EPOS: Electronic Publishing mit OODBS und SGML

Einreichung zur Veröffentlichung

Sergej Melnik

melnik@aix550.informatik.uni-leipzig.de

Universität Leipzig, Institut für Informatik

März, 1997

Zusammenfassung

Anhand eines konkreten Einsatzbeispieles (Vorlesungsverzeichnis SS'97[1] des Instituts für Informatik der Universität Leipzig) wird ein auf einer objektorientierten Datenbank und SGML basierendes System für elektronisches Publizieren vorgestellt. Unter Verwendung einer einfachen Substitutionssprache lassen sich Layout-Informationen für SGML-Dokumente kodieren, die in mehreren Formaten (z.B. HTML, LaTeX, ASCII) ausgegeben werden können. Passwortgeschützte Modifizierung von Dokumenten erfolgt direkt im Web. Dank Implementierung in Java ist das System äußerst portabel.

1 Vorwort

Wie jedes Semester, so auch im Sommersemester 1997 galt es, das Vorlesungsverzeichnis des Instituts für Informatik zu erstellen. Üblicherweise enthält ein derartiges Verzeichnis detaillierte Beschreibungen von über 60 am Institut angebotenen Lehrveranstaltungen sowie eine hierarchisch strukturierte Zuordnung der Lehrveranstaltungen zu den einzelnen Studiengängen.
Somit ergaben sich aus dieser Zielstellung die folgenden generellen Anforderungen an die Erstellung und Verwaltung des Vorlesungsverzeichnisses:

- es handelte sich um ein Dokument mit über 30 Autoren, die disjunkte Teilbereiche zu bearbeiten hatten
- das Dokument sollte für alle Interessenten zugänglich gemacht, d.h. im Web veröffentlicht werden

[1] http://www.informatik.uni-leipzig.de/

- für den berechtigten Autorenkreis sollte das Dokument unter Verwendung entsprechender Authentifizierungsmechanismen jederzeit modifizierbar sein, wodurch dessen Aktualität gewährleistet wird
- eine gedruckte Version des Dokumentes sollte automatisch und mit hoher Satzqualität erzeugt werden können
- nicht zuletzt sollte hohe Wiederverwendbarkeit des Systems garantiert werden, da die Gültigkeit des betrachteten Dokumentes zeitbezogen ist. Eine neue Ausgabe wird in regelmäßigen Zeitabständen erstellt.

Diesen Anforderungen konnte die bisherige Verfahrensweise nicht genügen. Existierende Systeme kamen aufgrund ihrer Leistungs- bzw. Kostenmerkmale nicht in Frage, deshalb entschloß man sich für eine Eigenentwicklung des Konzeptes und der Softwarebasis für die Lösung der obengenannten Aufgabenstellung.

2 Übersicht

Das Ziel der Entwicklung des Systems war es, modifizierbare strukturierte Dokumente, wie das obengenannte kommentierte Vorlesungsverzeichnis, mit relativ wenig Aufwand gleichzeitig im Web und auf Papier zu publizieren. Diese komplexe Aufgabe beinhaltet natürlich nicht nur die Formatierung der Dokumente, sondern den gesamten Weg von der Verfassung des Dokumentes bis zur vollständigen Präsentations- und Aktualisierungsfunktionalität.
Das heutige Umfeld des Electronic Publishing (EP) erschien uns in dieser Hinsicht etwas einseitig. Einerseits findet man komplexe vollständige Redaktionssysteme, wie z.B. *SigmaLink* von STEP [STE97], die im Verlagswesen eingesetzt werden und aus finanziellen Gründen und wegen hohem Aufwand für den Bereich der kleineren Publikationen ungeeignet sind. Auf der anderen Seite stehen Ansätze und Produkte, die die gewünschte Funktionalität, d.h. die Web-Anbindung und Modifizierbarkeit, nur bedingt realisieren. Weiterhin wird die Verfügbarkeit der veröffentlichten Dokumenten in existierenden Systemen oft dadurch stark beeinträchtigt, daß man auf eine bestimmte Hard- und Softwaregrundlage setzt.
In unserem Ansatz versuchen wir, diese Lücke zwischen Anwendbarkeit und Komplexität zu schließen. Den primären Anwendungsbereich unseres Systems sehen wir in Publikationen kleineren Umfangs, bei denen die Aktualität und die Verfügbarkeit im Vordergrund stehen. Die Verfügbarkeit wird dadurch gewährleistet, daß ein WWW-Browser die einzige Voraussetzung für die Verwendung des Systems ist, und die Aktualität der Publikationen kommt mittels datenbankgestützen Modifizierungsmechanismen zustande, die ebenfalls vom Browser aus angesteuert werden. Die Architektur des Systems ist schematisch in Abbildung 1 dargestellt und wird im folgenden ausführlicher

skizziert.

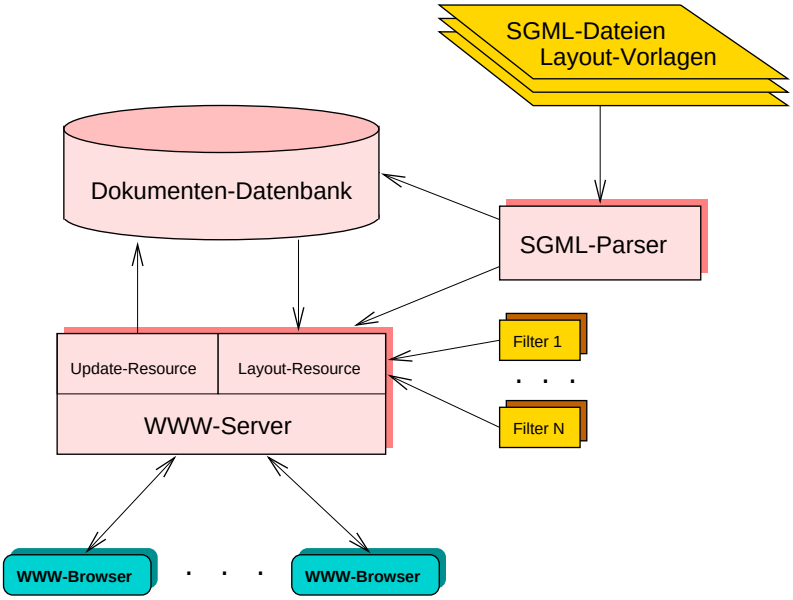

Abbildung 1: Architektur des Systems

Als Basis für die Verfassung von strukturierten Dokumenten in EPOS dient SGML (Standard Generalized Markup Language, ISO Standard 8879:1996) [Szi95]. SGML stellt eine Methode zur Verfügung, die die Informationen von den Abhängigkeiten der eingesetzten oder einzusetzenden Anwendungsprogramme befreit. So kann der Datenbestand in verschiedenen Applikationen verarbeitet und in den unterschiedlichsten Repräsentationsformen weitergegeben werden. Viele Verlage haben bereits begonnen, SGML einzusetzen, beispielsweise im Bereich von Enzyklopädien, Wörterbüchern, Nachschlagewerken, technischen Dokumentationen und wissenschaftlichen Publikationen. Die Struktur der Information wird mittels einer sogenannten Dokument-Typ-Definition (DTD) festgelegt, die applikationsneutral und prüfbar ist. SGML bietet mit Kenntnis der Strukturen und Elemente innerhalb der Dokumente die Methode, nicht nur gesamte Dokumente, sondern die darin enthaltenen Informationseinheiten zu verwalten.

Der Verwaltungsaspekt von SGML-basierten Dokumenten wird bereits seit einer relativ langen Zeit erforscht. In Produktionsumgebungen, wo die Verläßlichkeit und das Zeitverhalten das Kriterium Nummer Eins sind, ist es üblich, für die Speicherung von SGML-Dokumenten relationale Datenbanken einzusetzen, wobei sich der Schwerpunkt mit weitergehenden Erfahrungen [Böh95] immer mehr in den Bereich der objekt-orientierten Datenbanken verschiebt, die sich prinzipiell für die Verwaltung von strukturierten Informationen besser eignen [Loo95]. Aus diesem Grund haben wir eine objekt-orientierte Datenbank (ObjectStore PSE for Java [Obj97]) eingesetzt. Dadurch läßt sich die Modularität des Dokumentenaufbaus voll ausnutzen. Außer schnellem Retrieval und Editierbarkeit der Dokumententeile erübrigt sich damit das mehrmalige Parsen von SGML-

Quellen. Nach dem Importieren in die Datenbank können die SGML-Elemente ohne weiteres sofort manipuliert werden.
Anders sieht es mit der Formatierung von SGML-Dokumenten aus. Einer der entscheidenden Vorteile von SGML besteht darin, daß die medienabhängige Präsentation leicht an die Struktur des Dokumentes geknüpft werden kann, ohne die Informationen selbst zu verändern. So können die Informationen zu jeder Zeit ohne Änderungen mit der neuesten Technologie bearbeitet werden. Allerdings ergeben sich mit dem Etablieren der neuen Medien auch immer neue Anforderungen an die Formatierungsstandards und -werkzeuge. Es entstehen neue Spezifikations- und Transformationssprachen, deren Mächtigkeit von ganz einfachen Substitutionsanweisungen bis zu einer vollständigen Programmiersprache reicht. Bekannte Beispiele sind in diesem Kontext

- die Substitutionssprache des Amsterdam SGML Parsers [WE89], die im Linux Documentation Project (LDP) in der Applikation Linuxdoc-SGML [WH96] verwendet wird,
- das FOSI (File Output Specification Instance), das sich bei der Erzeugung von Druckversionen durchgesetzt hat,
- Cascading Style Sheets (CSS1) vom W^{3}C-Konsortium [WK97], die das Make-up von HTML Dokumenten verbessern sollen und nicht zuletzt
- das DSSSL (Document Style Semantics and Specification Language, ISO/IEC 10179:1996) [ISO96], welches zur Zeit wahrscheinlich die vielversprechendste Entwicklung darstellt.

Da, wo sich hohe internationale Gremien nicht entscheiden können, welche Spezifikationssprache die am besten geeignete für die Formatierung von SGML-Dokumenten ist, werden wir wahrscheinlich auch keinen Durchbruch schaffen. In erster Linie geht es uns darum, eine brauchbare Lösung bereitzustellen, die einen Kompromiß zwischen der Komplexität und der Ausdrucksfähigkeit der Spezifikationssprache darstellt. Als allgemeiner Rahmen ist die Einbindung von *Filtern* für verschiedene Spezifikationssprachen vorgesehen. Solche Filter sind in einer Programmiersprache geschriebene Module, die bei Vorgabe einer SGML-Datei (die eventuell in eine Datenbank importiert wurde) und einer Layout-Vorlage die Formatierung des Dokumentes entsprechend der verwendeten Spezifikationssprache vornehmen.
Von uns wurde daher ein Filter für eine einfache Substitutionssprache entwickelt. Sie enthält zwar keine programmiersprachen-ähnlichen Anweisungen wie `for`, `if`, `while` usw., erlaubt aber eine Formatierung des Dokumentes für mehrere Ausgabeformate wie z.B. HTML, LaTeX, RTF, ASCII mit ISO-8859-1, Groff oder GNU-Info, basierend auf dem Konzept von Sichten oder Views (siehe Abschnitt 4).
Viel Wert wurde dabei auf die schnelle Erlernbarkeit dieser Sprache gelegt, die eigentlich nur drei Substitutionskonstrukte beinhaltet. Da wir hauptsächlich kleinere Publikationen (in einem

Umfang von einigen hundert Seiten) im Anwendungsvisier hatten, galt für uns das Ziel, die Layout-Vorlagen mit geringem Zeitaufwand anfertigen zu können. Einige praktische Erfahrungen werden im Abschnitt 9 angesprochen.
Soll das Dokument im Web verfügbar sein, ist eine Konvertierung in HTML unumgänglich. Alternative Formatierungsmöglichkeiten (grafische Ausgabe mittels Java, Plug-Ins, ActiveX usw.) sind notwendigerweise mit dem Ausschluß eines nicht vernachlässigbaren Nutzerkreises verbunden und damit zur Zeit weniger akzeptabel. Es ist also eine Translation von SGML in HTML notwendig. Bei der Erstellung einer druckfertigen Ausgabe finden wir es sinnvoll, auf vorhandenem Fundament aufzubauen und anstatt direkter Erzeugung einer Postscript- oder PDF-Version eine intermediäre generische Markupsprache wie LaTeX oder Groff einzusetzen. Dadurch wird der Übersetzungsaufwand drastisch reduziert.
Wie aus der Abbildung 1 ersichtlich ist, erfolgt die Web-Anbindung von Dokumenten mittels speziellen Modulen für den Web-Server. Als Web-Server verwenden wir Jigsaw, den Server des W^3C-Konsortiums [WK97], der ebenfalls komplett in Java geschrieben wurde. Das Update-Modul führt Änderungsoperationen an der Datenbank aus, während das Layout-Modul anhand von vorgegebenen Filtern die eigentliche Ausgabeformatierung bewerkstelligt.

3 Dokument als Sammlung von persistenten Objekten

Was bei Datenbanken längst akzeptiert ist, *Datenstrukturierung*, *Redundanzfreiheit*, *zentraler Zugriff*, *benutzerspezifische Views*, *Normalisierung*, wird für Dokumente kaum angewandt. Eine konventionelle Lösung beim Publizieren im Web ist es immer noch, das gesamte Dokument mittels eines gängigen Textverarbeitungsprogramms (z.B. MS Word) oder eines Satzsystems (z.B. LaTeX) zu verfassen und anschließend nach HTML zu konvertieren. Dabei kann man durchaus beeindruckende Ergebnisse erzielen. Besitzt das Dokument jedoch eine ausgefallenere Struktur, die im allgemeinen einem Graphen isomorph ist, reicht eine sequentielle Sichtweise nicht mehr aus. Um die Querverweise einzelner Dokumententeile in den Griff zu bekommen, würde man ein WWW-Autorenwerkzeug oder einen Web-Publisher einsetzen. Dies wiederum erschwert die Erzeugung einer sequentiellen Version des Dokumentes, weil sich im allgemeinen der logische Aufbau dessen von der hyperlink-basierten Form unterscheidet.
Die Redundanz stellt ein weiteres Problem dar. Nehmen wir als Beispiel das Vorlesungsverzeichnis: verschiedene Elemente der Veranstaltungsbeschreibung (Name, Dozent, Typ usw.) erscheinen im Inhaltsverzeichnis, im Stundenplan, in der Gesamtbeschreibung und in der Studiengangaufteilung sogar mehrfach, da einige Vorlesungen für mehrere Studiengänge angeboten werden. Wollte ein Dozent den Namen seiner Lehrveranstaltung ändern, müßte die Aktualisierung an allen betroffenen Stellen des Dokumentes vorgenommen werden. Eine gleichzeitige Änderung stellt somit ein nicht zu

unterschätzendes Konsistenzproblem dar. Zeichenerklärung oder Impressum, die auf jeder HTML-Seite erscheinen sollen, verursachen analoge Probleme, die sich nur mit erheblichen Cut-and-Paste–Aufwand bewältigen lassen.

Überdies ist die Frage der Editierbarkeit des Dokumentes zu lösen, die "online" von mehreren Benutzern simultan erfolgen soll. Liegt das Dokument in einer Einzeldatei vor, löst dessen Bearbeitung von einem Benutzer eine Schreibsperre für die anderen aus, sofern kein Versionsmanagement benutzt wird. Durch einen modularen Aufbau des Dokumentes kann man Abhilfe schaffen.

Der Kern unseres Ansatzes besteht darin, ein Dokument als eine strukturierte Sammlung von persistenten Objekten aufzufassen, die in einer Datenbank untergebracht werden. Solche Objekte (z.B. Abschnitte, Elemente einer Aufzählung) besitzen bestimmte Attribute, wie z.B. den Titel eines Kapitels, können mit anderen Objekten in verschiedenen Relationen stehen und sollen imstande sein, sich dem Benutzer in der einen oder anderen Form zu präsentieren. So muß z.B. ein Objekt vom Typ *Kapitel* "wissen", daß es seinen Titel fett und in einer bestimmten Schriftgröße anzuzeigen und seine Unterabschnitte aufzufordern hat, sich der Reihe nach darzustellen.

Der entscheidende Vorteil des Aufbaus eines Dokuments als eine strukturierte Objektsammlung liegt nach unserer Meinung im verwaltungstechnischen Aspekt, in erster Linie im Hinblick auf die Editierbarkeit. Wie bereits angesprochen, lassen sich kleine Dokumentenportionen mit wenig Aufwand modifizieren, disjunkte Teile können gleichzeitig ohne Konsistenzprobleme und unnötigem Ressourcenverbrauch bearbeitet werden. Im Einbenutzerbetrieb triff dieses Argument ebenfalls zu (wer bereits eine über 20 MB große monolitische Datei in seinen Editor geladen hat, kann das nachempfinden). Diese Idee spielt bereits seit Jahren eine herausragende Rolle in der Entwicklung des OpenDoc und CORBA-Standards für zusammengesetzte Dokumente [Sie96].

Es bleibt immer noch zu klären, wie die logische Dokumentenstruktur im Rahmen dieses Ansatzes erfaßbar ist, und wie die Layoutformatierung realisiert werden kann.

Um eine Abbildung zwischen dem Quellformat eines Dokumentes und einer Kollektion von persistenten Objekten zu schaffen, verwenden wir SGML (siehe Abschnitt 2). Dabei werden den Objekten entsprechende SGML-Elemente zugeordnet, die Instanzen eines bestimmten Typs darstellen. Ist diese Betrachtungsweise überhaupt adäquat? Einige Probleme bereiten z.B. die Einbettung der Graphenstruktur des Dokumentes in einen streng hierarchisch aufgebauten SGML-Baum, sowie die Verwaltung von multimedialen Datentypen. Trotz dieser Schwierigkeiten bringt das Wesen von SGML viele Vorteile, die bereits erwähnt wurden. Unter anderem waren das die strenge Typisierung von Dokumenten, scharfe Trennung zwischen Inhalt und Layout, sowie die breite Akzeptanz von SGML als Austauschformat. Eliminierung von Formatierungsangaben im Quelltext erweist sich insbesondere dann als vorteilhaft, wenn man mehrere Ausgabeformate bzw. -medien einsetzt. Einen weiteren Pluspunkt bringt die Möglichkeit einer inhaltsbasierten Suche. Mit wachsender Datenflut wird es immer schwieriger, den gesuchten Ausschnitt aus riesigen Dokumen-

ten(sammlungen) in kürzester Zeit sicher wiederzufinden. Es ist hilfreich, wenn das Ergebnis der Suche in einem Archiv beispielsweise durch die Beschränkung auf das Vorkommen im Titel präzisiert werden kann. Wenn man den Dokumententyp kennt, lassen sich also sehr genaue Suchanfragen formulieren.

Insgesamt betrachten wir hier SGML als eine Abstrahierung von der zugrundeliegenden Objektstruktur, als ein ASCII-Äquivalent der internen objektorientierten Repräsentation des Informationsgehalts.

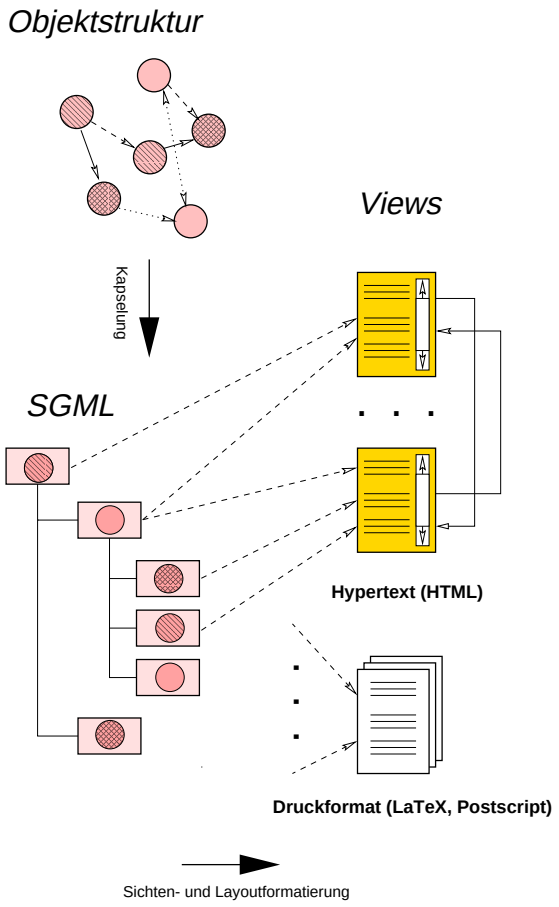

Abbildung 2: Zusammenhang von Objekten, SGML und Views

Wie bereits angesprochen, wird das Objektverhalten, insbesondere das Layout einzelner SGML-Elemente, mit Hilfe einer einfachen Skriptsprache festgelegt. Diese kann in SGML gekapselt und uniform abgespeichert werden. Im Unterschied zu Cascading Style Sheets Level 1 (CSS1) vom W^3C-Konsortium [WK97] beschränkt sich die Beschreibung der Objekte nicht nur auf das Make-up, d.h. Schriftart, Farbe usw. Unsere Herangehensweise bietet außerdem weitergehende Flexibilität der Gestaltung der Dokumentenstruktur, die auf logischen Sichten (Views) auf das Dokument basiert. Der Zusammenhang zwischen persistenten Objekten, SGML und Views ist in

Abbildung 2 wiedergegeben. Das Sichtenkonzept oder Views, kombiniert das datenbankgestützte Dokument-Skelett mit der Layoutformatierung. Die nächsten zwei Abschnitte beschreiben eine mögliche konzeptuelle Grundlage dieser Verknüpfung.

4 Die Views

Benutzerspezifische Views gehören seit langem zur Standardausrüstung beim Design einer Datenbankanwendung. Dieses Konzept versuchen wir auf strukturierte Dokumente zu übertragen, die als eine hierarchisch aufgebaute Datenbank aufgefaßt werden können. In unserem Ansatz basiert das Sichtenkonzept auf der Tatsache, daß die interne Struktur des Dokumentes mit der, die dem Leser präsentiert werden soll, im allgemeinen nicht übereinstimmt. Ein Beispiel dafür sind die Fußnoten, die Anmerkungen zu einem bestimmten Sachverhalt darstellen. Mit diesem inhaltlich verknüpft, erscheinen sie jedoch in einer Druckversion im allgemeinen unter der Schriftseite und in einem Hypertext als Querverweise auf den Fußnotentext.
Bei einem komplexen Aufbau eines Hypertextes tritt das Problem auf, daß man verschiedene Dokumententeile auf unterschiedliche Weise kombinieren muß, um die gewünschte Präsentation des Dokumentes zu ermöglichen. So stellt man dem Benutzer üblicherweise einen Mechanismus zur Verfügung, zwischen differenzierten Detailierungsstufen und Dokumententeilen zu navigieren, was z.B. ein Inhaltsverzeichnis sein kann. Das Inhaltsverzeichnis ist allerdings nichts anderes als eine Sicht auf das Dokument, in der die Darstellung von Abschnitten auf ihre Überschriften reduziert wird. Bei sehr ausführlichen Inhaltsverzeichnissen braucht man wiederum verschiedene Sichten darauf, um dem Leser einen besseren Überblick über die Struktur des Gesamtdokumentes zu verschaffen.
Die Views beschränken sich in unserer Interpretation nicht nur darauf, den Detailiertheitsgrad der Betrachtung zu spezifizieren. Es handelt sich um einen Querschnitt des Dokumentes, in dem die relevanten Teile zusammengeführt werden (siehe Abbildung 2). Die Bildung dieses Querschnitts basiert auf den Hilfsparametern Format, Modus und Kontext.
Durch das Format wird das Ausgabemedium spezifiziert. Ein Format in diesem Sinne kann angeben, daß die Ausgabe z.B. in HTML, LaTeX oder BibTeX erfolgt, oder daß sie für ein Information Retrieval-System bestimmt ist (siehe Abschnitt 8). Die Formate sind nicht vordefiniert, sie werden entsprechendend den Bedürfnissen des Benutzers festgelegt. Mittels Formatspezifikation kann man unter anderem eine einheitliche Behandlung von Spezialsymbolen und Satzzeichen ermöglichen. Dazu zählen beispielsweise die Umlaute. Das Format stellt also einen Rahmen bereit, in dem die eigentliche Sichtenerzeugung abläuft. Es bezieht sich nicht nur darauf, festzulegen, daß das Fettgedruckte mit HTML-Tags oder LaTeX-Markup hervorgehoben wird. Das Format ist auch dann ausschlaggebend, wenn man aufgrund der Mediumspezifikation zu entscheiden hat, ob ein

sequentielles oder ein hyperlink-basiertes View generiert werden soll.
Damit jede Sicht nach Maß zugeschnitten werden kann, soll die Sichtenbildung so gestaltet werden, daß beliebige Dokumententeile in eine Sicht einbezogen werden können. Das kann dadurch erreicht werden, daß das Dokument jedesmal hierarchisch, ausgehend von dem Top-Element des SGML-Baumes, abgearbeitet wird. Mit Hilfe von Modus, dem Elementtyp des darzustellendes Objektes, wird das Verhalten von anderen relevanten Dokumententeilen beeinflußt. So wird z.B. der Zeichenerklärung im Modus `Studiengang` "bewußt", daß sie ebenfalls in diese Sicht gehört. Da ein bestimmtes Element an verschiedenen Stellen des Dokumentes referenziert werden kann, ist eine Verfeinerung des Modus notwendig. Diese wird durch den Kontext realisiert. Der Kontext einer Instanz I_x ist definiert als ein beliebiger Teil des Pfades $I_1, \ldots, I_x$ von der Top-Instanz des Baumes bis zur Instanz I_x. Damit läßt sich z.B. unterscheiden, ob eine Vorlesung innerhalb einer Studiengangbeschreibung oder im Stundenplan referenziert wurde, um danach die Formatierungsanweisungen zu richten.
Die Kodierung der Sicht- und Layoutformatierungsanweisungen besteht im wesentlichen in der format-, modus- und kontextspezifischen Zuordnung zu den im Dokument verwendeten Elementen. Der nächste Abschnitt befaßt sich mit der Layoutspezifikation und der zugrundeliegenden Skriptsprache.

5 Die Layoutformatierung

Im vorangegangenen Abschnitt ist eine Vorgehensweise beschrieben worden, wie einzelne Teile des Dokumentes zu unterschiedlichen Views zusammengefaßt werden können. Diese Betrachtung läßt sich in das objektorientierte Paradigma, von dem wir ausgegangen sind, wie folgt einbetten:

- Polymorphie der Objekte wird durch Format, Modus und Kontext wiedergespiegelt. So werden z.B. für das Objekt "Kapitel" je nach gewünschter Sichtweise des Benutzers entweder nur die Hyperlinks zu den entsprechenden Unterabschnitten angeben oder deren vollständiger Inhalt.

- Vererbung erübrigt sich, weil keine Methoden im üblichen Sinne existieren. Die Formatierungsanweisungen lassen sich in verschiedenen Elementen mit bestehenden Mechanismen wiederverwenden.

- Kapselung wird dadurch modelliert, daß nur das Verhalten von Nachfolge-Objekten in der Objekthierarchie direkt beeinflußt werden kann.

Wie läßt sich das Verhalten der Objekte beschreiben? Im Rahmen des Metamedia-Projektes [McC97] an der Waterloo University ist die Idee entwickelt worden, das Verhalten der Objekte

direkt in einer Programmiersprache (Java) zu kodieren. Trotz der ungeheueren Flexibilität dieses Ansatzes ergeben sich Probleme, zum einen durch die erhöhte Komplexität der Dokumentenbeschreibung, die einer expliziten Programmierung bedarf, zum anderen durch den aufgrund der zusätzlich notwendigen Compilierung verlängerten Entwicklungsprozeß, und durch die Tatsache, daß Dokumente wie Applikationen ausgeführt werden müssen. Außerdem war die Ausgabe auf Grafik unter Verwendung eines visuellen Markups fokussiert. Unsere Alternative zur Kodierung des Objektverhaltens ist eine möglichst einfache Substitutionssprache, in der abwechselnd formatbezogene Formatierungsanweisungen (z.B. HTML-Elemente) und Referenzen auf Dokumentinstanzen vorkommen.

Wie bereits angesprochen, werden etliche Layoutspezifikationen vom Inhalt des Dokumentes getrennt. Trotzdem muß es einen Weg geben, dem Dokument eine bestimmte Layoutvorlage zuzuweisen. Dazu wird ein Filter, also eine Java-Klasse spezifiziert, der für die eigentliche Formatierung zuständig ist (siehe Abbildung 1). Dieser kann sowohl lokal als auch im Internet auffindbar sein. Dem Filter steht die Aufgabe zu, eventuelle weitere Parameter auszuwerten und anhand dieser alle Formatierungsinformationen zu gewinnen. Derzeit ist nur ein einziger Filter verfügbar, dessen Konzeptgrundlage (Views) in vorangegangenen Abschnitten angesprochen wurde. Jedoch läßt diese generelle Vorgehensweise einen breiten Spielraum für eine Vielfalt von unterschiedlichen Dokumentenfiltern. Außerdem ist es auch denkbar, daß man die Formatierungsangaben zum Dokument von diesem räumlich trennt, und nur den reinen SGML-Inhalt an die Benutzer ausliefert, die das Dokument lokal verfügbar haben möchten, wobei die Filter bei Bedarf nachgeladen werden. In diesem Framework läßt sich das Modell vom Metamedia-Projekt wiederfinden, wenn man alle Layoutanweisungen in Java programmiert und über einen geeigneten Filter verfügbar macht.Der nächste Abschnitt konzentriert sich darauf, wie Dokumente verwaltet werden können und wie sie ihren Weg in die Datenbank hinein und aus der Datenbank heraus ins Web finden.

6 Aus der Datenbank ins Web

Die komplexe Aufgabe der Veröffentlichung von modifizierbaren strukturierten Dokumenten beinhaltet, wie gesagt, nicht nur die Formatierung der Dokumente, sondern den gesamten Weg von der Verfassung des Dokumentes bis zur vollständigen Präsentations- und Aktualisierungsfunktionalität. Bevor ich auf den Lebenszyklus des Dokumentes eingehe, möchte ich noch etwas Licht auf die Architektur des Gesamtsystems werfen (wie in Abbildung 1 dargestellt). Zwei zentrale Fragen waren zu entscheiden:

1. wie die Speicherung der Dokumente im Hinblick auf effiziente Modifizierbarkeit erfolgt, und
2. ob die HTML-Formatierung auf der Client- oder Server-Seite bewerkstelligt wird.

Die erste Version des Systems war dateisystem-orientiert. Der Hauptvorteil sollte eine einfachere Integration ins traditionelle dateiorientierte Web sein. Dieser Ansatz verfügte über eine Reihe von Schwächen und warf alsbald ernste Probleme auf. Da die Dokumente in Quellform, in SGML also, vorlagen, war bei jeder Ausgabe ein Parsing-Vorgang notwendig. Trotz des Caching von vorgeparsten Dateien stieg die Antwortzeit mit der Dokumentengröße linear. Mit noch schwerwiegenderen Problemen war die Modifikation von Dokumenten verbunden. Zwecks Vergabe von Zugriffsberechtigungen war eine Aufsplittung des Dokumentes in viele kleinere Module notwendig. Dadurch erhöhte sich der Verwaltungsaufwand dramatisch. Wesentlich vielversprechender sah dagegen die Speicherung in einer objektorientieren Datenbank, gekoppelt an die Auffassung eines Dokumentes als eine Sammlung von persistenten Objekten (siehe Abschnitte 2 und 3), aus. Neben der Abhilfe für die obengenannten Probleme schafft dieser Ansatz Voraussetzungen für die Interoperabilität mit anderen Systemen z.B. über eine CORBA-Schnittstelle [Sie96].

Nichtzuletzt spielt dieser Aspekt auch bei der Entscheidung, ob man die Formatierungsaufgabe auf den Client verlagert, eine Rolle. In diesem Falle wäre effizientes Manipulieren von Objekten eines lokalen Dokumentes von dem Client aus vonnöten, was ebenfalls unter CORBA-Einsatz möglich wäre. Derzeit wird an der Entwicklung in diese Richtung gearbeitet; bis dahin erfüllt der Server alle Formatierungsaufforderungen.

Wie bereits gesagt, ist das gesamte System vollständig in Java geschrieben. Um die Formatierungswerkzeuge an einen Web-Server anzubinden, wird der Java–Web-Server Jigsaw [WK97] benutzt. Die Anbindung erfolgt mittels Einsatz von sogenannten Jigsaw-Ressourcen, die mit Sun's Servlets vergleichbar sind. Es werden zwei allgemein ausgelegte Ressourcen (oder Module) für jeweils die Ausgabe und die Aktualisierung der Dokumenteninhalte verwendet. Die Aktualisierung ist nur für die in eine Datenbank importierten Dokumente möglich, während die Ausgabe auch aus einer SGML-Datei direkt generiert werden kann, um eine inkrementelle Bearbeitung von Dokumenten zu erleichtern. In diesem Fall erübrigt sich die Aufnahme des Dokumentes in die Datenbank. So kann der Benutzer an einer SGML-Datei schreiben und ab und zu den Reload-Button seines Browsers betätigen, um eine formatierte Version gleich darzustellen.

Die Referenzierung innerhalb eines Dokumentes basiert direkt auf seiner hierarchischen Struktur. Um einen Unterbaum eindeutig zu identifizieren, sammelt man alle Elementennamen beim Traversieren des Baumes zum gewünschten Element. Damit hat man zunächst einen Pfad, wie z.B. `DOCUMENT.TITELSEITE.AUTOR`, der aber nicht notwendigerweise eindeutig ist. Damit man gleichnamige Elemente auf derselben Hierarchieebene unterscheiden kann, spezifiziert man eine Gruppe von Parametern dieser Elemente, die analog zu dem Primärschlüssel einer Tabelle die Eindeutigkeit des Datensatzes garantieren. Solche Elemente sind z.B. Kapitel, Abschnitte, Absätze usw. In solchen Fällen könnte der Pfad wie folgt aussehen: `DOCUMENT.SECTION|ID=VORWORT`.

Nun möchten wir das typische Nutzungsszenario des Systems etwas näher unter die Lupe nehmen. Dabei gehen wir davon aus, daß das Zieldokument bereits in der einen oder anderen digitalen Form vorliegt. Als erstes muß das Dokument nach SGML konvertiert werden. Verfügt der Benutzer über ein entsprechendes Information Reengeneering-Werkzeug, wie z.B. OmniMark von Exoterica, einen der leistungsfähigsten und flexibelsten SGML-Konverter auf dem Markt, so kann dieser Schritt weitgehend automatisiert werden. Anderenfalls ist es erforderlich, die Dokumentenstruktur zu analysieren und diese explizit festzulegen. An dieser Stelle bieten sich zwei unterschiedliche Vorgehensweisen an. Legt man Wert auf die Dokumententypisierung, so schreibt man eine DTD (Document-Typ-Definition) für das zu publizierende Dokument und läßt es von einem SGML-Parser als ASCII-Repräsentation seiner Elementenstruktur ausgeben (Element Structure Information Set). Spielt die Typisierung eher eine untergeordnete Rolle, beispielsweise wenn das Dokument einmalig ist, kann man sich die DTD sparen, muß aber gezwungenermaßen auch bestimmte Konventionen einhalten (wie z.B. die explizite Angabe aller schließenden Tags).
Der nächste Schritt ist die Erstellung einer Layout-Vorlage. Wird das Dokument nicht "portiert", sondern ist von Anfang auf dieses System ausgerichtet, kann man die Layout-Vorlage mit der Inhaltsstruktur auch parallel entwickeln. Generell wird ein Werkzeug zur Verfügung gestellt, mit dem man das Dokument in allen unterstützen Formaten ausgeben kann. Ist das Zielformat das HTML, lassen sich die Ergebnisse, wie oben erwähnt, sofort im Browser anzeigen.
Wenn das Dokument in seiner Layout-Vorlage Modifikationsmöglichkeiten vorsieht, ist dessen Import in die Datenbank notwendig. Außerdem beschleunigt sich dadurch die Ausgabe erheblich. Die Anzahl der Dokumente in der Datenbank ist nur durch ihre maximale Größe begrenzt. Damit ist die Veröffentlichung des Dokumentes vollzogen. Der nachfolgende Abschnitt beschäftigt sich mit der Gestaltung der Modifizierungsmechanismen.

7 Modifizierbarkeit und Authentifizierung

Bei der Aufbereitung eines Dokumentes, das von vielen Benutzern direkt im Web editiert werden soll, kann bis auf einen (eventuell Java-fähigen) Web-Browser weder eine einheitliche Hard- noch Softwarebasis vorausgesetzt werden. Das Spektrum reicht von Unix-Workstations über Intel-Plattformen mit MS-Windows bis zum Macintosh. Trotzdem muß man allen Nutzern einen Schreibzugriff auf die von ihnen verwalteten Dokumententeile ermöglichen. Die ideale Lösung — die Anbindung eines SGML-Editors — kommt zur Zeit wegen vielfältigsten Zielplattformen und fehlenden Standardschnittstellen also nicht in Frage. Eine sehr tragfähige Variante dieser Anbindung kann mittels Java-CORBA-Kombination realisiert werden. Java-Module bringen die Plattformunabhängigkeit mit sich, wobei CORBA für das einheitliche Interface zum entfernten Dokument sorgt. An dieser Kopplung wird derzeit weitergearbeitet. Da aber eine akzeptable Lösung schnell

bereitgestellt werden sollte, blieb nur eine einzige Implementierungsmöglichkeit: die HTML-Forms. Es stellte sich die Frage, wie man ohne zusätzliche Programmierung die Bearbeitung von Eingabemasken vornehmen kann. Man möchte doch sehr ungern für jedes editierbare Dokument ein zusätzliches CGI-Skript anfertigen. Doch bevor die Eingabemasken ausgewerten werden können, müssen sie erst einmal generiert werden. Hier kommt wieder das Sichtenkonzept ins Spiel. Und zwar, läßt sich eine solche Maske wie eine entsprechende Sicht auf das Dokument im Blickwinkel der Aktualisierung auffassen. Konkret wird das mittels des Parameters Format gelöst. Auf diese Weise können die Eingabemasken direkt in der Layout-Vorlage mitkodiert werden. Besitzt der zu modifizierende Teil des Dokumentes eine unkomplizierte Struktur, so kann dem Benutzer durch die Maskenaufteilung in Felder verborgen bleiben, daß dahinter ein SGML-Dokument steckt. Ist die Struktur ausgefeilter, hat der Nutzer auch einen breiteren Spielraum, indem er SGML-Markup innerhalb von Forms verwenden kann.

Für die Auswertung von Masken ist das zugehörige Update-Modul verantwortlich. Mittels Maskenparameter wird die Ressource u.a. darüber informiert, welches Teil von welchem Dokument mit gegebenen Daten aktualisiert werden soll. Die Vergabe von Zugriffsberechtigungen erfolgt mittels Basic Authentication (HTTP 1.0), die von dem Web-Server Jigsaw unterstützt wird. Jeder geschützte Bestandteil des Dokumentes wird mit einer Update-Ressource verknüpft. Diese wird über Web-Server–Konfigurationsmenüs so vorkonfiguriert, daß sie ausschließlich Aktualisierungsanfragen für einen festgelegten Teil des SGML-Baumes des Dokumentes akzeptiert. D.h. nur über diese Ressource läßt sich ein gegebenes Dokumententeil editieren. Anschließend wird dieser Ressource ein sogenannter Authentifizierungs-Bereich zugeordnet. Dadurch erreicht man, daß nur validierte Benutzer, die im entsprechenden Bereich eingetragen sind, auf die Ressource und somit auf das betroffene Teil des Dokumentes Zugriff haben (vgl. mit *Rollen*-Begriff [STE97]).

8 Zusätzliche Features

Besitzt die Dokumenten-Datenbank einen größeren Umfang, muß sie an ein Information Retrieval-System (IRS) geknüpft werden, damit man die Nadel, den gesuchten Dokumententeil, in einem Heuhaufen finden kann. Wenn das IRS die Dokumentenstruktur zur Qualifizierung der Recherche heranziehen kann (wie z.B. OpenText 5), ist es dem Benutzer möglich, die Struktur meist ohne Kenntnisse in die Suche einbinden. Unterstützt das zugrundeliegende IRS aber "nur" Volltext-Retrieval, läßt sich in EPOS die Kopplung der Dokumenten-Datenbank an das IRS auch mit Hilfe der Ausgabe in ASCII-Format bewerkstelligen. Damit entfällt für das IRS das nochmalige Parsen von HTML-Dateien und die Entschlüsselung von HTML-spezifischen Symbolen. Aber auch beim Einsatz von Systemen, die eine intelligente HTML-Vorbearbeitung vornehmen, stößt man auf Unbequemlichkeiten. Zum einen enthalten manche Dokumente Textteile, die nicht immer indexiert

werden sollen, wie z.B. Tabellen mit statistischen Daten. Zum anderen enthält eine HTML-Seite (sprich, ein spezifisches View) viele Informationen, die bei ihrer Aufnahme in die Indexierung eine Verschlechterung des Suchergebnisses (genauer, der Precision) bewirken können. Das sind beispielsweise sich von Seite zu Seite wiederholende Textteile wie das Impressum, Struktur-Indikatoren wie Überschriften von höheren Ebenen usw. All das läßt sich bei der Sichtenspezifikation für die Indexierung genau anpassen, indem man bestimmte IRS-Views definiert.
Zum Testzweck wurde eine exemplarische Datenbank an das IRS von dem SQUIRREL-System gebunden [MB96]. Wiederum lag die prinzipielle Schwierigkeit in der fehlenden Umgebung für verteilte Objekte. Damit die Aktualisierung der Index-Tabellen automatisch vorgenommen werden kann, ist eine Meldungsstrategie erforderlich, die es dem IRS erlaubt, die geänderten Inhalte gezielt zu reindexieren. Anderenfalls wird man gezwungen, einen netzlastverursachenden Index-Roboter einzusetzen.
Zur Erleichterung des Daten-Austausches sowie zusätzlicher Datenbank-Sicherung wurde eine Archivierungsstrategie implementiert. Bei der Archivierung werden modifizierte Teile der Datenbank in SGML exportiert. Die automatische Archivierung läßt sich mittels bestimmter Parameter der Eingabemasken steuern. Damit steht jederzeit eine aktuelle SGML-Version des Dokumentes zur Verfügung und kann zum Daten-Austausch verwendet werden. Außerdem werden die Daten bei der Zerstörung der Datenbank verschont.

9 Zusammenfassung und Erfahrungen

Das vorliegende System ermöglicht eine Publikation von beliebigen SGML-Dokumenten im Web und als Papierversion auf der konzeptuellen Grundlage von Views und persistenten Objekten (siehe Abschnitte 3 und 4), wobei die Layout-Vorlagen vom Inhalt getrennt werden. Die Modifikation von passwortgeschützten Teilen von Dokumenten erfolgt direkt im Internet. Die Ausgabe wird von einem multithreaded Web-Browser–Modul realisiert, das einen gleichzeitigen Zugriff von mehreren Benutzern ermöglicht. Mitintegriert ist die Archivierung sowie die Anbindung an ein Information Retrieval-System.
Dem Datenmodell liegt die frei verfügbare objektorientierte Datenbank ObjectStore PSE for Java zugrunde (die Standardversion). Diese kann direkt von Object Design [Obj97] bezogen werden. In der derzeitigen Implementierung beträgt das Größenverhältnis des Quelldokumentes zu seiner Repäsentation in der Datenbank etwa 1:4. Der Web-Server Jigsaw ist ebenfalls frei verfügbar und wird vom W^3C-Konsortium [WK97] entwickelt. Das System ist in Java implementiert und damit plattformneutral und portabel.
Mit etwas Erfahrung können verschiedene Dokumente relativ schnell für EPOS aufbereitet wer-

den. So nahm die manuelle Transformation des Fach- und Studienführers[2] für die Linguistische Datenverarbeitung/Computerlinguistik knapp über drei Stunden in Anspruch. Als eine weitere Testanwendung wurden einige Dokumente aus dem DLI-Projekt (Digital Library Initiative) [DLI97] verwendet. Eine rudimentäre HTML-Formatierung von diesen Dokumenten[3] war bereits in etwa 30 Minuten möglich. Die vorliegende Publikation, die Sie entweder in Ihren Händen halten oder in Ihrem Browser sehen, wurde ebenfalls aus einer SGML-Datei extrahiert und liegt sowohl in einer Papierversion als auch in elektronischer Form im Web[4] vor. Weitere Informationen zu EPOS und Anwendungsbeispiele sind auf unserem Home-Server unter *http://leipzig.informatik.uni-leipzig.de:8080/* verfügbar.

10 Danksagung

Ich möchte mich bei Herrn Prof. Rahm für seine kontinuierliche Unterstützung bei der Entwicklung des Systems bedanken. Timo Böhme danke ich für seine wertvollen Hinweise in der Implementierungs- und Testphase.

Literatur

[Böh95] Klemens Böhm. Administering Structured Documents in Digital Libraries, 1995.

[DLI97] DLI. Digital Library Initiative. http://dli.grainger.uiuc.edu/default.htm, 1997.

[ISO96] ISO/IEC 10179. *Information technology – Text and office systems – Document Style Semantics and Specification Language (DSSSL).* ISO/IEC, 1996.

[Loo95] Mary E.S. Loomis. *Object Databases: The Essentials.* Addison-Wesley, 1995.

[MB96] Sergej Melnik and Timo Böhme. SQUIRREL: Why And How. http://leipzig.informatik.uni-leipzig.de:8080/, 1996.

[McC97] Michael McCool. The Metamedia Project. http://www.cgl.uwaterloo.ca/~mmccool/Meta/index.html, 1997.

[Obj97] Object Design. ObjectStore PSE for Java. http://www.odi.com/, 1997.

[Sie96] Jon Siegel. *CORBA Fundamentals and Programming.* Wiley, 1996.

[2] http://leipzig.informatik.uni-leipzig.de:8080/links/sf

[3] http://leipzig.informatik.uni-leipzig.de:8080/links/DLI-2

[4] http://leipzig.informatik.uni-leipzig.de:8080/links/epos

[STE97] STEP Stürtz Electronic Publishing GmbH. *SigmaLink: Das intranet-basierte Redaktionssystem mit voller SGML- und XML-Unterstützung.* STEP, 1997.

[Szi95] Horst Szillat. *SGML: eine praktische Einführung.* Thomson, 1995.

[WE89] Jos Warmer and Sylvia van Egmond. The Implementation of the Amsterdam SGML Parser, 1989.

[WH96] Matt Welsh and Greg Hankins. Linuxdoc-SGML User's Guide. http://www.caldera.com/LDP/Linuxdoc-SGML.html, 1996.

[WK97] W^{3}C-Konsortium. http://www.w3.org/pub/WWW/, 1997.