), 1998

[Lusti 92] Lusti, M.: Intelligente Tutorielle Systeme. R.Oldenbourg Verlag, München, 1992

[Noble 99] Noble, J.; Taivalsaari, A.; Moore, I. (Hrsg.): Prototype-Based Programming – Concepts, Languages and Applications. Springer-Verlag Singapore Pte. Ltd., 1999

[Piemont 99] Piemont, C.: Komponenten in Java – Einsatz und Entwicklung von Java Beans mit Visual Age for Java. dpunkt-Verlag für digitale Technologie, Hüthig GmbH, Heidelberg, 1999

[Raggett 98] Raggett, D.; Le Hors, A.; Jacobs, I. (Hrsg.): HTML 4.0 Specification – W3C Recommendation, revised on 24-Apr-1998, http://www.w3.org/TR/REC-html40/

[Rieder 99] Rieder, R.: Autorensysteme. In Screen Business Online 04/99, MACup Verlag GmbH, 1999, S. 70-74

[Sameting 97] Sametinger, J.: Software Engineering with Reusable Components. Springer-Verlag Berlin Heidelberg New York, 1997

[Schult 99] Schult, T.J.: Lehrzeug – Lernsoftware entwickeln mit ToolBook II Assistant 7. In c't 15/99, Verlag Heinz Heise GmbH & Co KG, 1999, S. 65

[SGML 86] International Standard Organisation, Information Processing – Standard Generalized Markup Language. ISO, Genf, 1986

[TeachML 99] Project CHAMELEON, TeachML DTD. Technische Universität Dresden, Heinz-Nixdorf Stiftungslehrstuhl für Multimediatechnik, 1999, http://www-mmt.inf.tu-dresden.de/projekte/CHAMELEON/TeachML/TeachML.htm

[Teege 97] Teege, G.: Feature Combination: A New Approach to Tailorable Groupware. Proc. Workshop on Tailorable Groupware: Issues, Methods and Architectures, Group97, Phoenix, AZ, 1997

[Weck 98] Weck, W.: Inheritance Using Contracts & Object Composition. In Object Oriented Technology, ECOOP'97 Workshop Reader, Bosch, J.; Mitchell, S. (Hrsg.), Springer Verlag 1998

[Wehner 99] Wehner, F.: Project CHAMELEON. http://www-mmt.inf.tu-dresden.de/projekte/CHAMELEON/

[Welsch 96] Welsch, N.: Entwicklung von Multimedia-Projekten mit Macromedia-Director und Lingo für Macintosh und Windows. Springer Verlag, 1996

[Wood 98] Wood, L. (Hrsg.): Document Object Model (DOM) Level 1 Specification – Version 1.0. W3C Recommendation 1 October, 1998, http://www.w3.org/TR/REC-DOM-Level-1/

6 Anhang – Auszug aus der XML DTD für TeachML

```
<!-- External Entity: import SMIL DTD -->
<!ENTITY    %    smil-document    SYSTEM    "http://www.w3.org/TR/REC-
smil/SMIL10.dtd">
%smil-document;

<!-- Internal Entity Declarations -->
<!ENTITY % name-attr "name CDATA #REQUIRED">
<!ENTITY % descr-attr "description CDATA #REQUIRED">
<!ENTITY % loDetail-attr "levelOfDetail CDATA #REQUIRED">
<!ENTITY % loDifficulty-attr "levelOfDifficulty CDATA #REQUIRED">

<!-- teachml is the root tag -->
<!ELEMENT teachml (author+,learner*,course+)>
<!ATTLIST teachml
      xmlns         CDATA         #FIXED         'http://www-mmt.inf.tu-
      dresden.de/chameleon/teachml/'>

<!-- course is the container for learn blocks -->
<!ELEMENT course (learnBlock*)>
<!ATTLIST course
      %name-attr;
      %descr-attr;>

<!-- learnBlock contains any number of didactical paths -->
<!ELEMENT learnBlock (didacticalPath+)>
<!ATTLIST learnBlock
```

```
      %name-attr;
      %descr-attr;
      authors IDREFS #REQUIRED
      learners IDREFS #IMPLIED>

<!-- didacticalPath with level of detail and level of difficulty -->
<!ELEMENT didacticalPath (fact,fact?,fact?,fact?,exercise?)>
<!ATTLIST didacticalPath
      %descr-attr;
      requiredExperiences IDREFS #IMPLIED
      %loDetail-attr;
      %loDifficulty-attr;>

<!-- exercise contains at least one test -->
<!ELEMENT exercise (singleTest+)>
<!ATTLIST exercise
      %descr-attr;>

<!-- singleTest is a container for different types of tests -->
<!ELEMENT singleTest (multipleChoiceTest|completionTest)>
<!ATTLIST singleTest
      %descr-attr;
      %loDetail-attr;
      %loDifficulty-attr;>

<!-- fact: to be specified, so far an SMIL document or just text -->
<!ELEMENT fact (smil|txt)>
<!ATTLIST fact
      %descr-attr;>

<!-- a learner has personal data, access list and a history -->
<!ELEMENT learner (personalData,accessList,history)>
<!ATTLIST learner
      tutor IDREF #IMPLIED
      internalExperiences IDREFS #IMPLIED
      path IDREFS #REQUIRED
      aims IDREFS #IMPLIED
      %loDetail-attr;
      %loDifficulty-attr;>

<!-- history contains history entries -->
<!ELEMENT history (historyEntry*)>

<!-- history entry with link to the content and a success value -->
<!ELEMENT historyEntry EMPTY>
<!ATTLIST historyEntry
      reference IDREF #REQUIRED
      success (yes|no)'no'>

<!-- personalData contains personal information -->
<!ELEMENT personalData  (physical-address?)>
<!ATTLIST personalData
      title CDATA #IMPLIED
      surname CDATA #REQUIRED
      firstnames CDATA #IMPLIED
      email CDATA #IMPLIED>
```