Konzeption und Umsetzung von multimedialen Nutzerschnittstellen im eNoteHistory-Musikarchiv

Diplomarbeit

Universität Rostock, Institut für Informatik

vorgelegt von
Holm Engelbrecht, geb. Fanselow
geboren am 19.06.1977 in Waren/Müritz
Matrikel-Nr.: 097201989

Gutachter: Prof. Dr. Andreas Heuer,
Prof. Dr. Clemens H. Cap

Betreuer: Dipl.-Ing. Temenushka Ignatova,
Dipl.-Inf. Ilvio Bruder

Abgabedatum: 15. September 2004

Diese Diplomarbeit widme ich
meinen Großeltern.

Uroma Frieda

Opa Hans

Oma Erna

Opa Kurt

Oma Ruth

&

Oma Ilse

Zusammenfassung

Im Rahmen dieser Arbeit ist eine leistungsstarke und flexible Schnittstelle zur Erstellung von multimedialen Front-Ends für digitale Bibliotheken entstanden. ***LibScens*** baut konsequent auf eine dreistufige Architektur auf, welche sich in Nutzerschicht, Schnittstellenschicht und Datenbankschicht gliedert. Das Konzept ermöglicht einerseits die Anbindung mehrerer heterogener Datenbanksysteme und lässt andererseits ein weites Spektrum divergierender Ausgabeformate zu.

Im Mittelpunkt steht dabei der modulare Entwurf der Szenariobausteine, welche den Programmablauf definieren. In klar getrennten Abschnitten stellen sie die Verbindung mit dem Nutzer her, sind für die Befüllung und Konvertierung der Eingabedaten aus den verschiedenen Quellen verantwortlich und legen die konkreten Repräsentationsformen von komplexen Datenstrukturen fest. Durch den Einsatz so genannter Hilfsfunktionen kann die Funktionalität der Szenariobausteine projektspezifisch angepasst oder erweitert werden. Geschachtelte Templates hingegen ermöglichen die Adaption von Interaktions- und Darstellungsformen zur Laufzeit.

Unter Einsatz fortschrittlicher Standards wie XML, XPath und XSL-Transformationen in einer Java-2-Laufzeitumgebung wurde im Implementationsteil ein Prototyp entwickelt, der zukunftsträchtige Technologien mit performanten Softwarewerkzeugen vereint.

Abstract

Within the scope of this work, a high-performance and flexible interface for creating multimedia front-ends for Digital Libraries was developed. ***LibScens*** is consequently based on a three-stage architecture, which can be subdivided into user layer, interface layer and database layer. The concept enables the connection of heterogeneous database systems on the one hand and is able to generate a wide range of divergent output formats on the other hand.

The central point of this work is represented through the modular design of the scenario modules, which are responsible for controlling the program sequence. In clearly separated sections the modules will establish the connection with the user, they will fill and convert the input tables with content from various sources and define the concrete representation form of complex structures. By adjusting the so-called help functions, the functionality of the scenario modules can easily be adapted or expanded to fit the specific demands of the project. Nested templates on the other side can be used for modifying interaction and presentation components.

By using progressive standards such as XML, XPath and XSL-Transformations inside a Java 2 runtime environment, a prototype was evolved, which combines seminal technologies with high-capacity software tools.

CR-Klassifikation

H.5.1: Multimedia Information Systems
H.5.2: User Interfaces
H.5.4: Hypertext/Hypermedia

Keywords

Multimedia Interface, XSL Transformation, Digital Library, User Scenario, Usability

Inhaltsverzeichnis

Zusammenfassung **1**

Inhaltsverzeichnis **2**

Aufgabenstellung **4**

1 Einleitung **7**
1.1 Vorbemerkung 7
1.2 Motivation 8

2 Grundlagen **11**
2.1 User Centered Design 11
2.2 Unterteilung der Nutzer aus Usability-Sichtweise 12
2.2.1 Standardnutzer 15
2.2.2 Expertennutzer 16
2.2.3 Administratoren 17
2.3 Vergleich digitaler Biliotheken 18
2.3.1 MyCoRe 19
2.3.2 Greenstone 23
2.3.3 Weitere digitale Bibliotheken 28
2.4 Ableitung der Ziele zur Umsetzung der Schnittstelle 30

3 Konzeption **32**
3.1 Einordnung von ***LibScens*** in eNoteHistory 32
3.2 Überblick der unterliegenden Architektur 34
3.2.1 Schnittstellenarchitektur 34
3.3 XML-Szenariobausteine 36
3.3.1 Modularer Aufbau der XML-Szenariobausteine 36
3.3.2 Betrachtung der Zustandsspeicherung von Szenarios 37
3.3.3 Konzept der Basisdatentypen, Hilfsfunktionen & -prozeduren 39
3.3.4 Details der Deskriptoren 40
3.3.5 Details der Eingabetabellen 41
3.3.6 Details der komplexen Strukturen 44
3.3.7 Details zum Seitenlayout 51
3.4 Integration der Nutzerauthentifizierung 56

4 Implementierung **59**
4.1 Techniken zur Umsetzung von ***LibScens*** 59
4.1.1 Front-End 59
4.1.2 Interface 60
4.1.3 Back-End 61
4.2 Dreistufiger Transformationsprozess 61
4.3 Aufbau der Java-Klassenstruktur 63
4.4 Ablaufplan eines Szenarioaufrufs 64
4.5 Ausgewählte Implementierungsdetails 66
4.5.1 Details des Deskriptorenmoduls 66
4.5.2 Interface und dynamisches Laden der Hilfsfunktionen 67
4.5.3 Rekursive Abarbeitung von DOM-Knoten 68
4.5.4 Integration eines Logbuchs 69
4.5.5 Implementierungsdetails der Eingabetabellen 71
4.6 Layoutumsetzung mittels XSL-Templates 72

5 Evaluierung des Prototyps **75**
5.1 Usability-Betrachtungen des Prototyps 75
5.1.1 Testmethoden 76
5.1.2 Inspektionsmethoden 79
5.2 Performance-Messungen des Prototyps 81
5.2.1 Festlegung der Anforderungen an die Leistungsfähigkeit 82
5.2.2 Aufteilung und Messung der Abschnitte 83
5.2.3 Techniken für komplexe Strukturen und XSL-Transformationen 86

6 Zusammenfassung und Ausblick **88**

A Screenshots digitaler Bibliotheken **90**

B Beispiele **93**
B.1 Beispiel 1 — Entfernung von Knoten 93
B.2 Beispiel 2 — Hinzufügen von Knoten 94

C Glossar **97**

E Literaturverzeichnis **99**

F Abbildungsverzeichnis **104**

Aufgabenstellung

Konzeption und Umsetzung von multimedialen Nutzerschnittstellen im eNoteHistory-Musikarchiv

Bearbeiter

- Holm Engelbrecht
  engel@informatik.uni-rostock.de

Gutachter

- Prof. Dr. Andreas Heuer,
  heuer@informatik.uni-rostock.de
- Prof. Dr. Clemens H. Cap
  clemens.cap@informatik.uni-rostock.de

Betreuer

- Dipl.-Ing. Temenushka Ignatova
  ti005@informatik.uni-rostock.de
- Dipl.-Inf. Ilvio Bruder
  ilr@informatik.uni-rostock.de

Starttermin

- 01.01.2004

Beschreibung

Die meisten Ansätze zur Definition einer Nutzerschnittstelle auf Datenbanken gehen von den Datenstrukturen der Datenbank aus. Dies spiegelt oft nicht das tatsächliche Nutzungsszenario wieder und wird selten auf Usability getestet.

Die Diplomarbeit soll ausgehend von den Nutzungsszenarien einer digitalen Bibliothek eine Nutzerschnittstelle definieren. Für die Datenbankanbindung soll dann eine Umsetzungsschicht mit einem geeigneten Modell entwickelt werden. Diese Schicht soll möglichst verschiedene Nutzungsszenarien, wie Suche, Navigation, Anfragen, unterstützen und gleichzeitig auch robust gegenüber Spezialitäten verschiedener Datenbanksysteme sein. Dabei muss ein Weg zwischen Performance und Funktionsvielfalt gefunden werden. Datenbankfunktionalität, wie Anfragen, eingeschränkte Volltextsuche, Funktionsaufrufe (UDFs) auf unterschiedlichen Datenstrukturen, soll unterstützt werden. Weitere spezielle Nutzungsfunktionen sollen über eine geeignete Integrationsschnittstelle (Plug-Ins) eingebaut werden können.

Das erstellte Konzept ist innerhalb des eNoteHistory-Projektes prototypisch umzusetzen. Für die Modellierung und Datenpräsentation sind XML- und XSLT-Techniken zu verwenden. Als Werkzeuge sollen IBM DB2 als objektrelationales Datenbanksystem, sowie Java und Java Servlets in einer Tomcat-Webserver-Umgebung genutzt werden.

Arbeitsschritte

Folgende Schritte werden innerhalb der Arbeit bearbeitet:

- Einarbeitung in das Thema, Literaturrecherche
- Untersuchung der Nutzungsszenarien, Aufstellung von Forderungen an die Nutzerschnittstelle
- Vergleich verschiedener Ansätze für eine Nutzerschnittstelle, Bewertung bzgl. der Anforderungen
- Entwicklung eines geeigneten Modells für die Anbindung einer Datenbank an die Nutzerschnittstelle
- Prototypische Implementierung des entwickelten Ansatzes
- Tests innerhalb des eNoteHistory-Projektes

Literatur

1 Datenbank-Anwendungsprogrammierung

2 Rahm und Vossen: *Web & Datenbanken*, d.punkt, 2003

3 Börner und Chen: *Visual Interfaces to Digital Libraries*, Springer, 2002

Erklärung

Ich erkläre mich damit einverstanden, dass die Ergebnisse meiner Diplomarbeit im Rahmen von Forschung und Lehre am Fachbereich Informatik der Universität Rostock genutzt werden können.

Kapitel 1

Einleitung

1.1 Vorbemerkung

Digitale Bibliotheken sind aus unserem Alltag nicht mehr wegzudenken. Selbst bei der Suche nach einer Definition zu dem Wort „Bibliothek“ greift man heutzutage auf einen digitalen Ableger dieses Genres zurück:

„[Eine Bibliothek ist] eine planmäßig angelegte Büchersammlung (auch Sammlung von Handschriften, neuerdings auch von audiovisuellem Material), ferner das Gebäude, in dem sie aufgestellt ist.“ [BRO00]

Bei der Definition einer digitalen Bibliothek wird das in der obigen Definition genannte Gebäude in den meisten Fällen durch eine technische Komponente ersetzt. Eine Interpretation des Begriffes ist etwa die folgende:

„Digitale Bibliotheken sind Organisationen, die Mittel zur Verfügung stellen, inklusive des Personals, um die digitalen Werke auszuwählen, zu strukturieren, den geistigen Zugriff zu gewähren, auszuwerten, zu verteilen, die Integrität zu gewährleisten und die Fortdauer zu sichern, so dass der Zugriff für eine festgelegte Gemeinschaft oder über ein Kommunikationssystem leicht und wirtschaftlich ist.“ [WAT98]

Ihnen wird schon jetzt eine immens wichtige Rolle im Bereich der „Datenbanken und Informationssysteme“ zugewiesen, wie die zahlreichen Neu- und Weiterentwicklungen von Software und dazugehörigen Werkzeugen für digitale Bibliotheken eindrucksvoll unter Beweis stellen. Ihre Bedeutung wird aber in Zukunft einen noch weitaus höheren Stellenwert annehmen, bedenkt man, dass es in vielen Teilen der Gesellschaft Versuche gibt, das gesammelte, historische Wissen in riesigen Datencontainern zu archivieren. Und diese Entwicklung befindet sich erst im Anfangsstadium.

Das bekannteste Einsatzgebiet ist dabei nach wie vor das World Wide Web. Bei jedem Stöbern in einem Internet-Versandhaus wie „Amazon“ [AMA04] greifen wir auf eine derartige Datensammlung zurück, eine Suche in Lexikas wie dem „Brockhaus - multimedial“ [BRO00] basiert ebenfalls auf einem derartigen System. Ebenso verhält es sich bei der Recherche nach wissenschaftlichen, technischen oder medizinischen Büchern und Zeitschriften im „SpringerLink“-Online-Archiv [SPR04].

1.2 Motivation

Abseits der im vorausgehenden Abschnitt genannten ausgefeilten kommerziellen, dadurch auch meist sehr spezialisierten Systemen gibt es eine Vielzahl von Forschungsimplementierungen und Prototypen, welche sich mit dem Gebiet frei verfügbarer digitaler Bibliotheken beschäftigen. Zu diesen Systemen zählen unter anderem MyCoRe [MCR04], Greenstone [GRE04], i-Tor [ITO04] oder EPrints [EPR04]. Auf diese Systeme wird im Kapitel 2.3 auf Seite 18 genauer eingegangen.

Hier erkennen selbst fachfremde Nutzer oftmals, dass die angebotene Funktionalität und der Umgang mit derartigen Sammlungen von „dokumentenähnlichen Objekten“ noch nicht perfekt ist. Dabei sind die Schwachstellen bei den unterschiedlichen Systemen sehr verschieden und unterschiedlich stark ausgeprägt. Während es bei der einen digitalen Bibliothek an der Usability für Administratoren fehlt, kann man bei anderen Systemen nur eine spezielle Datenbank unterlegen und es mangelt deshalb an Flexibilität.

Eine Vision, die sich bei der Erstellung einer Schnittstelle für digitale Bibliotheken ergibt, sieht etwa wie folgt aus: Die Schnittstelle sollte sehr flexibel gegenüber den sich ständig ändernden, technischen Umgebungen gestaltet werden. Dies beinhaltet eine Anpassung an unterschiedliche Eingabe- bzw. Ausgabemedien, verschiedene Datenbanksysteme und spezielle Anfragesprachen. Es muss robust gegenüber allen möglichen Arten von Spezialfällen und Fehlern sein. Um etwaigen Evolutionen im Bereich von digitalen Bibliotheken nicht im Weg zu stehen, ist eine gut gekapselte Struktur der Architektur unabdinglich, da nur so einzelne Teilbereiche angepasst und notfalls ausgewechselt werden können. In jedem Bereich der Entwicklung dieser Schnittstelle muss der Nutzbarkeit oder Usability des Systems ein wichtiger Stellenwert zugeordnet werden.

Im folgenden Abschnitt werden noch einmal die wichtigsten allgemeingültigen Anforderungen an digitale Bibliotheken genauer erläutert:

1. **Robustheit**
   Nur robuste Systeme können sich auf Dauer durchsetzen. Gerade bei Webanwendungen besteht bei fehleranfälligen Programmen die große Gefahr, dass Teile oder die komplette Anwendung aufgrund von Fehlern ausfallen. Während ein Anwender bei der Nutzung lokaler Programme beim Auftreten von Fehlern immer die Möglichkeit hat, das Programm erneut zu starten, steht diese Alternative bei Webanwendungen nicht zur Verfügung.

   In einem derartigen Fall müsste nun ein Administrator kontaktiert werden, welcher die notwendigen Schritte einleitet. Geschieht dies nicht umgehend, ist es nicht selten, dass eine Anwendung über einen längeren Zeitraum nicht erreichbar ist oder nur Teile abgefragt werden können. Kein Anwender und Administrator würde diesen Grad an Zuverlässigkeit hinnehmen.

2. **Usability**
   Jedes System ist nur so gut, wie es sich bedienen lässt. Das schließt folgende drei Nutzergruppen mit ein: Standardnutzer, Expertennutzer und Systemadministratoren.

   Oftmals wird wenig Know-how in die Usability investiert oder nur einige der genannten Nutzergruppen adäquat unterstützt. Während sich ein Experte mit fast allen genannten Systemen mehr oder weniger gut zurecht finden wird, hat der Standardnutzer oftmals Probleme mit dem Einstieg.

Auf der anderen Seite wird den Systemadministratoren nicht unbedingt die Arbeit erleichtert, indem er Datenbündel über die Kommandozeile einpflegen oder die Nutzerdaten manuell in einer Datei bearbeiten muss, wie es bei MyCoRe im Moment der Fall ist [LUE04].

Zur Usability zählt außerdem, was für Hilfen den Nutzern und Administratoren von digitalen Bibliotheken zur Verfügung gestellt werden. Natürlich ist eine intuitive Bedienung immer anzustreben, wobei bei kompletter Umsetzung eine Hilfsdatei obsolet werden würde. Dies kann jedoch gerade bei komplexen Systemen, wie digitalen Bibliotheken, nicht vollständig gewährleistet werden.

3. **Strukturierte Architektur**
   Wie auch in anderen Bereichen der Informatik spielen gerade bei digitalen Bibliotheken Effizienzbetrachtungen eine besonders wichtige Rolle. Dieser Umstand ergibt sich schon allein anhand der großen Datenmengen, welche bewältigt werden müssen. Viele Autoren drängen deshalb zu folgenden Hinweisen: „Die Anforderung lautet, die Informationen schneller und effizienter zu verarbeiten.“ [GUL02]

   Dies ist nicht zwingend der richtige Ansatz, vor allem im Hinblick auf mögliche Erweiterungen und Eingliederung in andere Projekte. Die Performance-Evaluierung darf zwar nicht vernachlässigt werden, aber sie darf keinesfalls auf Kosten der strukturierten Architektur gehen. So müssen die sauberen Grenzen zwischen den unterschiedlichen konzeptuellen Bausteinen der Schnittstellen jederzeit bestehen bleiben. Nur dann können bestimmte Konzepte erweitert, angepasst oder ausgewechselt werden. Auch zur Erläuterung und für das Verständnis von zu Grunde liegenden Konzepten ist eine gut verständliche Struktur von Vorteil.

4. **Flexibilität**
   Erst durch eine gut strukturierte Architektur wird Flexibilität möglich. Flexibilität wird bei dieser Schnittstelle hauptsächlich in folgenden Bereichen benötigt:

   - *Ein- und Ausgabeformate:*
     Die Ein- und Ausgabe von digitalen Bibliotheken sollte möglichst vielfältig sein. Als Ausgabeformate kommen beispielsweise eine reine HTML-Betrachtung, ein PDF-Dokument oder eine XML-Datei in Frage. Durch die Verwendung einer auf Informationen beschränkten Datenspeicherung, wie beispielsweise XML, kann zu einem späteren Zeitpunkt das Layout festgelegt werden.
   - *Datenbank:*
     Viele Systeme von digitalen Bibliotheken beschränken sich auf bestimmte Datenbanken oder Typen von Datenbanken. Diese Spezialisierung bringt neben Vorteilen aber hauptsächlich folgende Probleme mit sich. Werden z.B. kommerzielle Datenbanken als Back-End eingesetzt, dann wird nicht jeder potentielle Endanwender Lizenzen hierfür besitzen oder erwerben wollen. Außerdem sind Nutzer von bestimmten Datenbanken oft nicht gewillt, ihre bewährten Systeme gegen neue auszutauschen, da dies einen erheblichen Mehraufwand an Einarbeitungszeit bedeuten würde.
   - *Anfragesprache:*
     Beschränkt man sich nun nicht auf eine bestimmte Datenbank, dann wird man zwingend mit unterschiedlichen Anfragesprachen konfrontiert. Deshalb sollte man auch in diesem Bereich auf eine weitgehende Abkapselung oder Modularisierung achten.

5. **Einsatz bekannter Techniken**
   Bei der Konzeption der Module für digitale Bibliotheken sollte, falls möglich, auf bekannte Techniken der Informatik zurückgegriffen werden. So macht es wenig Sinn, komplett neue Strukturen zur Speicherung von Dateien zu entwerfen, wenn stattdessen XML verwenden werden kann. Statt neue Konvertierungsmechanismen zu kreieren, kann auf XSLT-Transformationsalgorithmen zurückgegriffen werden.

   Als Gründe hierfür ist Folgendes anzuführen: Bekannte Techniken erfordern im Allgemeinen weniger Aufwand in der Konzeptionsphase. Sie reagieren robust auf Spezial- und Fehlerfälle. Auch für den Administrator der digitalen Bibliothek oder der Schnittstelle ist es in diesem Fall erheblich einfacher, sich in die Technologie einzuarbeiten, sie zu verstehen und gegebenfalls anzupassen. Es ist darauf zu achten, dass die verwendeten Produkte von Drittanbietern möglichst frei verfügbar sind, gut dokumentiert und ausführlichen Tests unterzogen wurden. Als Richtlinie kann man sich dabei an Standards orientieren, die bereits von speziellen Gremien, wie z.B. dem World Wide Web Consortium (W3C) [W3C04], verabschiedet wurden.

Diese Diplomarbeit wird hier ansetzen und versuchen, Lösungen für die angeführten Anforderungen zu finden oder Alternativen aufzuzeigen. Dabei wird sich zeigen, in welchem Umfang diese überhaupt erfüllt werden können. Vorab werden einige verfügbare digitale Bibliotheken auf die oben genannten Forderungen hin untersucht. Diese Systeme werden im Kapitel 2.3 ab Seite 18 eingehender behandelt.

Kapitel 2

Grundlagen

2.1 User Centered Design

User Centered Design (UCD) ist ein Ansatz, bei dem der Designprozess auf den Informationen derjenigen Anwender basiert, welche das Produkt nutzen werden. Im UCD-Ansatz steht dabei in jeder Phase des Produktzyklus der Nutzer im Vordergrund. Dies gilt sowohl für die Planungs-, Design- als auch für die Entwicklungsphase. [UPA03]

Während der Ansatz der fokussierten Entwicklung im Hinblick auf die Nutzerbedürfnisse in anderen Bereichen der Industrie seit jeher im Vordergrund steht, ist dieses Bestreben in der Softwareindustrie erst seit wenigen Jahren aufgekommen. Dabei war und ist es für Softwareprodukte ungleich schwerer, konkrete Methoden zur Umsetzung des Designprozesses zu finden. So gibt es einen internationalen Standard[1], der als Basis für viele UCD-Methodologien dient. Dieser Standard definiert zwar ein generelles Verfahren, der benutzerorientierte Aktionen während des gesamten Entwicklungszyklus mit einbezieht, aber spezifiziert keine exakten Methoden.

Wurde einmal die Notwendigkeit der nutzerorientierten Gestaltung erkannt, besteht das Modell aus vier Aktionen, die den Hauptzyklus der Arbeit darstellen (siehe Abbildung 1). Diese Aktionen sind im Einzelnen:

- **Spezifizieren des Anwendungskontextes** - Identifiziere die Anwender, die das Produkt nutzen werden, welche Hilfsmittel sie dafür verwenden und unter welchen Bedingungen sie es einsetzen.
- **Spezifizieren der Anforderungen** - Identifiziere alle unternehmerischen Anforderungen oder Nutzerziele, die vom Produkt erfüllt werden müssen, um erfolgreich zu sein.
- **Produzieren von Designlösungen** - Dieser Teil des Prozesses kann in einzelnen Teilabschnitten durchgeführt werden, ausgehend von einem rohen Konzept bis hin zu einem kompletten Design.
- **Auswertung von Designvorschlägen** - Den wichtigsten Teil des gesamten Prozesses stellt die Evaluierung dar. Idealerweise sollte sie durch Usability-Tests mit Unterstützung der tatsächlichen Nutzer vorgenommen werden. Eine interessante Studienarbeit zu diesem The-

[1] ISO 13407: Benutzerorientierte Gestaltung interaktiver Systeme

ma wurde von Manja Nelius [NEL03] vorgelegt. Die Auswertungen der Designs sind ebenso wesentlich wie die Qualitätstests bei guter Softwareentwicklung.

Eine detailliertere Aufschlüsselung der einzelnen Arbeitsschritte einer konkreten Methode zur nutzerorientierten Gestaltung kann unter [UPA03] nachgelesen werden.

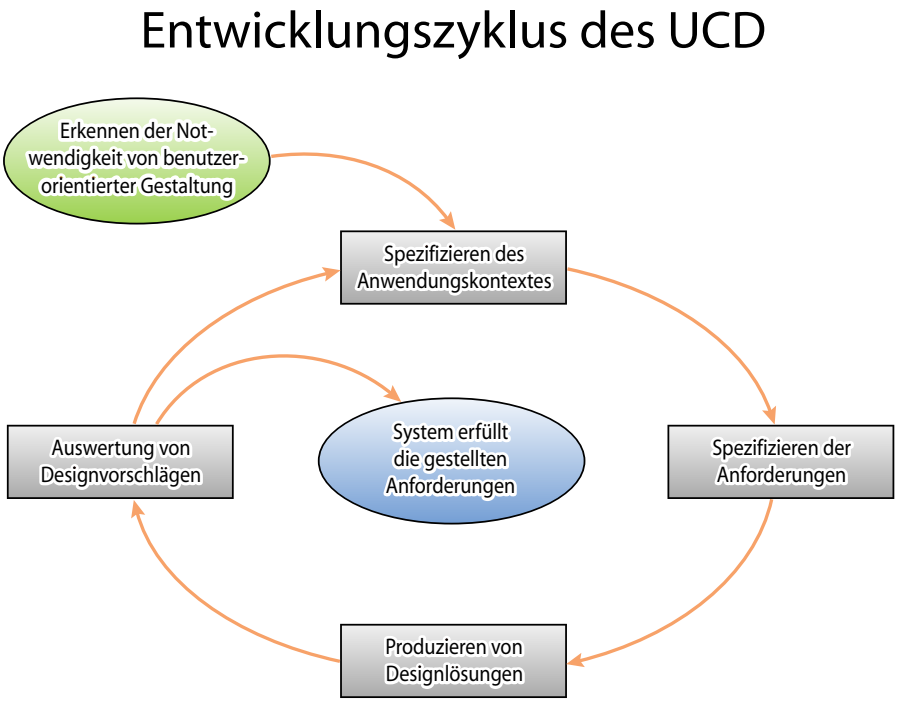

Abbildung 1: *Entwicklungszyklus der benutzerorientierten Gestaltung*

2.2 Unterteilung der Nutzer aus Usability-Sichtweise

Den ersten Schritt des nutzerorientierten Gestaltens (siehe Abschnitt 2.1) stellt demnach die „Spezifikation des Anwendungskontextes“, also die Identifikation der Nutzer und deren Umfeld, dar. Auch bei Jakob Nielsen, dem wohl bekanntesten Vorreiter der Usability, steht in seiner Abhandlung *„Usability Engineering“* [NIE93] an erster Stelle:

> *„1. Know the User.“*

Um die Gruppierungen zu bilden, müssen erst einmal Merkmale definiert werden, nach denen unterschieden werden kann. Eine Liste der 18 wichtigsten generellen Designprinzipien wird in [GAL97] aufgeführt. Die Punkte wurden von unterschiedlichen Prinzipien führender Organisationen und Persönlichkeiten im Bereich der Usability-Forschung[2] abgeleitet. Für tiefergehende Studien sei auf das Standardwerk „*Designing the User Interface*“ von Ben Shneiderman [SHN92] und auf „*The Essential Guide to User Interface Design*“ von Wilbert O. Galitz [GAL97] verwiesen. Ein komprimierter Vortrag wurde vom Autor selbst [ENG03] verfasst. In der anschließenden Liste werden die generellen Designprinzipien in alphabetischer Reihenfolge aufgelistet:

[2] Zu den genannten Organisationen und Personen zählen: von Garlitz (1992), IBM (1991), Mayhew (1992), Microsoft (1992, 1995), Open Software Foundation (1991), Verplank (1988).

Prinzip	kurze Erläuterung zum Prinzip
1.) Ästhetik	Sorge für ein ästhetisches Gleichgewicht. Der Kontrast zwischen Bildschirmelementen muss dabei aussagekräftig sein und es sollten Gruppierungen gebildet werden, wann immer diese sinnvoll sind.
2.) Effizienz	Minimiere die Augen-, Hand- und andere anfallende Steuerbewegungen. Übergänge zwischen verschiedenen Systemsteuerelementen sollten leicht und effizient erfolgen.
3.) Erfahrung	Entwickle nach bekannten Konzepten und nutze dabei eine Sprache, welche dem Nutzer bekannt ist. Unterstütze dies so weit wie möglich durch Metaphern der realen Welt.
4.) Flexibilität	Ein System muss auf verschiedene Nutzerwünsche flexibel reagieren. Sorge dafür, dass unterschiedlich ausgebildete Nutzergruppen entsprechend ihren Kenntnissen mit dem Produkt interagieren können.
5.) Geradlinigkeit	Aufgaben sollten direkt und intuitiv ausgeführt werden. Geradlinigkeit kann durch die Objekt-Handlungs-Sequenz „direkter Manipulationssysteme“[3] hervorgerufen werden.
6.) Klarheit	Konzipiere eine visuell, konzeptionell und sprachlich klar verständliche Schnittstelle. Die Texte sollten dabei frei von „Computer-Jargon“ sein.
7.) Kompatibilität	Das Produkt sollte kompatibel zu den Nutzern, Aufgaben und anderen Produkten sein. Die Nutzerperspektive muss also immer in den Arbeitsprozess integriert werden.
8.) Konfigurierbarkeit	Erlaube Konfigurationen und Rekonfigurationen von Einstellungen. Dabei müssen jedoch immer sinnvolle Grundeinstellungen vorhanden sein. Sie verbessern unmittelbar das Gefühl der Kontrolle.
9.) Konsistenz	Ein System sollte durchweg gleich aussehen, handeln und funktionieren. Designkonsistenz spielt eine Hauptrolle bei allen Designaktivitäten.

[3] Als *direkte Manipulation* bezeichnet man nach [SHN83] eine Interaktionsform, bei der das zu bearbeitende Objekt ständig sichtbar ist und mit schnellen, rücknehmbaren und inkrementellen Aktionen ohne Nutzung einer komplexen Kommandosprache direkt bearbeitet werden kann.

Prinzip	kurze Erläuterung zum Prinzip
10.) Kontrolle	Der Nutzer muss jederzeit die Kontrolle über die Interaktion besitzen. Kontrolle gibt dem Anwender das Gefühl, dass das System auf *seine* Aktionen reagiert.
11.) Modularisierung	Designs müssen modular und austauschbar gestaltet werden. Dies ist zwingend notwendig, um auftretende widersprüchliche Designprinzipien auszubalancieren.
12.) Reaktion	Das System muss umgehend auf eingehende Nutzeranfragen reagieren. Sorge für eine umgehende visuelle oder auditive Bestätigung aller Nutzeraktionen.
13.) Schlichtheit	Sorge für eine Schnittstelle, die so schlicht wie möglich ist. Um dies zu erreichen, können beispielsweise moderne Methoden, wie die *Progressive Disclosure* - *„schrittweise Enthüllung“*[4], genutzt werden.
14.) Toleranz	Toleriere allgemeine und unvermeidbare menschliche Fehler. Schütze den Anwender gegen mögliche fatale Fehler. Sollten Fehler auftreten, dann sorge für eine konstruktive Mitteilung.
15.) Transparenz	Der Nutzer sollte den Fokus nur auf seine Aufgabe richten dürfen, ohne Kenntnisse der unterliegenden, internen Mechanismen zu besitzen.
16.) Verständlichkeit	Ein System sollte einfach zu verstehen und zu erlernen sein. Der Fluss der Aktionen, Reaktionen, visuellen Repräsentationen und Informationen sollte in einer vernünftigen Reihenfolge angeordnet sein.
17.) Vorhersagbarkeit	Der Nutzer sollte den natürlichen Verlauf jeder Aufgabe erwarten. Die Reaktionen eines Systems sollten durch das Wissen oder die Erfahrungen, die ein Anwender zu einem früheren Zeitpunkt einmal gemacht hat, vorhersagbar sein.
18.) Wiederherstellbarkeit	Handlungen sollten abgebrochen und rückgängig gemacht werden können. Falls Probleme während des Arbeitsverlaufs entstehen, sollte man umgehend zu einem *bestimmten* Punkt zurückkehren können.

Abbildung 2: *Übersicht der 18 generellen Designprinzipien*

[4] Mit *Progressive Disclosure* ist gemeint, dass komplizierte und detaillierte Funktionen so lange versteckt werden sollten, bis sie benötigt werden. Dieses System wird beispielsweise auch in den neueren Microsoft Windows-Versionen eingesetzt.

Diese Prinzipien sind geeignet, um eine Unterteilung der Nutzer von digitalen Bibliotheken vorzunehmen. Im *theoretischen* Idealfall sollten alle Punkte der zusammengefassten Prinzipien strikt eingehalten und jedem Prinzip die gleiche Priorität zugeteilt werden. In der *Praxis* lässt sich dies jedoch nicht umsetzen und ist zudem nicht sinnvoll, da sich je nach Erfahrungsgrad und Einsatzgebiet des Anwenders verschiedene Gewichtungen ergeben.

Selbst bei Betrachtung eines einzelnen Nutzers über einen längeren Zeitraum könnten und werden sich dessen Prioritäten entscheidend verlagern. So war jeder Experte auch einmal ein Laie und musste somit langsam sein Umfeld kennenlernen und verstehen. Erst im Laufe der Zeit wandelten sich seine Ansprüche. Ein interessanter Vergleich wird dazu von Prof. Dr. Michael Weber [WEB02] vorgenommen. Es wird ein so genannter „*Intimacy Gradient*“ vorgestellt, welcher wie folgt erklärt wird:

> *„Einerseits gibt es Räume mit verschiedener Privatsphäre (Flur vs. Schlafzimmer), andererseits will man Besucher je nach Bekanntheitsgrad unterschiedlich weit in die Privatsphäre eindringen lassen. [Die] Lösung [ist,] vom Eingangsbereich bis zu den hintersten Räumen einen Gradienten zunehmender Privatsphäre [zu] schaffen.“*

Abgesehen vom Gradienten überlappen sich die Designprinzipien von Abbildung 2 in vielen Bereichen und stehen dabei häufig im Konflikt zueinander, das heißt, es ist praktisch unmöglich *ein* ideales Produkt für *alle* Anwender zu erstellen. Auch deshalb muss durch die Priorisierung bestimmter Punkte eine Anpassung an unterschiedliche Nutzergruppen erfolgen.

Die Frage, die sich unmittelbar stellt, ist: *„Wie feingranular muss dieser Gradient gewählt werden?“* Oder anders ausgedrückt: *„Wieviele Nutzergruppen sind sinnvoll?“*

Im Laufe der Implementierungsarbeiten an dem Prototypen für das eNoteHistory-Musikarchiv und den damit verbundenen Treffen mit Musikwissenschaftlern und Bibliothekaren haben sich dabei drei unterschiedliche Gruppen herauskristallisiert: **Standardnutzer**, **Expertennutzer** und **Administratoren**. Dabei sind die Anforderungen an den Umgang mit digitalen Bibliotheken der einzelnen Gruppierungen komplett unterschiedlich. Nachfolgend werden die Mitglieder der Gruppen kurz vorgestellt und anhand einer einfachen Priorisierung der Designprinzipien definiert[5].

2.2.1 Standardnutzer

Unter der Gruppe „Standardnutzer“ werden alle Laien und Gelegenheitsnutzer eines Systems zusammengefasst. Bei dieser Gruppe steht ganz klar die schnelle und intuitive Aufnahme des Kontextes im Vordergrund. Bei eNoteHistory versteht man unter den Standardnutzern die im Internet recherchierenden Bibliothekskunden.

Die grafische Repräsentation von bestimmten Inhalten und Interaktionsmöglichkeiten muss zu jedem Zeitpunkt übersichtlich gestaltet sein. Dieser Umstand liegt unter anderem in den begrenzten Kapazitäten des menschlichen Gehirns begründet. Zu diesem Phänomen gibt es zahlreiche Untersuchungen, die schon vor Jahrzehnten durchgeführt wurden, aber dennoch weiterhin gültig sind. Zu einem der interessantesten Artikel gehört eine Abhandlung von George A. Miller [MIL56], indem das temporäre Aufnahmevermögen des Gehirns mit der „magischen Nummer Sieben“ beziffert wird. Dies bedeutet zum Beispiel, dass in einer grafischen Nutzerschnittstelle nicht mehr als

[5] Selbstverständlich ist eine derartige Priorisierung immer subjektiv und an bestimmte Umgebungsvariablen gebunden. Dennoch lässt sich ein Trend erkennen, der diese Unterteilung rechtfertigt.

sieben zu einem bestimmten Kontext gehörenden Menüpunkte angegeben werden sollten. Dieser Fakt wird in vielen GUIs[6] nicht beachtet. Bei einer sehr großen Anzahl von Auswahlmöglichkeiten kann man dieser Anforderung gerecht werden, indem die Menüpunkte logisch gruppiert bzw. geschachtelt werden und dann immer nur ein Ausschnitt der Möglichkeiten dargestellt wird. Dies entspricht dem Konzept des „*Wizards*“, welches etwa in vielen Softwareinstallationsroutinen umgesetzt wird. Bei ***LibScens*** bedeutet dies, dass anstatt eines einzigen oftmals mehrere und weniger komplex gestaltete Szenarios sinnvoller sind.

Standardnutzer legen außerdem viel Wert auf gut dokumentierte und verständliche Hilfen zum Aufbau komplexer Strukturen oder zu den möglichen Interaktionsalternativen. Dies widerspricht natürlich in gewisser Weise dem Grundsatz, dass gute Software selbstdokumentierend sein sollte. Das ist jedoch auch beim besten Bestreben oftmals nicht realisierbar. Eine Priorisierungstabelle für Standardnutzer wird in Abbildung 3 dargestellt.

Prinzip	eher unwicht.	wichtig	sehr wichtig
1.) Ästhetik			sehr wichtig
2.) Effizienz	eher unwichtig		
3.) Erfahrung	eher unwichtig		
4.) Flexibilität	eher unwichtig		
5.) Geradlinigkeit		wichtig	
6.) Klarheit			sehr wichtig
7.) Kompatiblität		wichtig	
8.) Konfigurierbarkeit	eher unwichtig		
9.) Konsistenz			sehr wichtig
10.) Kontrolle			sehr wichtig
11.) Modularisierung	eher unwichtig		
12.) Reaktion			sehr wichtig
13.) Schlichtheit		wichtig	
14.) Toleranz			sehr wichtig
15.) Transparenz			sehr wichtig
16.) Verständlichkeit			sehr wichtig
17.) Vorhersagbarkeit			sehr wichtig
18.) Wiederherstellbarkeit			sehr wichtig

Abbildung 3: *Prioritätentabelle für Standardnutzer*

2.2.2 Expertennutzer

Zu den Experten zählen die Bibliothekare oder geschultes Personal. Sie haben andere Ansprüche an die Schnittstelle, weil sie das Interface ungleich häufiger nutzen. Bei ihnen steht die *Funktio-*

[6] GUI: Graphical User Interface

nalität und *Geschwindigkeit* im Vordergrund. Dieser Umstand kann z.B. auf Kosten einer klaren Strukturierung der Schnittstelle gehen und würde somit gegen einige der 18 genannten Prinzipien verstoßen. Desweiteren benötigen Expertennutzer auch eine komplett unterschiedlich gestaltete Hilfedatei oder sogar gar keine Hilfe, weil sie in der Arbeit geschult sind und mit auftretenden Spezialfällen adäquat umgehen können. In Bezug auf eNoteHistory sind die Expertennutzer entweder Bibliothekare oder Hilfswissenschaftler.

Eine weitere Unterscheidung zu den Standardnutzern ist der Grad der gewährten Funktionalität. So können Experten beispielsweise auf spezielle Suchmechanismen von digitalen Bibliotheken zugreifen, welche dem Standardnutzer gänzlich verwehrt bleiben. Ferner könnte Experten erlaubt werden, die Datenbank mit Datenbündeln zu befüllen oder generell die Datenbank zu manipulieren. Expertennutzer sollten auch aus diesem Grund schon einen gewissen Grad an Verständnis für die Interna der Schnittstelle besitzen. In Abbildung 4 wird die Gewichtung für Experten dargestellt.

Prinzip	eher unwicht.	wichtig	sehr wichtig
1.) Ästhetik		wichtig	
2.) Effizienz			sehr wichtig
3.) Erfahrung		wichtig	
4.) Flexibilität			sehr wichtig
5.) Geradlinigkeit		wichtig	
6.) Klarheit		wichtig	
7.) Kompatiblität		wichtig	
8.) Konfigurierbarkeit			sehr wichtig
9.) Konsistenz		wichtig	
10.) Kontrolle		wichtig	
11.) Modularisierung	eher unwichtig		
12.) Reaktion		wichtig	
13.) Schlichtheit		wichtig	
14.) Toleranz			sehr wichtig
15.) Transparenz		wichtig	
16.) Verständlichkeit		wichtig	
17.) Vorhersagbarkeit			sehr wichtig
18.) Wiederherstellbarkeit		wichtig	

Abbildung 4: *Prioritätentabelle für Expertennutzer*

2.2.3 Administratoren

Eine weitere entscheidenende Nutzergruppe stellen die Administratoren dar. Sie sind für die Erstellung von kontextbezogenen Schnittstellen zuständig. Beispielsweise müssen sie das Layout anpassen und die Datenbankanbindung warten. Für Administratoren ist eine dynamische Schnittstelle mit

auswechselbaren Modulen und flexibler Gestaltung von Front- und Back-End unabdingbar. Administratoren benötigen für eine effektive Arbeit eine gute Kenntnis der Interna einer digitalen Bibliothek. Die folgende Abbildung 5 stellt die Prioritäten für Administratoren dar:

Prinzip	eher unwicht.	wichtig	sehr wichtig
1.) Ästhetik	eher unwichtig		
2.) Effizienz		wichtig	
3.) Erfahrung			sehr wichtig
4.) Flexibilität			sehr wichtig
5.) Geradlinigkeit	eher unwichtig		
6.) Klarheit	eher unwichtig		
7.) Kompatiblität			sehr wichtig
8.) Konfigurierbarkeit			sehr wichtig
9.) Konsistenz	eher unwichtig		
10.) Kontrolle		wichtig	
11.) Modularisierung			sehr wichtig
12.) Reaktion	eher unwichtig		
13.) Schlichtheit	eher unwichtig		
14.) Toleranz		wichtig	
15.) Transparenz	eher unwichtig		
16.) Verständlichkeit		wichtig	
17.) Vorhersagbarkeit		wichtig	
18.) Wiederherstellbarkeit		wichtig	

Abbildung 5: *Prioritätentabelle für Administratoren*

Auffallend ist die Tatsache, dass Administratoren oftmals nicht zu den wichtigen Gruppen gezählt und beim Entwurfsprozess von digitalen Bibliotheken außer Acht gelassen werden. Im Endeffekt entscheiden sie jedoch darüber, ob ein bestimmtes Produkt eingesetzt wird oder ob Alternativen in Betracht gezogen werden. Sie sind damit zu der wichtigsten Gruppe zu zählen.

2.3 Vergleich digitaler Biliotheken

Im folgenden Abschnitt werden kurz einige der interessantesten Entwicklungen und Neuerungen im Bereich digitaler Bibliotheken vorgestellt. Zuerst wird die Philosophie und der Hintergrund der Entwicklung der Systeme angerissen. Danach wird insbesondere auf die Bereiche Architektur, Usability, Flexibilität und die verwendeten Technologien eingegangen. Über die Robustheit dieser Systeme kann im Allgemeinen, aufgrund der relativ geringen Nutzungszeit im Rahmen dieser Diplomarbeit, kein aussagekräftiges Urteil gefällt werden.

2.3.1 MyCoRe

Diese digitale Bibliothek eignet sich aus mehreren Gründen hervorragend als Anwendungsbeispiel. Einerseits ist die Universität Rostock zum Teil unmittelbar in den Entwicklungsprozess integriert und zum Anderen soll dieses System, wenn es in einem finalen Status vorliegt, auch in verschiedenen Fakultäten der Universität Rostock eingesetzt werden. Dadurch muss ein gewisser Grad an Kompatibilität bzw. Interoperabilität mit der in dieser Arbeit zu entwerfenden Schnittstelle gewährleistet werden. Es gilt zu untersuchen, an welchen Stellen der Architektur und in welchem Umfang dies möglich ist.

Einleitung und Entstehungshintergrund

MyCoRe geht von folgender Ausgangssituation aus: *„An deutschen Universitäten werden zunehmend multimediale Daten sowohl für die Forschung als auch zur Unterstützung der Lehre in elektronischer Form erstellt. [..] Bestände müssen in einer geeigneten Umgebung systematisch erschlossen und auch langfristig zugänglich gemacht werden, eine Aufgabe, die von einer digitalen Bibliothek geleistet werden kann. Dabei werden die multimedialen Daten mit Metadaten versehen, um eine späteres Auffinden oder Zusammenstellen bestimmter Dokumente zu erleichtern.“* [LUE04]

Unter dieser Prämisse entstand Ende 1997 das Projekt MILESS (Multimedialer Lehr- und Lernserver Essen) an der Universität Essen. MILESS war dabei aber lediglich auf die lokalen Bedürfnisse dieser Universität angepasst. Um ein allgemeingültiges Modell zu erstellen, war zwingend eine Erweiterung und Flexibilisierung des Metadatenmodells erforderlich. Daraus entstand Anfang 2001 das MyCoRe-Projekt.

Unterliegende Architektur

Aus dem Aufbau des MyCoRe-Komponentenmodells, wie in Abbildung 6 dargestellt, wird ersichtlich, dass der MyCoRe-Kern sowohl Daten- und Objektmodell, Services und Präsentation, Autoren- und User-Unterstützung enthält und ebenso die Speicherung der Daten vorsieht. Damit ist der MyCoRe-Kern jedoch nicht als reine Schnittstelle zwischen dem Nutzer bzw. der Applikation und der Datenspeicherung in einer Datenbank entworfen worden. Es wird vielmehr der Versuch unternommen, die Speicherung in das Projekt zu integrieren.

Dies stellt ein wichtigen Unterschied zu ***LibScens*** dar. In diesem Zusammenhang wird gerade auf eine Trennung der unterschiedlichen Schichten geachtet. Dabei wird die Datenbank völlig getrennt von der Schnittstelle betrachtet. Da MyCoRe die Datenspeicherung integriert, wird natürlich auch die Metadatenstruktur von MyCoRe festgelegt. Sie ist jedoch so ausgelegt, dass die Möglichkeit besteht, neue Typen zu erstellen und in das Projekt zu integrieren. Dabei bestehen die Metadatenobjekte aus drei Komplexen:

- `structure` - Beinhaltet Informationen zur Einordnung der Datensätze in eine logische Struktur. Verweise zu Eltern- und Kindmetadatensätze.
- `metadata` - Hier werden die eigentlichen Metadaten entsprechend des Datenmodells gespeichert. Die Notation der Daten erfolgt dabei in XML-Form.
- `service` - Informationen zur Verwaltung des Metadatensatzes, z.B. Erstellungsdatum oder Verwaltungs-Flags.

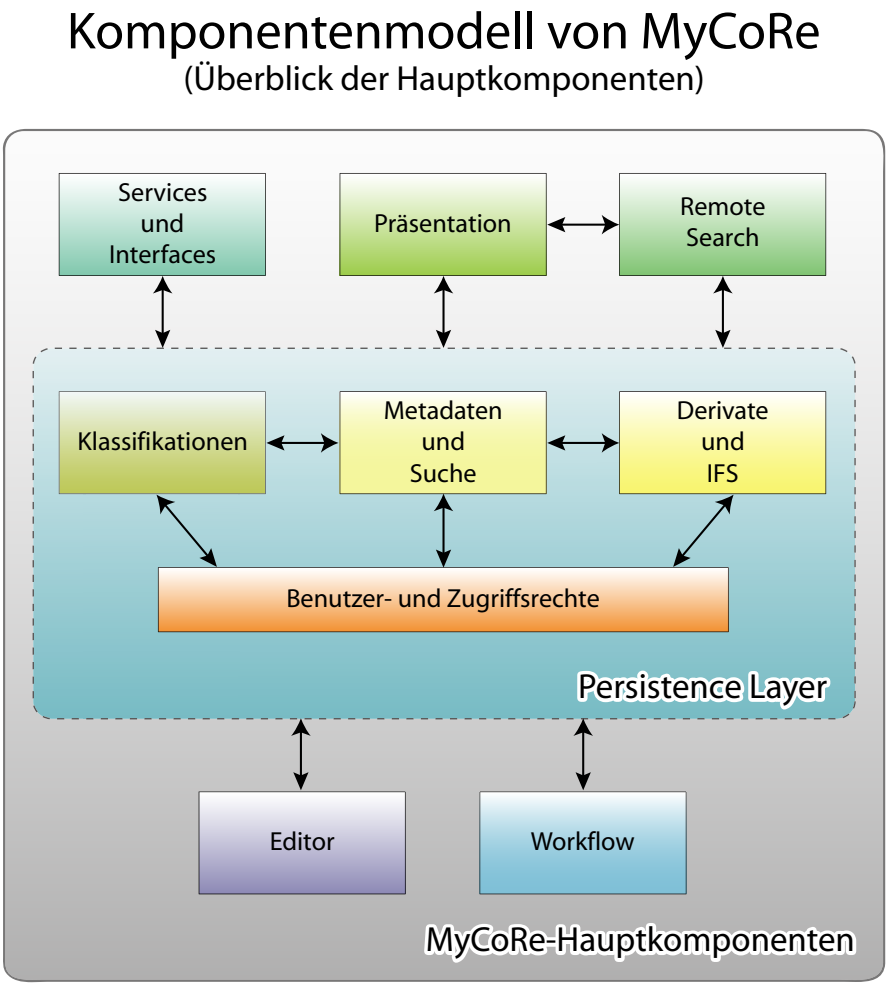

Abbildung 6: *Komponentenmodell von MyCoRe*

Das grobe Schichtenmodell von MyCoRe (siehe Abbildung 7) verdeutlicht examplarisch den Einsatz von MyCoRe in räumlich getrennten Umgebungen. Über eine Anwendung (Applikation B), welche in beiden Standorten vorliegt, wird die Kommunikation und die Integration der beiden MyCoRe-Laufzeitumgebungen realisiert.

Verwendete Technologien

MyCoRe wurde größtenteils in Java [JAV04] implementiert. Damit ist man weitgehend unabhängig vom benutzten Betriebssystem und kann einen hohen Grad an Kompatibilität garantieren.

Für die dynamische Gestaltung der Internetanbindung hat man sich konsequenterweise für Java Servlets [SER04] entschieden. Für die Speicherung der Metadaten wird das momentan weit verbreitete Format der Extensible Markup Language (XML) [XML04] benutzt, welche wiederum mittels Extensible Stylesheet Language Transformations (XSLT) [XSL04] in nahezu beliebige Ausgabedokumente umgewandelt werden können.

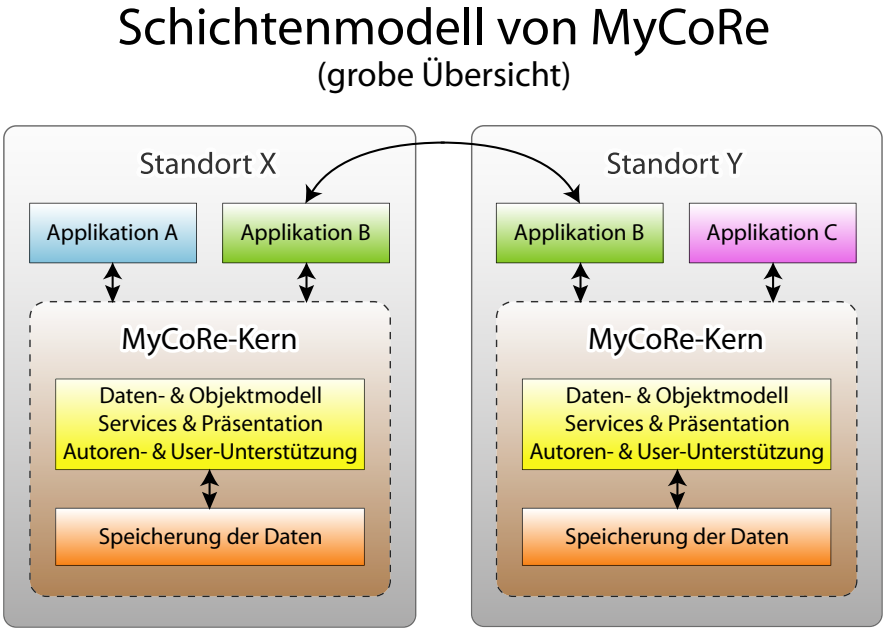

Abbildung 7: *Grobes Schichtenmodell von MyCoRe*

Eine Suche auf Metadaten wird in MyCoRe durch eine extrem vereinfachte Version der XPath-Spezifikation [XPA04] des W3C umgesetzt. Dabei wurden nur die von den Programmierern als sinnvoll erachteten Elemente implementiert und um die aus der Text-Suche bekannten Operatoren `like` und `contains`, wie beispielsweise bei dem IBM Net Search Extender [NSE04], erweitert.

MyCoRe verwaltet nicht nur die Metadaten, sondern auch die dazugehörigen Dateien selbst. Dazu besitzt MyCoRe ein internes Dateisystem (Internal File System, IFS)[7].

Flexibilität der Komponenten

Durch die einschneidend vereinfachten XPath-Anfragen werden natürlich die Möglichkeiten zur Suche eingeschränkt. Die Entwickler von MyCoRe liefern für diesen Umstand die Erklärung, dass die Nutzung verschiedener Persistence-Layer zur Speicherung der Metadaten die oben genannten Einschränkungen erforderlich macht, da eine Abbildung von komplexen Abfragen auf die verwendete Speicherschicht möglichst identisch funktionieren soll. Leider gibt es dabei doch Unterschiede in der Qualität der verwendbaren Produkte (IBM Content Manager 8 oder die XML:DB eXist). [LUE04]

Dieser Ansatz kann in Spezialfällen von komplexen Suchkonstrukten dazu führen, dass diese nicht adäquat umgesetzt werden können, sollten die Konstrukte von XPath dafür nicht ausreichend sein. Anstatt die Anfragesprache zu beschränken, könnte es eventuell sinnvoller sein, ein Interface zur Anbindung beliebiger Anfragesprachen zu definieren. *Eine* Implementierung dieses Interfaces könnte dann die XPath-Umsetzung sein. Ein anderer Weg wäre eine Realisierung von einer umfangreichen Anfragesprache der Schnittstelle auf eine Anfragesprache der Datenbank mittels eines Wrappers. Dies würde jedoch einen erheblichen Mehraufwand bedeuten.

[7] Die Verwaltung der Metadaten kann ebenfalls durch Fremdsoftware vorgenommen werden, wie z.B. IBM DB2 oder IBM Content Manager.

Usability des Systems

Da sich das MyCoRe-System noch in der Entwicklungsphase befindet, ist bisher auch nur ein Prototyp als Beispiel verfügbar. In Abbildung 42 auf Seite 90 wird die exemplarische Verwendung der Nutzerschnittstelle von MyCoRe dargestellt (*http://ibmdlh.rz.uni-leipzig.de/mycoresample/*).

Die Nutzbarkeit des Systems aus der Sicht des Standardnutzers ist sehr gut. Für einen Laien und Gelegenheitsnutzer ist der Aufbau klar erkenntlich, die Suche intuitiv und die Ergebnisse werden deutlich repräsentiert.

Aus Sicht des Expertennutzers werden jedoch schon einige Schwächen deutlich. So fehlt z.B. die Möglichkeit, erweiterte, personalisierte Einstellungen vorzunehmen. Alternative Repräsentationsmöglichkeiten wären ebenfalls wünschenswert.

Für Administratoren gibt es derzeit noch viel Handlungsbedarf. So gestaltete es sich teilweise als schwierig, das System voll funktionsfähig zu installieren. Auch die interaktiven Möglichkeiten zur Änderung bzw. Modifikation von MyCoRe durch den Administrator sind eingeschränkt. Zurzeit ist es beispielsweise nur über die Kommandozeile möglich, Datenbündel zu importieren oder zu aktualisieren. Dateien interaktiv über den Browser in das System hochzuladen ist zwar vorgesehen, jedoch noch nicht umgesetzt. Gerade aus Usability-Sicht stellt dies eine Schwachstelle dar. Zwar kann eine Kommandozeile eine sinnvolle Ergänzung zu einer GUI sein, die grafische Schnittstelle muss in modernen Systemen jedoch immer Vorrang haben. Auch für die Benutzer- und Zugriffsrechteverwaltung existieren momentan keine grafischen Verwaltungsschnittstellen oder Editoren. Eine Einbindung von externen Benutzerverwaltungen (z.B. an LDAP-Server) ist ebenfalls noch nicht vorhanden, jedoch für zukünftige Entwicklungen geplant.

Damit stellt MyCoRe eines der Produkte dar, in dem nicht alle Nutzergruppen adäquat unterstützt werden. Während die Standardnutzer komfortabel mit dem System interagieren können, wurden die Expertennutzer und Administratoren nicht gut in den Entwicklungsprozess integriert bzw. nicht als wichtige Nutzergruppen im Designprozess identifiziert. Dies ist ein klarer Nachteil, bedenkt man, dass die Aministratoren letztendlich darüber entscheiden, welche digitale Bibliothek eingesetzt wird. Die Programmierer versuchen auf der einen Seite die Werkzeuge zu vereinfachen und einzugrenzen (vereinfachte XPath-Spezifikation), bietet den Experten jedoch auf der anderen Seite keine komfortablen Systeme an, um diese einfach nutzen zu können. Einige der angesprochenen Schwächen werden sicherlich in der nächsten Veröffentlichung von MyCoRe (Version 0.9) behoben.

Zusammenfassung und Fazit

MyCoRe liefert gute Ansätze zum Umgang mit digitalen Bibliotheken. Einige Funktionalitäten sind eingegrenzt (wie z.B. die extrem vereinfachte XPath-Spezifikation), andere sind in der Basisversion verfügbar, können jedoch erweitert werden (beispielsweise das Schema und die Structures). Da sich MyCoRe noch im Entwicklungsstadium befindet, gibt es noch viele Baustellen und oftmals keine ausgereiften Lösungen. Die Umsetzung der Usability ist dabei sehr unausgewogen. Sie konzentriert sich zu stark auf den Endnutzer und verliert dabei die Experten und Administratoren aus den Augen.

Da sich die Universität Rostock dem Projekt angeschlossen hat, ist eine Integration von ***LibScens*** und MyCoRe wünschenswert. Die Konzepte divergieren jedoch häufig sehr stark, so dass überprüft werden muss, inwiefern eine Kopplung sinnvoll und möglich ist.

2.3.2 Greenstone

Das Greenstone-Projekt befindet sich schon in einem sehr ausgereiften Stadium und wird deshalb auch schon in vielen digitalen Bibliotheken eingesetzt. Dieses Projekt bietet sich als Kontrast zum vorher besprochenen MyCoRe-Projekt an, weil es zum Beispiel auf komplett anderen Technologien basiert.

Einleitung und Entstehungshintergrund

Bei Greenstone handelt es sich um eine Zusammenstellung von Software zum Erstellen und Verteilen von Kollektionen digitaler Bibliotheken. Es stellt einen neuen Weg zur Organisation von Informationen und deren Veröffentlichung über das Internet oder auf CD-ROM dar. Greenstone wurde durch das New Zealand Digital Library Project der Universität Waikato erstellt und wird in Zusammenarbeit mit der UNESCO und der Human Info NGO entwickelt und vertrieben. Es handelt sich dabei um Open Source Software, welche unter *http://www.greenstone.org* gemäß den Bestimmungen der GNU - General Public License verfügbar ist. [BAI03]

Greenstone stellt ebenso wie MyCoRe keine reine Schnittstelle dar, sondern ein komplettes System für digitale Bibliotheken. Dies wird auch daran deutlich, dass man in den Architekturschemata keine unabhängige Komponente für die Datenbank vorsieht, sondern diese immer in eine Schicht integriert wird. Greenstone liefert standardmäßig ein Information-Retrieval-System mit. Trotzdem ist es möglich, gewisse Komponenten, wie beispielsweise die Datenbank, im Nachhinein auszutauschen.

Das gesamte New Zealand Digital Library Project wurde so konzipiert, dass von der Erstellung über die Importierung, Änderung und Eliminierung von Datenbeständen prinzipiell alles mit Greenstone möglich ist. So kann durch Einsatz des so genannten `Collectors` eine komplette Kollektion mittels einer grafischen Nutzerschnittstelle kreiert werden. Dies beinhaltet auch das Anlegen der entsprechenden Datenbankstrukturen, wie z.B. das Erstellen von unterschiedlichen Indizes.

Unterliegende Architektur

Abbildung 8 zeigt die Prozessstruktur von Greenstone. Dabei sind oben mehrere Nutzer angeordnet, welche durch Computerterminals repräsentiert werden. Sie greifen auf unterschiedliche Greenstone-Kollektionen zu. Diese Kollektionen wurden zuvor einem Import- und Aufbauprozess unterzogen. Zuerst werden die Dokumente, unten in der Abbildung angeordnet, in das XML-kompatible Greenstone-Archivformat importiert. Daraufhin werden unterschiedliche durchsuchbare Indizes und eine Kollektionsinformationsdatenbank erstellt, welche die zum Browsen benötigte hierarchische Struktur beinhaltet. Nach diesem Prozess ist die Kollektion fertig und kann auf eingehende Nutzeranfragen reagieren. Zwei Komponenten sind dabei zentral für das Design des Laufzeitsystems:

- `Receptionist` - der Empfänger
- `Collection Servers` - die Kollektionen-Server

Aus Nutzersicht ist der Empfänger die Verbindung zur digitalen Bibliothek. Er akzeptiert Nutzereingaben, welche typischerweise in Form von Keyboard-Einträgen und Mouse-Klicks erfolgen,

analysiert diese und schickt anschließend eine Anfrage zum entsprechenden Kollektionen-Server. Dieser ermittelt die erfragten Informationen und gibt sie an den Nutzer zur Präsentation zurück.

Der Kollektionen-Server agiert praktisch als abstrakter Mechanismus, der den Inhalt der Kollektionen verwaltet. Der Empfänger dagegen ist für die Nutzerschnittstelle verantwortlich. Beide kommunizieren über ein definiertes Protokoll. In Abbildung 8 geschieht dies mittels eines so genannten „Null-Protokolls“, wobei sich Empfänger und Kollektionen-Server auf ein und demselben Rechner befinden.

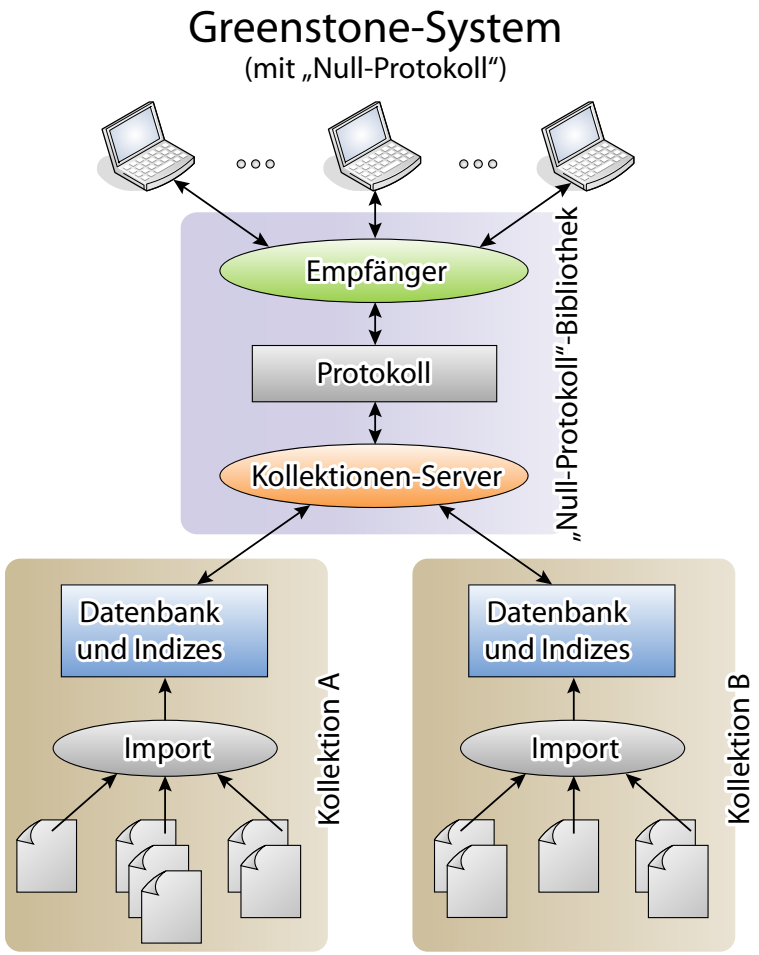

Abbildung 8: *Greenstone-System in Verbindung mit dem „Null-Protokoll“*

Das konzeptuelle Framework wird in Abbildung 9 deutlich. Sie zeigt die Hauptteile der Greenstone-Laufzeitumgebung. Zuerst initialisiert der Empfänger dabei seine Komponenten. Dann werden die CGI-Argumente geparst, um zu überprüfen, welche Aktion aufgerufen werden soll. Bei der Durchführung der Aktion nutzt die Software das „Null-Protokoll“, um auf den Inhalt der Kollektionen zuzugreifen. Die Antwort dient dazu, eine entsprechende Internetseite zu generieren. Dies geschieht mit Unterstützung der Formatkomponenten und Makrosprache.

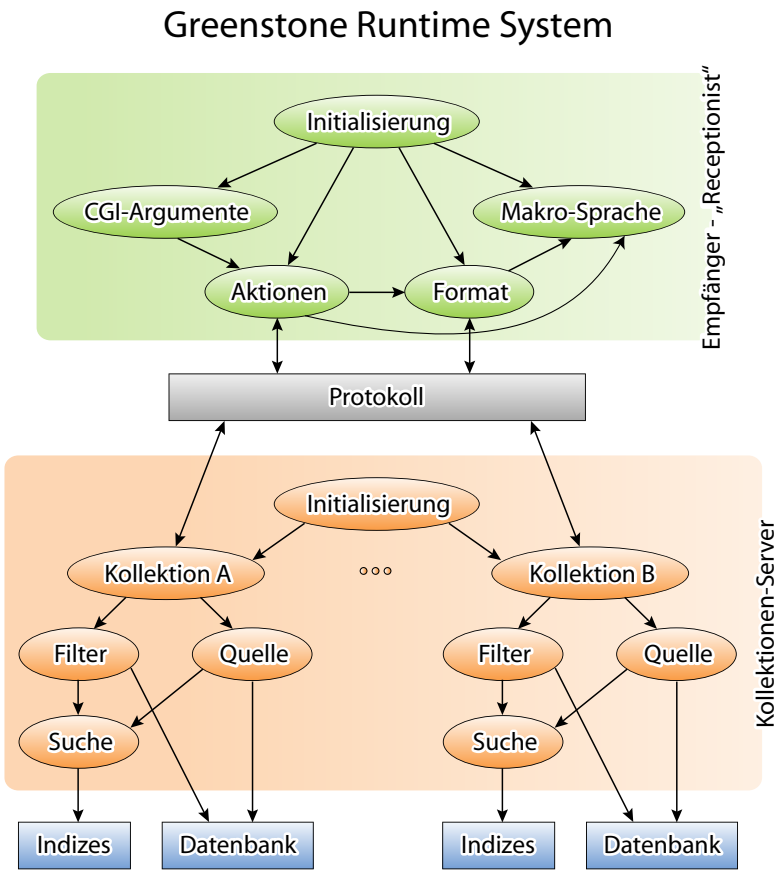

Abbildung 9: *Das Greenstone-Runtime-System*

Verwendete Technologien

Im Gegensatz zu MyCoRe setzt man bei Greenstone auf die Programmiersprache C++ zur Realisierung der Laufzeitumgebung. Eine weitere Programmiersprache, die bei Greenstone ausgiebig genutzt wird, ist Perl [PER04]. So werden z.B. bei der Erstellung von Kollektionen Perl-Skripte gestartet, welche für den Import der Daten im Ausgangsformat in die interne Struktur von Greenstone verantwortlich sind. Außerdem wird Perl zur Generierung der dynamischen Internetseiten verwendet. Dazu wird das Common Gateway Interface (CGI) für Perl genutzt.

```
<!DOCTYPE GreenstoneDirectoryMetadata [
  <!ELEMENT DirectoryMetadata (FileSet*)>
  <!ELEMENT FileSet (FileName+,Description)>
  <!ELEMENT FileName (#PCDATA)>
  <!ELEMENT Description (Metadata*)>
  <!ELEMENT Metadata (#PCDATA)>
  <ATTLIST Metadata name CDATA #REQUIRED>
  <ATTLIST Metadata mode (accumulate|override) "override">
]>
```

Abbildung 10: *Greenstone-Archivformat: Document Type Definition (DTD).*

Bei der internen Repräsentation der Datensätze wird bei Greenstone ebenfalls XML [XML04] eingesetzt. Diese Form der Speicherung wird als Greenstone-Archivformat deklariert und enthält neben den eigentlichen Inhalten der unterschiedlichen Datensätze auch deren Metadaten. Die Portierung von unterschiedlichen Eingabedatentypen in XML erfolgt dabei wiederum durch Perl-Plugins. Die DTD wird in Abbildung 10 und ein Beispiel-XML-Dokument in der Abbildung 11 dargestellt.

```
<?xml version="1.0" ?>
<!DOCTYPE GreenstoneDirectoryMetadata SYSTEM
"http://greenstone.org/dtd/GreenstoneDirectoryMetadata/1.0/
GreenstoneDirectoryMetadata.dtd">
<DirectoryMetadata>
  <FileSet>
    <FileName>nugget.*</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse</Metadata>
      <Metadata name="Place" mode="accumulate">Nugget Point</Metadata>
    </Description>
  </FileSet>
  <FileSet>
    <FileName>nugget-point-1.jpg</FileName>
    <Description>
      <Metadata name="Title">Nugget Point Lighthouse</Metadata>
      <Metadata name="Subject">Lighthouse</Metadata>
    </Description>
  </FileSet>
</DirectoryMetadata>
```

Abbildung 11: *Greenstone-Archivformat: XML-Datei für einen Beispieldatensatz.*

Als Back-End wird bei Greenstone eine Kombination von Managing Gigabytes (MG) [WIT99] und des GNU Database Managers (GDBM) [GDB99] eingesetzt. Bei MG handelt es sich um Software zur Kompression und Indizierung von Dokumenten und Bildern, die von Ian H. Witten, Alistair Moffat und Timothy C. Bell erstellt wurde und sich praktisch auf dem Stand von 1999 befindet. Der GDBM ist die GNU-Implementation der Standard-Unix-DBM-Bibliothek, welche ursprünglich von Berkeley entwickelt wurde. GDBM stellt einen Standardteil von Linux dar [BAI03] und muss unter Windows zusätzlich installiert werden. Die letzten Versionen dieses Datenbankmanagers befinden sich ebenfalls auf dem Stand von 1999. Diese von Greenstone angebotene Lösung scheint vor dem Hintergrund der raschen Entwicklung im Bereich der Datenbanken und der Tatsache, dass die eingesetzten Produkte nunmehr schon fünf Jahre alt sind, als nicht mehr zeitgemäß. Greenstone räumt jedoch ein, dass es durchaus möglich ist, eine alternative Datenbank oder andere Retrieval Systeme anzubinden. Inwieweit dies tatsächlich praktisch realisierbar ist, konnte im Rahmen dieser Diplomarbeit nicht eruiert werden.

Die Anpassung der Nutzerschnittstelle wird mittels einer eigens für Greenstone entwickelten Makrosprache gewährleistet. „Diese Makrosprache wurde entwickelt, um ein digitales Bibliothekensystem bereitzustellen, welches zum einen einen konsistenten Stil besitzt und zum anderen als Schnittstelle für unterschiedliche Programmiersprachen fungiert.“ [BAI03]. Auch diese Ansicht ist überholt. Hierfür wurden in der Zwischenzeit andere Open-Source-Techniken wie beispielsweise XSLT entwickelt. Diese lassen sich wesentlich besser in XML einbetten und stellen heute eine Standardlösung für diesen Bereich dar. Administratoren der digitalen Bibliothek sollten sich in derartige Systeme schneller einarbeiten können, da auch z.B. im Internet viele Beispiele und reichhaltige Dokumentationen zu finden sind.

Zusätzlich zu den oben genannten Technologien müssen eventuell zusätzliche externe Programme, wie beispielsweise das GNU Image Manipulation Program (GIMP), installiert werden.

Trotz divergierender Techniken kann Greenstone, ebenso wie MyCoRe, auf unterschiedlichen Plattformen ausgeführt werden. Auf diese Weise sind beide Systeme sowohl unter Unix als auch unter Windows lauffähig. Allerdings sind dafür einige systemspezifische Eigenheiten zu beachten, weshalb die Installation eines gleichen Systems auf unterschiedlichen Betriebssystemen oftmals voneinander differiert.

Flexibilität der Komponenten

Da Greenstone grundsätzlich als eine Gesamtlösung für digitale Bibliotheken entworfen wurde, ist z.B. die Flexibilität der Datenbankanbindung natürlich nicht in einem Ausmaß gegeben, wie dies bei einer dreistufigen Schnittstelle (Nutzer - Schnittstelle - Datenbank) der Fall wäre. Greenstone gestattet jedoch den Austausch des Back-End (Managing Gigabytes) durch Alternativen.

Das User Interface von Greenstone hätte dagegen flexibler arrangiert werden können. Jede grundlegende Abweichung vom Standardbeispiel gestaltet sich sehr schwierig und muss über die eigens für Greenstone entwickelte Makrosprache und Formatkomponenten umgesetzt werden. Dies bedeutete zwangsweise ein hohes Maß an Einarbeitungsaufwand für Administratoren.

Usability des Systems

Die unverkennbare Oberfläche von Greenstone wird in Abbildung 43 auf Seite 91 dargestellt. Hierbei handelt es sich um eine Kollektion von Autoren Alabamas aus dem 20. Jahrhundert (*http://diglib.auburn.edu/default.php*).

Die Gestaltung des User Interfaces für den Endanwender ist bei Greenstone, bis auf kleinere Ausnahmen, gut umgesetzt worden. Auf den Einsatz von aufwändigen Grafiken wird bewusst verzichtet. Dieser Umstand muss jedoch nicht zwingend negativ gewertet werden, da der Einsatz von visuellen Effekten in Nutzerschnittstellen gut geplant sein will. Nicht selten konfrontieren Hersteller von Internetseiten den Anwender mit einer Bilderflut, so dass dieser sich letztendlich überfordert fühlt und somit das Ziel nicht erreicht wurde. Dies ist bei Greenstone nicht der Fall. Der Nutzer findet sich sofort zurecht, da die Informationen gut strukturiert aufbereitet werden. Man hätte dem Ersteller der Nutzerschnittstelle für Greenstone jedoch mehr kreativen Spielraum zugestehen können.

Sehr positiv fällt bei diesem Projekt die gute Dokumentation und Hilfe auf. Sie ist klar strukturiert und wird detailliert dargestellt. Hinzu kommt, dass sie weitgehend in unterschiedlichen Sprachen verfügbar ist. Auch bei der Installation wird explizit auf Spezialfälle, Fehler und Möglichkeiten zu deren Lösung hingewiesen.

Die Erstellung von Kollektionen mittels des `Collectors`, wie oben beschrieben, ist jedoch laut [BAI03] nur für Standardkollektionen empfehlenswert. Es ist hingegen nicht für Kollektionen geeignet, welche eine komplett andere Struktur haben als das angegebene Standardbeispiel. Dazu muss die Kollektionskonfigurationsdatei manuell angepasst werden, welches laut deren Entwicklern [BAI03] ein erhebliches Vorwissen über die Architektur und die verwendeten Sprachen von Greenstone voraussetzt. Dies ist der Usability natürlich nicht zuträglich, lässt sich aber bei derart komplexen Mechanismen kaum vermeiden. Auch wird in diesem Fall für den so genannten `Build`-Prozess eine Kommandozeile bemüht, wofür zumindest ein intensives Durchlesen der Anleitung

erforderlich ist. Ein großer Nachteil von Greenstone ist, dass beim Hinzufügen eines Dokumentes zu einer Kollektion die gesamte Kollektion komplett neu gebildet werden muss.

Obwohl bei Greenstone relativ sparsam mit Bildern umgegangen wird, ist z.B. für jede Kollektion ein unterschiedliches Bild für deren Darstellung vorgesehen. Durch diese simple Möglichkeit wird die Unterscheidung von Kollektionen überraschenderweise ungemein erleichtert. Jede Kollektion besitzt außerdem eine angepasste Startseite, die Informationen über die Kollektion angibt und die Möglichkeiten erörtert, wie man am effektivsten die Kollektion nach Informationen durchsucht.

Dem Anwender wird die Möglichkeit gegeben, einige Nutzereinstellungen nach persönlichen Vorlieben anzupassen. Die Personalisierung hätte aber wesentlich ausführlicher ausfallen können. So wäre zumindest noch eine Auswahl der Ergebnisanzahl pro Suchseite und der Schriftgröße wünschenswert.

Zusammenfassung und Fazit

Greenstone ist ein gereiftes Produkt. Es findet zunehmend Einsatz in nichtkommerziellen Bereichen von digitalen Bibliotheken. Im Gegensatz zu MyCoRe setzt man auf andere Implementierungstechniken wie C++ oder Perl. Greenstone stellt ebenfalls eine Komplettlösung dar, wobei Managing Gigabytes als Back-End dient.

Einfache Änderungen der Schnittstelle sind mit grafischen Editoren möglich, komplexe Anpassungen müssen jedoch manuell eingepflegt werden. Dazu dient eine eigens entworfene Makrosprache, welche ein hohes Maß an Einarbeitungszeit erfordert. Allgemein wurden jedoch alle Nutzergruppen in den Entwicklungsprozess integriert und sind somit besser ausbalanciert, als es bei MyCoRe der Fall war.

2.3.3 Weitere digitale Bibliotheken

Neben den beiden digitalen Bibliotheken der vorherigen Abschnitte, welche intersiver inspiziert wurden, gibt es eine Vielzahl alternativer Umsetzungen, die nun betrachtet werden. Einen komprimierten Vergleich von frei verfügbaren digitalen Bibliotheken gibt Raym Crow in *„A Guide to Institutional Repository Software“* [CRO04]. Die Auswertung der dort gezeigten Übersichten ist unmittelbar in die Anforderungen und Ziele von ***LibScens*** eingeflossen.

Eprints

Die Eprints-Software hat die größte und weitverbreitetste Installationsbasis aller verglichenen digitaler Bibliotheken. Entwickelt an der Universität von Southampton, wurde die erste Version gegen Ende des Jahres 2000 veröffentlicht. Das Projekt wurde ursprünglich von CogPrints[8] gesponsort, wird nun jedoch durch JISC[9] als Teil des „Open Citation Project“ und NSF[10] unterstützt.

Eprints weltweit installierte Basis bringt ein umfassendes Netzwerk an Unterstützung für neue Implementierungsversionen mit sich. Die große Anzahl der installierten Eprint-Archive lässt darauf schließen, dass ein Institut das System relativ schnell erstellen und lauffähig machen kann, ohne dabei ein großes Maß an technischen Fachkenntnissen vorauszusetzen. Die Anzahl der Eprints-Installationen, welche die Basismöglichkeiten des Systems erweiterten — beispielsweise indem wei-

[8] CogPrints: Cognitive Sciences Eprint Archive.
[9] JISC: Joint Information Systems Committee.
[10] NSF: National Science Foundation.

terführende Suchalgorithmen integriert oder Metadaten erweitert wurden — zeigen auf, dass das System einfach modifiziert werden kann, um sich an lokale Gegebenheiten anzupassen. [CRO04]

In Abbildung 44 auf Seite 91 ist beispielhaft eine Oberfläche für das Eprints-Projekt zu sehen. Hierbei handelt es sich um ein Eprints2-Archiv, das von der Australian National University eingesetzt wird (*http://eprints.anu.edu.au/*).

i-Tor

i-Tor - Tools and Technologies for Open Repositories - wurde von der Sektion Innovative Technology-Applied (IT-A) des niederländischen Institutes für Wissenschaftliche Informationsdienste (NIWI) entwickelt. NIWI nennt i-Tor „eine Web-Technologie, durch die verschiedene Informationstypen mittels einer Netzschnittstelle dargestellt werden können“, unabhängig davon, wo die Daten gespeichert wurden oder in welchem Format sie vorliegen. i-Tor zielt darauf ab, eine „datenunabhängige“ Bibliothek zu implementieren, bei der der Inhalt und die Nutzerschnittstelle als zwei unabhängige Teile des Systems agieren. Deshalb fungiert i-Tor einerseits als OAI-Provider, dem es möglich ist, OAI-kompatible Bibliotheken und andere Datenbanken „abzugrasen“ bzw. einzusammeln (harvest), und andererseits stellt i-Tor einen Datenlieferanten dar.

Da i-Tor die Daten verschiedenster relationaler Datenbanken, Dateisysteme und Internetseiten verarbeiten kann, erlaubt das System den Instituten einen erheblichen Spielraum für die Organisation der digitalen Bibliothek. Es kann neue Datenbanken für die digitale Bibliothek generieren, aber auch auf bereits existierenden relationalen Datenbanken aufsetzen. Desweiteren unterstützt i-Tor die Sammlung von Daten direkt von einer persönlichen Homepage von Wissenschaftlern.

Aufgrund des Designs zwingt i-Tor den Designern keinen bestimmten Ablaufplan auf. Vielmehr bietet i-Tor Werkzeuge an, um jeden benötigten Ablaufplan von Organisationen umzusetzen, *ohne* den Ablauf in das i-Tor-System direkt zu integrieren. Das i-Tor-Design könnte eine angemessene Wahl für Institute sein, welche eine Bibliothek auf bestehende, ungleichartige digitale Bibliotheken aufsetzen wollen.

In Abbildung 45 auf Seite 92 wird ein Screenshot der i-Tor-Oberfläche (*http://www.i-tor.org/nl/toon*) dargestellt. Wie man dem Bild entnehmen kann, handelt es sich dabei jedoch vorerst nur um ein Testszenario der i-Tor-Engine.

CDSware

Die CERN-Dokumentenserversoftware (CDSware) wurde entwickelt, um den CERN-Dokumentenserver zu unterstützen. Die Software wird von CERN[11] gewartet und publiziert. Sie unterstützt elektronische Server für Vorabdrucke, Online-Bibliothekskataloge und andere webbasierte Dokumentenablagesysteme. CERN nutzt CDSware, um über 450 Datenkollektionen zu verwalten, die über 620.000 bibliografische Einträge und 250.000 Volltextdokumente umfassen, einschließlich Vorabdrucken, Zeitschriftenartikeln, Büchern und Fotografien.

[11] „Abkürzung für französisch Conseil Européen pour la Recherche Nucléaire, jetzt Organisation Européenne pour la Recherche Nucléaire (Europäische Organisation für Kernforschung) [..] , 1954 gegründete Organisation mit dem Ziel der gemeinsamen kernphysikalischen Grundlagenforschung; Sitz: Genf; 20 europäische Mitgliedsstaaten (2000). CERN betreibt den weltweit größten Verbund an Teilchenbeschleunigern.“ [BRO02]

CDSware wurde explizit für sehr große Bibliotheken erstellt, welche die verschiedensten Typen von Materialien beinhalten. Dies schließt multimediale Inhaltskataloge, Beschreibungen von Museumsobjekten, vertrauliche und öffentliche Dokumentreihen, etc. mit ein. Jede Version wird direkt unter strengen Auflagen in der CERN-Umgebung getestet, bevor sie für die Öffentlichkeit bereitgestellt wird.

2.4 Ableitung der Ziele zur Umsetzung der Schnittstelle

Im folgenden Abschnitt werden die im Kapitel 1.2 angerissenen Ziele um Forderungen erweitert, die sich aus der Auswertung der digitalen Bibliotheken (siehe Kapitel 2.3) und der Einbeziehung der generellen Designprinzipien an grafische Nutzerschnittstellen (siehe Kapitel 2.2) ergeben haben. Hier werden die konkreten Anforderungen und Ziele an ***LibScens*** feingranularer herausgearbeitet. Nachdem im Kapitel 2.1 der erste Schritt der „nutzerorientierten Gestaltung“ abgeschlossen wurde, repräsentiert dieser Teil der Arbeit folglich den zweiten Punkt: *„Spezifikation der Anforderungen“*. Die gesammelten Anforderungen und Ziele an ***LibScens*** sind im Einzelnen:

- Es muss eine Vielzahl divergierender Datenbanksysteme unterstützt werden. Die Einschränkung bestimmter digitaler Bibliotheken auf konkrete Datenbanksysteme, wie etwa das frei verfügbare MySQL oder die XML-Datenbank eXist, stellt ein klares Defizit dar [CRO04]. Falls möglich, gilt es eine Schnittstelle zu kreieren, welche eine beliebige Implementation unterschiedlicher Systeme erlaubt. Dies schließt sowohl relationale als auch objektorientierte bzw. objektrelationale Datenbanksysteme mit ein.

- Die technischen Fachkenntnisse aller beteiligten Nutzergruppen müssen so niedrig wie möglich angesetzt werden. Das schließt den Einsatz bekannter Open-Source-Techniken und -Werkzeuge mit ein. Beispielsweise stellt SQL eine weitverbreitete Anfragesprache an Datenbanken dar und XML kommt als Dialogsprache in Betracht. Völlig neu entworfene Verfahren verwirren oft selbst Expertennutzer und Administratoren, sind im Allgemeinen nicht gut wartbar und tragen nicht zur allgemeinen Akzeptanz des Systems bei.

- Erfahrenen Nutzern darf dabei nicht die Möglichkeit genommen werden, besondere Spezialitäten (z.B. DB-Abfragen, besondere Szenarios, etc.) in die Schnittstelle zu integrieren. Ein Grundsatz an komplexen Datentypen, Darstellungselementen und Szenarios muss dabei jederzeit verfügbar sein. Soweit dies möglich und sinnvoll ist, sollten Beispiele für den Umgang mit diesen Konstrukten gegeben werden.

- Derartige Besonderheiten z.B. von Datenbanken sollten sich nahtlos in die Architektur eingliedern oder diese erweitern, ohne dabei völlig andere Konzepte zu nutzen als diejenigen, die in der Standardumsetzung der digitalen Bibliothek vorgesehen sind. Es sollten also keine speziellen Anfragesprachen oder Makrosprachen neu entworfen werden, um das Ausgabeformat zu erstellen.

- Die Usability muss jederzeit und für alle Nutzerschichten der digitalen Bibliothek gewährleistet werden. Die Schnittstelle muss sich den Gegebenheiten eines Standardnutzers genauso anpassen wie der eines Administrators. Wie im Kapitel 2.3 nachzulesen ist, wird dieser Punkt nicht von allen vorhandenen digitalen Bibliotheken erfüllt.

- *„Konfigurieren statt Programmieren!“* - unter diesem Motto sollte die gesamte Schnittstelle konzipiert werden. Alle Standardszenarios müssen auf diese Weise erstellt, modifiziert und gewartet werden. Mittels Konfigurations- und Authentifizierungsdateien muss auf diesem Weg die Anzahl von fest programmierten Anteilen so gering wie möglich gehalten werden. Nur dadurch kann in der Laufzeitumgebung ein hoher Grad an Flexibilität gewährleistet werden, der einen Neustart bzw. eine erneute Kompilierung der Projektdateien überflüssig macht.

- Anbindungen an vorhandene Standardschnittstellen und bestehende Systeme sollten in die Konzeption integriert werden bzw. sollte die Schnittstelle um solche Spezialitäten erweiterbar sein. Zu diesen Schnittstellen zählen im Besonderen OAI und Dublin Core, als bestehendes System ist MyCoRe zu nennen.

- Die Implementierung sollte robust auf Fehlerfälle reagieren und den Nutzer immer mit Lösungsvorschlägen zu deren Vermeidung bzw. Lösung versorgen. Das macht den Einsatz von ausgeklügelten Logbüchern notwendig. Diese sollten Informationen differenzieren (Frage: *„Handelt es sich um eine Fehlermeldung oder Debug-Information?“*) und in unterschiedliche Ebenen unterteilen (Frage: *„Ist es eine allgemeine Mitteilung oder zählt die Meldung zu einer speziellen Session?“*).

- Grafische Oberflächen und Darstellungen bestimmter Datentypen müssen austauschbar sein, damit das System in unterschiedlichsten Bereichen eingesetzt werden kann. Dies setzt zwingend eine modulare Programmierung voraus. Falls möglich sollten Module, wie die Szenariobausteine, wiederum zur Laufzeit ausgewechselt werden können.

Kapitel 3

Konzeption

Die zentrale Aufgabe dieser Diplomarbeit besteht darin, ein Konzept für eine multimediale Nutzerschnittstelle im eNoteHistory-Musikarchiv zu entwickeln. Da in den vorhergehenden Abschnitten alle elementaren Voraussetzungen und Anforderungen an das Interface ermittelt und evaluiert wurden, folgt nun der Hauptteil: die Konzeption. Neben der Funktionsvielfalt spielt auch die Performance eine grundlegende Rolle. Zwar ist die Gesamtarchitektur völlig unabhängig von bestimmten Programmierungsdetails, dennoch kann oftmals ein Abwägen zwischen verschiedenen möglichen Implementierungsalternativen die Architektur entscheidend beeinflussen. Viele konzeptuelle Bausteine müssen deshalb auf deren praktische Durchführbarkeit und Leistung getestet und notfalls angepasst werden.

Bezogen auf den Entwicklungszyklus des „User Centered Design“ (siehe Kapitel 2.1), repräsentiert dieser Abschnitt den dritten Punkt des UCD: *„Produktion von Designlösungen“*.

3.1 Einordnung von *LibScens* in eNoteHistory

Bevor der Fokus auf die Architektur von ***LibScens*** gelegt werden kann, muss eine eindeutige Einordung der Schnittstelle in das eNoteHistory-Musikarchiv erfolgen. Das Projekt eNoteHistory beschäftigt sich mit der Schreiberidentifizierung in historischen Notenhandschriften. Dazu werden Algorithmen zur Handschrifterkennung, sowie der Klassifikation und Evaluation der Schriften entwickelt. Die Daten der Notendokumente werden in Dokumentenservern gespeichert. Auf die Informationen kann mittels unterschiedlicher Retrieval-Techniken zugegriffen werden. Ein wichtiges Teilprojekt stellt die automatisierte Bildverarbeitung und Bildanalyse der historischen Notendokumente dar. Dieser Abschnitt wird vom Fraunhofer Institut (Rostock) realisiert. Die fachspezifischen Anforderungen an eNoteHistory und an die Definition von Features werden vom Institut für Musikwissenschaft (Universität Rostock) aufgestellt und evaluiert. Der Schemaentwurf, die Datenverarbeitung und -repräsentation wird durch den Lehrstuhl für Datenbank- und Informationssysteme (Universität Rostock) bearbeitet. Die angesprochene Repräsentation soll dabei durch die Implementation dieser Diplomarbeit realisiert werden. Abbildung 12 verdeutlicht die Integration von ***LibScens*** in die Architektur des eNoteHistory-Musikarchives.

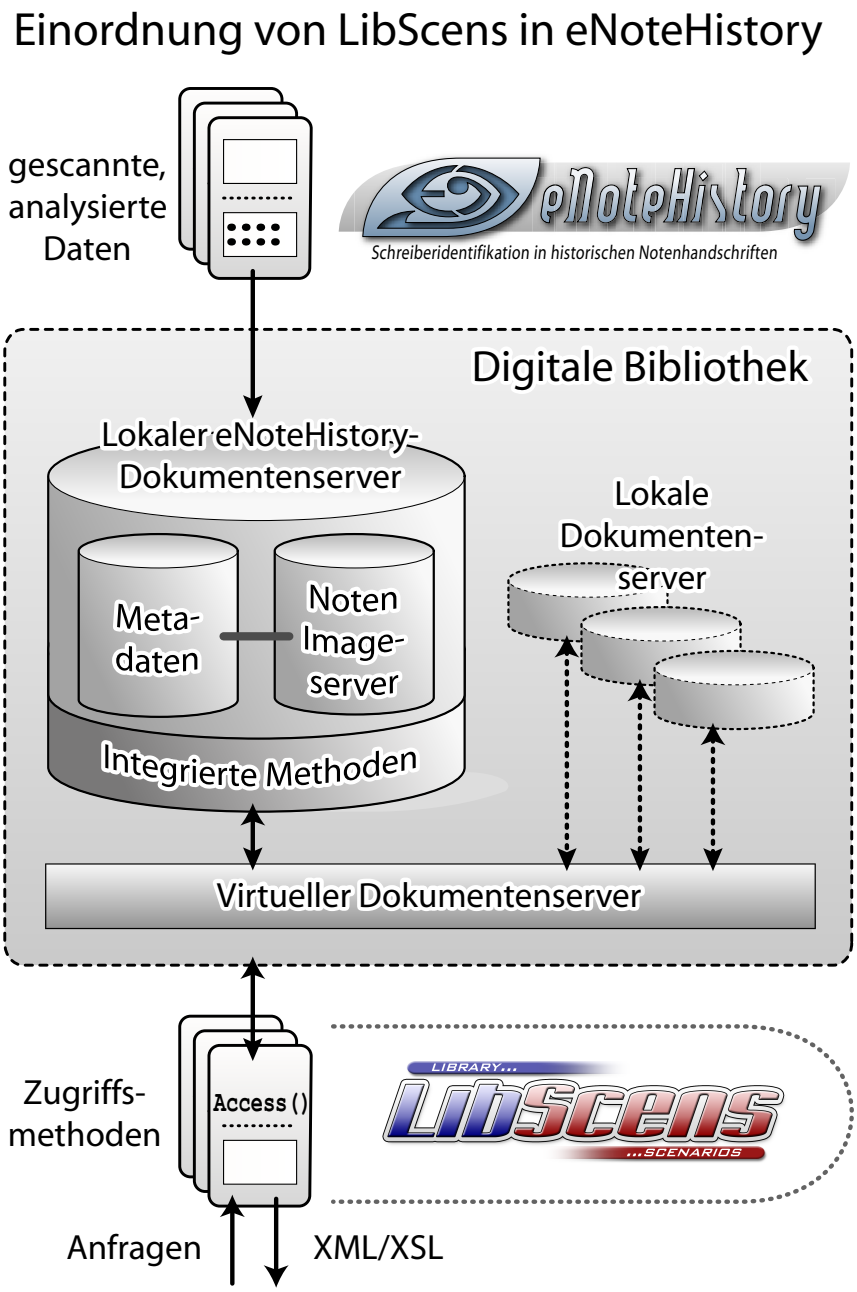

Abbildung 12: *Einordnung von **LibScens** in die eNoteHistory-Architektur*

Der dargestellte Prozess beginnt mit dem Import der gescannten und analysierten Daten in die digitale Bibliothek. Dies kann zurzeit nur manuell bewerkstelligt werden, an der automatisierten Version wird jedoch beim Fraunhofer Institut gearbeitet. In der digitalen Bibliothek werden die Daten in einem oder mehreren lokalen eNoteHistory-Dokumentenservern gespeichert. Derzeit handelt es sich bei dem eingesetzten Datenbanksystem um IBM DB2 in der Version 8.1. Geplant ist jedoch eine Migration der vorhandenen Daten in den IBM Content Manager. An der Umsetzung dazu wird innerhalb des Projektes gearbeitet. Ein lokaler Dokumentenserver besteht nun aus:

- den Metadaten der einzelnen eingescannten Musikwerke, wie z.B. den Schreibernamen, Entstehungszeitpunkt oder Signatur.
- dem Noten-Imageserver, welcher die zugehörigen Bilder in unterschiedlichen Versionen verwaltet. Dabei können die Bilder beträchtliche Größen von bis zu 30 MB besitzen.
- den integrierten Methoden, die beim Import bzw. Export der Daten angestoßen werden. Derartige Methoden sind beispielsweise Stored Procedures oder User Defined Functions.

Nach außen hin ist nur ein großer virtueller Dokumentenserver sichtbar, auf den mittels verschiedenartigen Zugriffsmethoden der Inhalt angesehen oder verändert werden kann. Diese Zugriffsmethoden empfangen die Anfragen von Nutzern und liefern als Ergebnis entweder XML- und XSL-Dokumente oder HTML als Kombination der beiden Dokumententypen zurück. Gerade die durch `Access()` gekennzeichneten Methoden in Abbildung 12 sollen von ***LibScens*** übernommen werden. Denkbar ist auch eine künftige Integration des Datenimports in die Schnittstellenarchitektur.

3.2 Überblick der unterliegenden Architektur

LibScens kann fortan völlig entkoppelt von eNoteHistory betrachtet werden, da das Konzept nicht an ein spezielles Projekt gebunden ist. Nachfolgende praktische Beispiele aus dem eNoteHistory-Musikarchiv dienen lediglich zur Verdeutlichung bestimmter Teilkonzepte und können ebenso durch adäquate Konstrukte anderer digitaler Bibliotheken ersetzt werden.

3.2.1 Schnittstellenarchitektur

Im Gegensatz zu vielen anderen digitalen Bibliotheken, die prinzipiell als zweischichtige Schnittstelle entworfen wurden, baut ***LibScens*** konsequent auf drei Schichten auf:

1. Nutzerschicht
2. Schnittstellenschicht
3. Datenbankschicht

Dieser Aufbau hat den Vorteil, dass spätere Wechsel auf andere Datenbanksysteme keine Anpassung in allen Schichten der Architektur erfordern und lediglich eine adaptierte Implementation an die neue Anfragesprache erfolgen muss. In Abbildung 13 werden die drei Schichten dargestellt.

Der Ablauf beginnt bei der Nutzereingabe. Der Anwender tätigt per Arbeitsstation eine Dateneingabe. Mögliche Arten der Eingabe sind Internetseiten, Kommandozeile oder angepasste, externe Programme. Die Daten werden daraufhin über das Netzwerk an die Schnittstelle gesendet. Sollte der Nutzer an demselben Rechner arbeiten, auf dem auch der Webserver läuft, also die ***LibScens***-Laufzeitumgebung, entfällt die Netzwerkschicht selbstverständlich. Die Nutzeranfrage beinhaltet neben der eigentlichen Anfrage auch noch zusätzliche Parameter wie `Post-` oder `Get`-Parameter bei HTML-Anfragen. Die Schnittstellenschicht empfängt die Anfrage durch *ein*[1]

[1] Theoretisch wäre auch ein Mapping der einzelnen Szenarios auf unterschiedliche URLs möglich, würde aber erhöhten Konfigurationsaufwand für den Server mit sich bringen. Dies müsste entweder von einem Administrator vorgenommen oder durch ***LibScens*** automatisiert werden. Da beide Varianten gleichmächtig sind, wurde die einfacher wartbare Methode gewählt.

Java Servlet. Das Servlet wählt nun je nach Request-Parametern das passende, angeforderte Szenario aus dem Pool aller vorhandenen XML-Szenariobausteine aus. Entprechende Szenarios sind beispielsweise:

- Navigation über Daten,
- Suche über Daten,
- Datenbefüllung und Nutzerverwaltung.

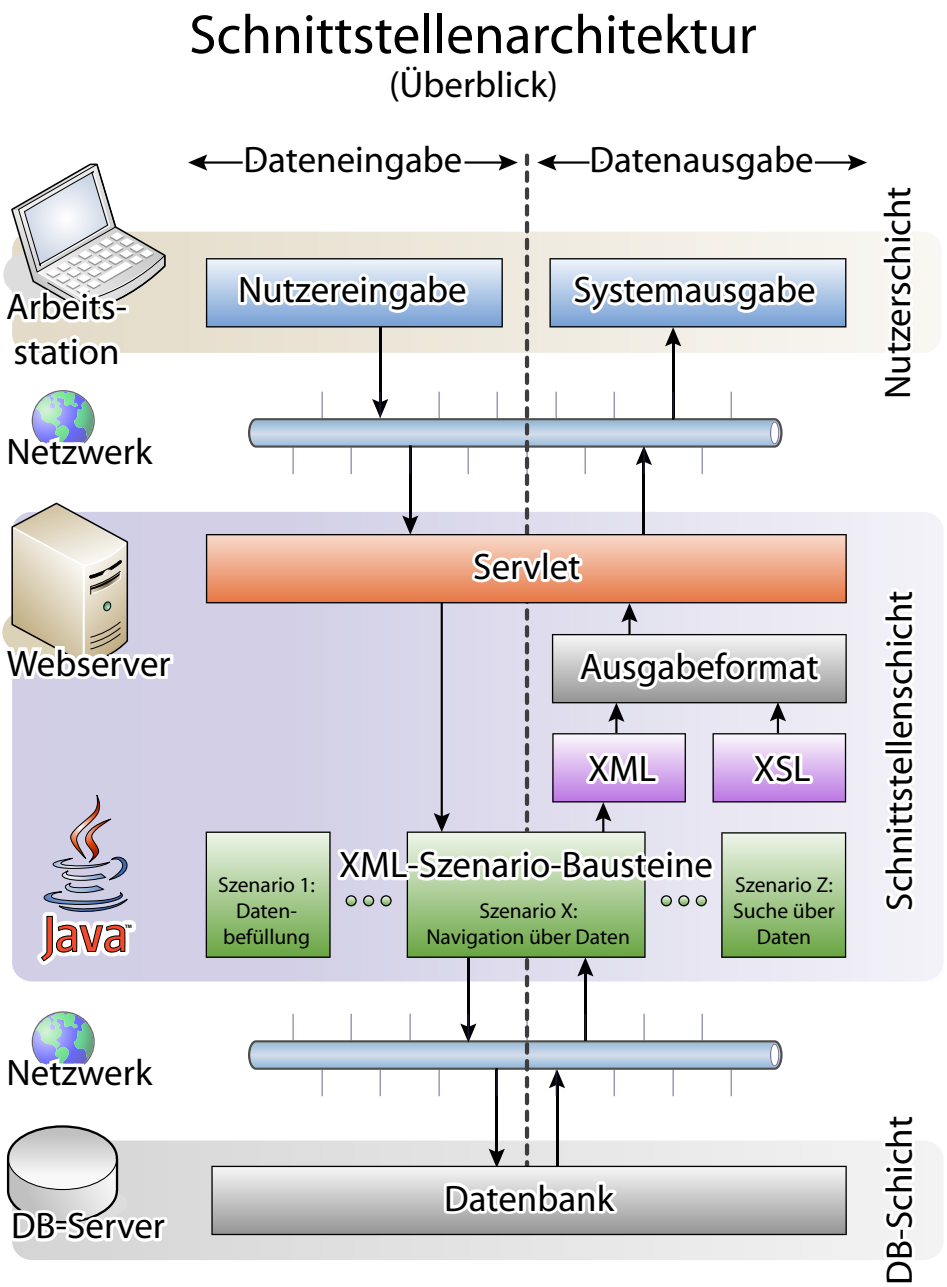

Abbildung 13: *Schnittstellenarchitektur von* ***LibScens***

Der Szenariobaustein wertet nun alle nötigen Attribute aus und stellt dafür im Allgemeinen auch eine Verbindung zur Datenbank her. Das erfordert erneut die Nutzung des Netzwerks, welche ebenfalls entfallen kann, sollte sich die Datenbank auf dem gleichen Rechner befinden. Während die Daten bisher von oben nach unten weitergereicht wurden, erfolgt nun praktisch der Rückweg (bottom-up). Dazu wurde die Grafik (siehe Abbildung 12) durch eine gestrichelte Linie in die beiden Abschnitte „Dateneingabe“ und „Datenausgabe“ unterteilt. Dies wurde jedoch nur der Übersichtlichkeit halber vorgenommen. Es handelt sich weiterhin um die gleichen Netzwerkwege. Die angeforderten Daten werden nun von der Datenbank zurück an das Szenario geschickt. Da nun am Szenario alle angeforderten Parameter vollständig vorhanden sind[2], erzeugt das Szenario eine XML-Datei, die die grobe Struktur des Ausgabeformates enthält. Das konkrete Format wird anschließend entweder mittels XSLT erzeugt oder der XML-Code wird direkt weitergegeben. Sinnvolle Ausgaben sind etwa HTML-, PDF- oder Postscript-Dateien[3]. Das Servlet sendet das Rückgabedokument direkt an die Nutzerschicht weiter, in der die Datenausgabe erfolgen kann. Sollte sich die serverseitige Umwandlung von XML mittels XSLT-Techniken als zu langsam erweisen, kann der Transformationsprozess alternativ auch in die nächste Schicht verlagert werden. Dies würde bedeuten, dass die unbearbeiteten XML-Dateien weitergesendet werden, die darauf etwa durch Java Applets innerhalb der Nutzerschicht weiterverarbeitet werden können.

3.3 XML-Szenariobausteine

Wie aus Abbildung 13 ersichtlich, stehen die XML-Szenariobausteine im Mittelpunkt des Architekturkonzeptes von ***LibScens***. Ein Szenariobaustein definiert sozusagen den Ablaufplan bei der Abarbeitung eines Servlets. Deshalb widmet sich der nächste Abschnitt zunächst dem prinzipiellen Aufbau der XML-Szenariobausteine. Da eine modulare Unterteilung des Szenariobausteins vorgenommen werden kann, folgen daraufhin die Erläuterungen der einzelnen Module.

3.3.1 Modularer Aufbau der XML-Szenariobausteine

Wie in der Einleitung schon ausführlich erklärt wurde, stellt XML eine moderne Lösung zur strukturierten Speicherung von Daten dar. Deshalb bietet sich XML auch als Speicherungsform der Ablaufpläne für die Weiterverarbeitung der empfangenen Daten an. Der modulare Aufbau eines derartigen Ablaufplans wird in Abbildung 14 aufgezeigt.

Die Deskriptoren dienen dabei vor allem zur späteren Identifikation des Szenarios. Danach werden die Eingabetabellen gefüllt, deren Inhalt von unterschiedlichsten Quellen und Ebenen des Architekturkonzeptes stammen können. Sind einmal alle Eingabetabellen komplett erstellt, dann erfolgt die Transformation der Daten in komplexe Datenstrukturen. Nach der Transformation befinden sich die Daten in einem Format, welches sich zur Weiterverarbeitung durch XSL-Techniken eignet. Diese stellen strukturierte Objekte dar, wie sie z.B. aus objektorientierten Programmiersprachen bekannt sind. Den klassifizierten Datenstrukturen kann nun Funktionalität zugeordnet

[2]HTML-Parameter aus der Nutzerschicht, Datenbankanfrageergebnisse aus der Datenbankschicht und eventuell zusätzlich anfallende Parameter aus der Schnittstellenschicht. Derartige zusätzliche Parameter aus der Schnittstellenschicht sind im Falle des Prototyps beispielsweise die Authentifizierungsdaten, die in Form von XML-Dateien abgelegt werden.

[3]Der Prototyp wird standardmäßig HTML als Format ausgeben, da mittels HTML alle nötigen Interaktionsformen relativ einfach umgesetzt werden können.

werden. Der letzte Abschnitt ist die Zusammenstellung des kompletten Seitenlayouts eines Szenarios. Hier werden neben den strukturierten Objekten auch andere Designelemente angeordnet. Durch die Zuordnung von XSL-Templates kann die letzte nötige Transformation der vorläufigen XML-Datei in das endgültige Ausgabeformat erfolgen.

In den folgenden Abschnitten werden die einzelnen Teilmodule eines Szenariobausteins erheblich detaillierter dargestellt. Außerdem erfolgt eine Erläuterung anhand von praktischen Beispielen, so weit dies möglich und sinnvoll ist.

Abbildung 14: *Modularer Aufbau eines XML-Szenariobausteins*

3.3.2 Betrachtung der Zustandsspeicherung von Szenarios

Bei Web-Anwendungen, welche außer von der Angabe der URL noch von weiteren Eingabeparametern (Post- und/oder Get-Parameter) abhängig sind, kann die Speicherung des aktuellen Zustands dabei prinzipiell an zwei unterschiedlichen Stellen vorgenommen werden:

In der Nutzerschicht: Dabei werden die Werte der Parameter entweder in Cookies oder als versteckte Formularwerte im Dokument selber abgelegt. Dann werden die Formularwerte wieder *komplett* als Post- bzw. Get-Parameter an das Szenario übergeben. Dabei wird sozusagen der gesamte aktuelle Zustand vom Server zum Klienten übermittelt, dann erfolgt die Zustandsänderung auf der Klientenseite und letztendlich wird der gesamte geänderte Zustand an den Server zurückgeschickt.

Die Nachteile bei dieser Methode sind offensichtlich. Zum einen entsteht ein erhöhter Kommunikationsaufwand, da immer der komplette Zustand versendet werden muss, und zum anderen geht man ein erhöhtes Sicherheitsrisiko ein, da so auch sensible Informationen, wie beispielsweise Nutzerdaten, prinzipiell klientenseitig gespeichert bzw. temporär abgelegt werden könnten. Dies würde zwar in einem versteckten Formularfeld geschehen, wäre aber jedem versierten Anwender jederzeit durch Ansicht des Quelltextes der HTML-Seite zugänglich.

In der Schnittstellenschicht: Bei dieser Methode werden die Werte serverseitig auf der Schnittstellenschicht zugehörig zur Session gespeichert. Für einen neuen Zustand müsste deshalb von der Klientenseite nur eine *Zustandsänderung* in einem vordefinierten Format erfolgen.

Die Nachteile des vorher vorgestellten Verfahrens stellen gleichzeitig die Vorteile der serverseitigen Speicherung dar: Der Kommunikationsaufwand wird verringert und das Sicherheitsrisiko wird minimiert bzw. im Idealfall (HTTPS) praktisch eliminiert.

Aus diesem Grund ist die serverseitige Speicherung die bessere Wahl und wird in ***LibScens*** umgesetzt. Dies bedeutet, dass die Zustandswerte nicht in einer Tabelle gespeichert werden dürfen, die nur temporär gültig ist, da dann sämtliche Werte beim erneuten Aufruf des Szenarios verloren gingen. Die Werte werden deshalb in so genannten Zustandstabellen innerhalb einer Session abgelegt. Außerdem ist eine Zweiteilung der Zustandstabellen sinnvoll:

- Einerseits sollen die Zustandswerte nur solange gültig sein, wie der Nutzer auf einem Szenario arbeitet. Sobald das Szenario gewechselt wird, sollen alle zugehörigen Zustandswerte gelöscht bzw. in ihren Ausgangszustand zurückgesetzt werden. Die zugehörige Tabelle trägt in Abbildung 15 den Namen *„Zustandstabelle für Szenario“*.

  Beispiel: Man stelle sich vor, es gäbe ein Szenario, bei dem eine Baumstruktur dargestellt würde. Diese Baumstruktur soll zusätzlich aufklappbar sein. Dafür müssen die momentan aufgeklappten Knoten in einer Zustandstabelle abgelegt werden. Sie würden dann beispielsweise in diese Kategorie fallen.

- Andererseits gibt es Zustandswerte, die eine gesamte Session überdauern sollen und nicht vom momentanen Szenario abhängig sind. Diese Tabelle wird in Abbildung 15 mit dem Namen *„Zustandstabelle für Session“* aufgeführt.

  Beispiel: Die Speicherung des Namens oder der Autentifizierungsdaten eines Nutzers sollen natürlich die gesamte Session überdauern und sind deshalb zu dieser Kategorie zu zählen.

Da nur die Zustandsänderung übertragen wird, müssen spezielle Methoden hinzugefügt werden, welche die Art der Zustandsänderung beschreiben. Ein sinnvoller Satz an Änderungsfunktionen ist der folgende:

- `add (Wert1, Wert2, ..)` - Hinzufügen von Werten an das Ende der Spalte einer spezifizierten Eingabetabelle.
- `remove (Wert1, Wert2, ..)` - Löschen der Werte aus einer spezifizierten Eingabetabelle.
- `addOrRemove (Wert1, Wert2, ..)` - Prüft, ob die Werte in der spezifizierten Spalte einer Tabelle existieren. Wenn ja, werden die Werte gelöscht, anderenfalls werden die Werte hinzugefügt.
- `pop (Ganzzahl n)` - Löschen der letzten n Werte der Spalte einer spezifizierten Eingabetabelle.

3.3.3 Konzept der Basisdatentypen, Hilfsfunktionen & -prozeduren

Die Hilfsfunktionen von ***LibScens*** basieren auf Basisdatentypen. Der Grundsatz an Basisdatentypen für ***LibScens*** besteht aus folgenden Typen:

- `LSString`[4]: dient als Container für Text,
- `LSInteger`: stellt den Behälter für Ganzzahlen dar,
- `LSBoolean`: enthält die Wahrheitswerte `true` oder `false`,
- `LSStringArray`: enthält eine Liste vom Datentyp `LSString`,
- `LSIntegerArray`: enthält eine Liste vom Datentyp `LSInteger`.

Es drängt sich die Frage auf, wie mit den Binary Large Objects verfahren werden soll. Eine Lösung dafür ist, welche beispielsweise auch im IBM Content Manager favorisiert wird, dass die binären Datenstrukturen (CLOBs[5] und BLOBs) nur *referenziert* werden und nicht die komplette Übertragung[6] an ***LibScens*** vorgenommen wird.

Ein XML-Dokument besteht neben anderen Strukturen hauptsächlich aus Elementen und Attributen. Das Konzept der Hilfsfunktionen bzw. Hilfsprozeduren basiert darauf, dass XML-Elemente entweder als Basisdatentyp interpretiert werden oder als Funktionen, welche als Funktionsergebnis wiederum einen Basisdatentyp zurückliefern, oder als Hilfsprozeduren, die ein leeres Element zurückgeben und hauptsächlich zur Befüllung oder Modifikation der Eingabetabellen dienen.

Mit anderen Worten: Es muss neben den Basisdatentypen auch Hilfsfunktionen geben, um beispielsweise die Datenbefüllung und Datenüberführung, also den Datenfluss, zu steuern. Hilfsfunktionen dienen unter anderem zur Lösung folgender Fragen:

[4]Zur besseren Trennung der Datentypen von Programmiersprachen und den Datentypen von ***LibScens***, werden allen Basisdatentypen die Buchstaben „`LS`“ vorangestellt.

[5]CLOB: Character Large Object

[6]Theoretisch wäre eine komplette Übertragung an ***LibScens*** jedoch ebenso möglich, da in den Eingabetabellen sämtliche Datentypen abgelegt werden könnten. Praktisch würden jedoch schon bei relativ geringen Datenmengen Performance-Engpässe auftreten. Zudem gibt es für viele Streaming-Datentypen spezielle Anwendungen, welche den Umgang mit den Datentypen wesentlich effizienter ohne zusätzlichen Implementierungsaufwand handhaben können. Beispielsweise gibt es für Videodaten spezielle Videoserver (bei IBM DB2 z.B. in Form von DB2 Extendern).

- *Wie wird eine Tabelle gefüllt bzw. woher bekommt sie ihre Daten?*
- *Auf welchem Weg werden die komplexen Strukturen mit den Daten der Tabellen befüllt?*

Hilfsfunktionen können also durchaus mit dem bekannten Konzept der „Methoden“ verglichen werden. Sie besitzen eine bestimmte Menge unterschiedlichster Eingabedatentypen und haben als Rückgabewert einen bzw. keinen Datentyp.

Alle Hilfsfunktionen aufzulisten, würde an dieser Stelle zu weit führen. Sie können in der Dokumentation der beiliegenden CD-ROM eingesehen werden. Um die unterschiedlichen Arbeitsweisen der Hilfsfunktionen zu demonstrieren, wird die Auslegung eines XML-Dokumentes als DOM-Baum[7] zu Rate gezogen. Jedes wohlgeformte XML-Dokument kann als Baumstruktur interpretiert werden. Mittels DOM kann eine Navigation durch die Dokumentenstruktur und kontextabhängige Zugriffe erfolgen. Außerdem ist eine Manipulation der Struktur möglich [KLE02].

Diese Strukturmanipulation wird für die rekursive Abarbeitung der Hilfsfunktionen von ***LibScens*** benötigt. Stellt man sich die Repräsentation eines XML-Dokumentes als Baumstruktur vor, dann gibt es genau zwei signifikante Arten der Modifikation des Baumes:

1. **Entfernung von Knoten**: Hierbei werden die Kindknoten der Wurzel rekursiv beginnend von den Blättern nacheinander abgearbeitet und in Basisdatentypen aufgelöst. Die Abarbeitung erfolgt also Bottom-Up. Sind alle Kindknoten als Grunddatentyp vorhanden, dann wird der zugehörige Vaterknoten berechnet und somit auch durch einen Basisdatentyp ersetzt. Im Anhang B.1 wird ein Beispiel zur Abarbeitung der Bottom-Up-Methode beschrieben.

2. **Hinzufügen von Knoten**: Die zweite Möglichkeit ist, dass zu einem Vaterknoten weitere Kindknoten hinzugefügt werden können. Dabei könnte zum Beispiel schon ein Kind eines bestimmten Typs existieren, das dann nur noch n-Mal geklont werden muss. Dies muss jedoch schon bei der Abarbeitung des Vaterknotens geschehen, *ohne* dass die zu klonenden Kinderknoten bereits bearbeitet wurden! In diesem Fall erfolgt die Abarbeitung Top-Down. Der Vater ist bei der Top-Down-Verarbeitung dafür verantwortlich, wie und in welcher Reihenfolge seine Kindknoten abgearbeitet werden. Im Anhang B.2 wird ein Beispiel zur Abarbeitung der Top-Down-Methode beschrieben.

3.3.4 Details der Deskriptoren

Der Deskriptor identifiziert ein Szenario. Dies erfolgt in der XML-Datei mittels der Elemente `<ScenarioName>` und `<ScenarioVersion>`. Außerdem erfolgt eine Bindung des Szenarios (Element `<URLBinding(s)>`) an eine oder mehrere unterschiedliche URL-Adressen, über welche durch einen Internetbrowser auf das Szenario zugegriffen werden kann.

Eine weitere Funktionalität ist die Definition der Datenbankkonnektoren (Element `<DBConnector(s)>`). Hier können unterschiedliche Verbindungen angegeben werden, die in der nächsten Phase zum Zugriff auf die Datenbanken benötigt werden. Prinzipiell handelt es sich hierbei um eine Schnittstelle zu unterschiedlichen Datenbanksystemen. Über die Datenbank-URL, den Nutzernamen und Passwort kann eine derartige Verbindung aufgebaut werden. Die konkrete Umsetzung der Konnektoren für den Prototyp kann unter Kapitel 4.5.1 nachgelesen werden.

[7] Die Abkürzung DOM steht für Document Object Model.

3.3.5 Details der Eingabetabellen

Die Definition der Eingabetabellen repräsentiert die Basisbefüllung der Szenariobausteine durch unterschiedliche Eingabeparameter. Dabei werden die Daten in das standardisierte Format der Tabellen überführt, wie aus relationalen Datenbanksystemen bekannt. Die programmiertechnische Abarbeitung zur Laufzeit sollte so konzipiert werden, dass später angelegte Tabellen auf Inhalte vorher befüllter Tabellen mittels spezieller Funktionen zugreifen können, um beispielsweise bestimmte Ergebnismengen einzugrenzen. So können auch verschiedene einfache Datenbankanfragen verknüpft werden, um komplexe Anfragen zu simulieren (z.B. geschachtelte Anfragen oder Views). Das ist besonders bei dem Einsatz von Datenbanksystemen hilfreich, die eine derartige Funktionalität standardmäßig nicht besitzen (z.B. MySQL Version 4.0).

Herkunft der Befüllungsdaten

Betrachtet man die drei unterschiedlichen Schichten der Abbildung 13 auf Seite 35, dann sind Eingabedaten aus jeder dort aufgelisteten Schicht möglich.

Beginnend bei der Nutzerschicht werden auf diese Weise zwei unterschiedliche Tabellen befüllt. Zum einen erfolgt der Import der Get-Attribute, die an die eigentliche Internetadresse mittels „`?`“ angehängt und durch „`&`“ getrennt werden. Jedes Szenario hat so mindestens ein Get-Attribut, welches das Szenario („`scenario=scenName`“) bestimmt. Alternativ könnten beispielsweise noch zwei Attribute sinnvoll sein, die eine Eingrenzung der anzuzeigenden Ergebnismenge definieren. Eine derartige beispielhafte URL könnte folgendes Aussehen besitzen „http://www.enotehistory.de/index?scenario=navigation&start=20&amount=5“ und dabei folgende Bedeutung haben: *„Rufe das Szenario mit dem Namen „navigation“ auf und zeige mir beginnend von Element 20 die nächsten 5 Ergebnisse an.“*

Neben den Get-Attributen kommen auch die Post-Attribute aus der Nutzerschicht. Diese werden nur versendet, wenn die `submit`-Funktion von Formularfeldern in HTML ausgelöst wird. Post-Attribute besitzen viele Vorteile gegenüber Get-Attributen. So sind sie einerseits nicht unmittelbar vom Nutzer einsehbar, falls dies gewünscht wird, und können außerdem Datenmengen beliebiger Größe transportieren. Bei Get-Attributen ist die Länge hingegen auf 255 Zeichen beschränkt. Die Attributnamen der Post-Attribute werden durch die Beschriftung der Formularfelder vorgenommen.

Desweiteren sind Attribute aus der Schnittstellenschicht denkbar. So könnte etwa ein Import von XML-Dateien, die auf dem Server abgelegt wurden, in ***LibScens*** erfolgen. Hierzu zählen alle Dateien, die auf dem Rechner abgelegt sind, auf dem auch die ***LibScens***-Laufzeitumgebung installiert wurde. Im Prototyp werden beispielsweise die Authentifizierungsdaten in Form eines externen XML-Dokumentes gespeichert.

Die Speicherung von Zuständen ist ebenfalls in dieser Schicht anzusiedeln. Über die Betrachtung der Szenarios als Zustandsautomaten wurde bereits im Kapitel 3.3.2 referiert. Es gibt nun zwei mögliche Arten von Zustandsvariablen: Zum einen diejenigen, die nur über die Verweildauer auf einem bestimmten Szenario gültig sind, und zum anderen Variablen, welche zur Session zugehörig sind und deshalb auch beim Betreten anderer Szenarios ihre Gültigkeit nicht verlieren. Der Szenarioadminstrator muss deshalb im Baustein Flags zur Einordnung in eine der beiden genannten Gruppen setzen können.

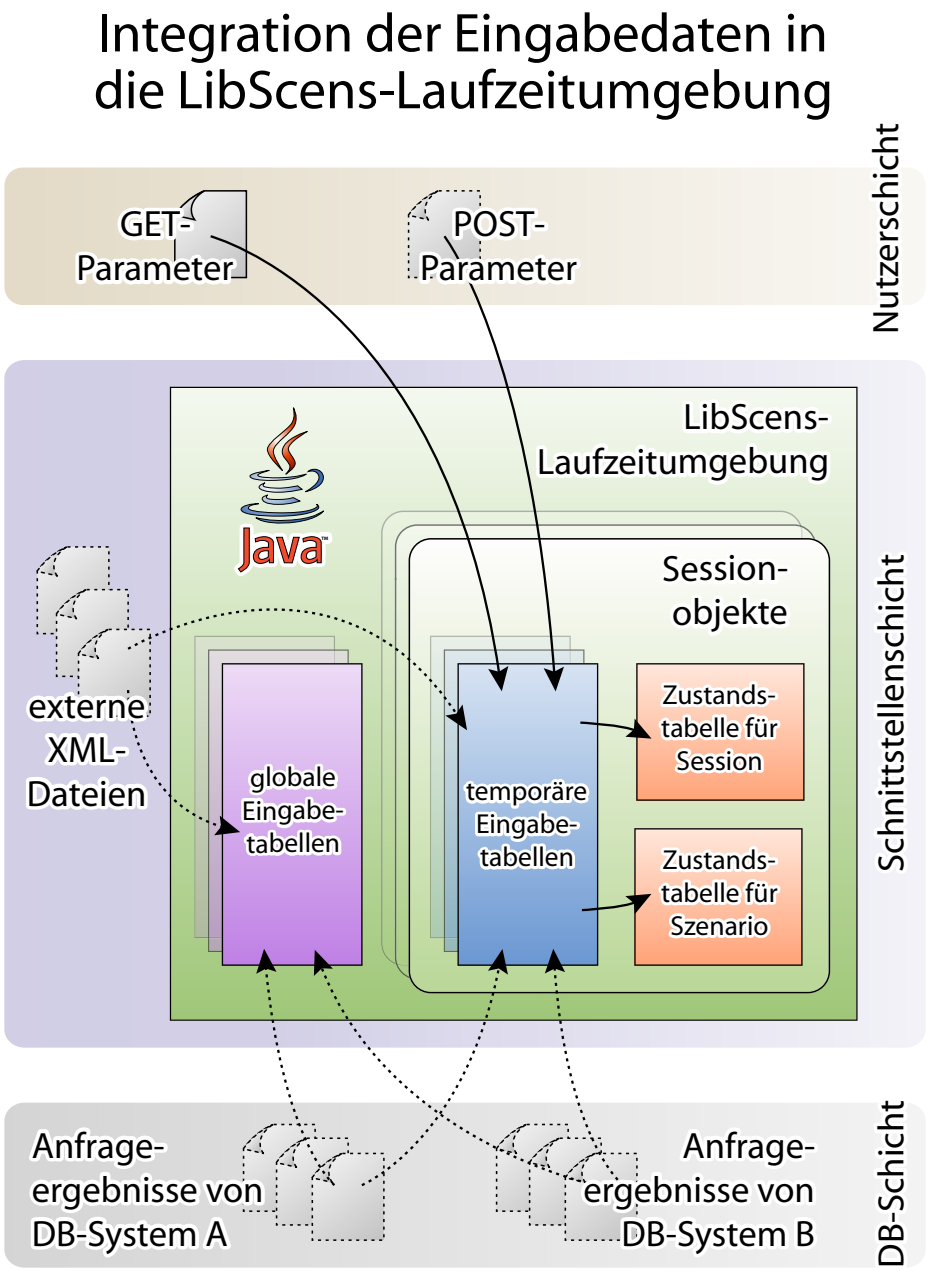

Abbildung 15: *Integration der Eingabedaten in die **LibScens**-Laufzeitumgebung*

Die wohl am häufigsten genutzten Eingabewerte kommen aus der Datenbankschicht und werden explizit durch Datenbankanfragen realisiert. Die Schnittstelle zur Datenbank wurde dafür so konzipiert, dass unterschiedlichste Datenbanksysteme anbindbar sind. Die Anfragen nutzen dabei jeweils eine der im Deskriptorenteil definierten Verbindungen. Dafür werden spezielle Hilfsprozeduren zur Verfügung gestellt, welche unter Angabe des Datenbanktyps, des zugehörigen Konnektors, des Datenbankanfrage-Strings, des Namens der zu befüllenden Eingabetabelle und der Caching-Methode eine Tabelle befüllt. Möglichkeiten zur Umsetzung werden im Kapitel 4.5.5 genauer beschrieben.

Klassifikation der Tabellen

Um bestimmte Performance-Engpässe von vornherein zu vermeiden, musste die Möglichkeit geschaffen werden, Eingabetabellen nach der Dauer ihrer Gültigkeit zu klassifizieren. Wie in Abbildung 15 deutlich wird, gibt es dafür drei unterschiedliche Arten, wie bei ***LibScens*** zur Laufzeit Eingabetabellen gecached[8] bzw. zwischengespeichert werden:

1. Globale Eingabetabellen,
2. Temporäre Eingabetabellen,
3. Zustandstabellen.

Die Zuordnung der Eingabetabellen wird dabei im Szenariobaustein festgelegt. Während globale Eingabetabellen eben aufgrund ihrer Allgemeingültigkeit im globalen Speicherobjekt abgelegt werden können, müssen die beiden anderen Tabellensorten direkt in einem Caching-Objekt gespeichert werden, das einer bestimmten Session-ID zugehörig ist.

Die einfachste Form stellen dabei die *Globalen Eingabetabellen* dar. Hierbei handelt es sich um komplette Eingabetabellen, welche vom aktuellen Zustand des momentanen Szenarios völlig unabhängig befüllt werden können und aufgrund der geringen Größe auch komplett in den reservierten Hauptspeicherbereich passen. Globale Eingabetabellen werden im Idealfall nur *einmal* während der Laufzeit befüllt. Sie werden lediglich bei Änderung der Umgebungsvariablen neu initialisiert, z.B. falls der XML-Szenariobaustein während der Laufzeit geändert wird.

Temporäre Eingabetabellen sind meistens Snapshots[9] von Relationen. Tabellen werden als temporär deklariert, wenn sie beispielsweise aufgrund ihrer gesamten Größe nicht in den Hauptspeicherbereich hineinpassen. Temporäre Tabellen sind deshalb, wie in Abbildung 15 gezeigt, immer an ein bestimmtes Sessionobjekt gebunden. Denn der Auszug einer Relation kann selbstverständlich von Nutzer zu Nutzer differieren. Temporäre Tabellen werden nach *jedem* Seitenaufruf gelöscht.

Das Konzept der Zustandsautomaten wurde bereits in Kapitel 3.3.2 vorgestellt. Die entsprechenden Zustandstabellen werden ebenfalls pro Sessionobjekt verwaltet. Der Unterschied zwischen den beiden Zustandstabellen ist folgender: Die *„Zustandstabelle für die Session“* wird einmal erstellt, kann dann modifiziert werden und wird erst bei Beendigung der Session gelöscht. Die *„Zustandstabelle für das Szenario“* hingegen wird dann gelöscht, wenn der Nutzer zu einem anderen Szenario wechselt.

Verwendung einer Zielanfragesprache

Da die Befüllung der Eingabetabellen auch gleichzeitig die einzige Schnittstelle zu einem Datenbanksystem darstellt, ist eine Zielanfragesprache unerlässlich. Prinzipiell sind zwei verschiedene Arten von Anfragesprachen sinnvoll:

Einerseits kann die *Originalanfragesprache* des unterliegenden Datenbanksystems genutzt werden. Dies wäre beispielsweise bei der Anbindung an IBM DB2 die Anfragesprache SQL-99. Der Vorteil dieser Methode ist, dass alle Besonderheiten des speziellen Datenbanksystems unterstützt

[8] Als Cache wird in diesem Fall ein ausgewiesener Puffer im RAM verstanden, welcher kleiner und zugleich schneller als der Plattenspeicher ist.

[9] Unter Snapshots versteht man einen Auszug aus einer Relation. Dafür müssen die Ober- und Untergrenze dieses Relationenauszugs angegeben werden.

werden können, z.B. die Stored Procedures oder User Defined Functions bei DB2. Der Nachteil dieser Methode ist, dass der Anwender zum einen gute Kenntnisse über die spezifische Anfragesprache mitbringen und zum anderen eventuell mehrere unterschiedliche Anfragesprachen parallel beherrschen muss, falls divergierende Datenbanksysteme angebunden werden.

Andererseits könnte auch eine *einheitliche Anfragesprache* für ***LibScens*** definiert werden, welche durch einen Wrapper den Code auf die exakte Anfragesprache des Datenbanksystems abbildet. Hierbei wäre natürlich ein erheblicher zusätzlicher Arbeitsaufwand für die Implementierung des Wrappers notwendig.

Im Falle des Prototyps wird deshalb die erste Variante bevorzugt. Sollte während der Evaluation mit mehreren Datenbanksystemen die zweite Variante als sinnvoller erscheinen, kann sie ohne Probleme jederzeit in ***LibScens*** eingebunden werden, da die dafür erforderliche Schnittstelle vorhanden ist.

3.3.6 Details der komplexen Strukturen

Die im vorherigen Abschnitt vorgestellten Eingabetabellen dienen zur Zwischenspeicherung der Daten in einem einheitlichen Speicherungsformat. Dabei sind jedoch die logisch zu einem Konstrukt zusammenhängenden Daten oftmals in unterschiedlichen Tabellen abgelegt. Außerdem sind natürlich Tabellendaten mitunter nicht so strukturiert, wie es der Kontext erfordert. Das so gespeicherte Format ist demnach für die XSL-Transformation gänzlich ungeeignet. Es muss deshalb in ein Format überführt werden, in dem die logisch zusammengehörenden Daten auch physisch in einem komplexen Objekt abgelegt sind. Nur so kann eine effektive XSL-Transformation erfolgen.

Es muss sich die Frage stellen: *„Warum werden die Eingabedaten nicht gleich in die komplexen Strukturen überführt und wieso wird der „Umweg“ über die Eingabetabellen genutzt?“* Der Hauptgrund hierfür liegt in der Tatsache begründet, dass die Konzepte von relationalen, objektrelationalen und objektorientierten Datenbanksystemen sehr stark divergieren und deshalb gewissermaßen auf ein Level normalisiert oder heruntergebrochen werden müssen. Die unterste Schicht der XML-Szenariobausteine sind deshalb die Tabellen, da auch die derzeit meisten praktisch genutzten Systeme auf Relationen, also Tabellen, basieren. So sind objektorientierte Anfrageergebnisse zwar leicht auf Tabellen abzubilden, jedoch relationale Anfrageergebnisse (beispielsweise mittels SQL) nicht unmittelbar in komplexe Objekte überführbar.

Der Versuch relationale Anfrageergebnisse auf Objekte abzubilden kann im schlechtesten Fall dazu führen, dass für jedes Grundelement eines Objektes eine separate Datenbankanfrage für den zugehörigen Attributwert aus der Relation gestellt werden muss. Dies wiederum würde zu einem nicht tolerierbaren Overhead an Informationen und damit verbundenen zeitlichen Mehraufwand führen, da hierfür mitunter mehrere tausend Anfragen zur Befüllung einer einzigen komplexen Struktur benötigt würden.

Aus diesem Grund werden alle Informationen erst einmal intern in den Eingabetabellen abgelegt und von dort aus mittels verschiedener Zuordnungsmechanismen in die komplexen Strukturen überführt, denn der Zugriff auf intern abgelegte Attributwerte gestaltet sich um ein Vielfaches schneller als adäquate Datenbankanfragen.

Nichtsdestotrotz können für objektorientierte Datenbanksysteme ohne Anpassung der Laufzeitumgebung zusätzliche Hilfsfunktionen implementiert werden, die den Schritt über die internen Eingabetabellen vermeiden und die Objekte *direkt* in die komplexen Strukturen überführen.

Genereller Aufbau komplexer Strukturen

Um die im nächsten Abschnitt beschriebene Klassifikation von komplexen Strukturen vornehmen zu können, müssen sie einem einheitlichen Aufbau genügen. Jede komplexe Struktur besteht aus Strukturobjekten. Ein Strukturobjekt ist entweder ein Strukturelement oder ein Struktur-Array.

Strukturelemente repräsentieren die kleinsten Elemente einer komplexen Struktur. Sie sind atomar und nicht weiter unterteilbar. Ihr Inhalt kann z.B. Text, ein Bild oder ein Eingabefeld sein. Aus strukturtechnischer Sicht stellen sie die Blätter eines Baumes dar. Neben dem eigentlichen Inhalt kann ihnen zusätzlich ein bestimmter Elementtyp, die Repräsentationsform und Funktionalität zugeordnet werden:

- *Elementtyp*: Jedem Strukturelement muss ein Elementtyp zugewiesen werden. Der Elementtyp definiert somit den Wertebereich des Inhalts. Einerseits kann so noch einmal vor der eigentlichen Darstellung überprüft werden, ob der Inhalt konform zu dem angegebenen Datentyp ist[10], andererseits besitzen die verschiedenen Elementtypen auch unterschiedliche Repräsentationsformen, die ihnen zugeordnet werden können. Der minimale Satz an Elementtypen ist:
  - `Text`: Enthält ein Textfeld beliebiger Länge und kann zusätzliche Stilinformationen für die Interpretation als HTML-Dokument besitzen.
  - `Number`: Enthält nur Ganzzahl- oder Fließkommawerte[11].
  - `Boolean`: Enthält nur Wahrheitswerte.

  Aber auch eine feingranularere Trennung macht Sinn, z.B. eine genauere Spezifizierung von `Number` in weitere Untertypen wie `Integer`, `Float` oder noch spezieller `Percent`. Je weiter der Bereich eingegrenzt werden kann, desto spezieller können die zugehörigen Repräsentationsformen gewählt werden[12].
- *Repräsentation*: Für die unterschiedlichen Elementtypen von Strukturelementen gibt es jeweils ein eigenes Set an Repräsentationsformen. So kann `Text` als Standardtext interpretiert werden, als Verweis auf ein Bild, als Überschrift oder etwa als Vorgabetext für ein Eingabefeld. Die Repräsentationsformen der Strukturelemente sind jedoch wesentlich spartanischer als die unterschiedlichen Interpretationsmöglichkeiten der gesamten komplexen Objekte.
- *Funktionalität*: Da für ***LibScens*** eine Funktionalität nur für Internetseiten zweckmäßig ist[13], beschränkt sich diese dabei sinnvollerweise auf das Setzen von Links zu anderen Szenarios. Dafür werden zusätzlich die Parameter angegeben, die übermittelt werden sollen, und die Art der Übermittlung wird definiert (Post oder Get). Für die korresponierende Zustandsänderung ist dann das aufgerufene Szenario verantwortlich.

[10] Beispiel: Angenommen der Elementtyp ist vom Typ `Number`, der eigentliche Inhalt ist jedoch ein `Text`. Die beiden Datentypen sind nicht vereinbar und es wird eine Fehlermeldung ausgegeben. Diese Überprüfung wird vorgenommen, um für die XSL-Bearbeitung einen *garantierten* Datentyp zur Verfügung zu stellen, der eine entsprechend angepasste Bearbeitung erlaubt.

[11] Zahlen müssen beispielsweise bei der XSL-Umwandlung für die Reihenfolge der Sortierung anders behandelt werden als reiner Text. Sie sind deshalb für eine exakte Bearbeitung zu einem anderen Elementtyp zu zählen.

[12] Für den stark eingegrenzten Bereich `Percent` von `Number` sind z.B. folgende spezielle Repräsentationsformen denkbar: Interpretation als Text, als Balkengrafik, als Farbverlauf oder als reiner Farbwert.

[13] Bei alternativen, statischen Ausgabeformaten wie PDF oder reinem XML macht eine Zuordnung von Funktionalität zu Elementen natürlich keinen Sinn.

Neben den Strukturelementen gibt es als Strukturobjekte noch die **Struktur-Arrays**[14]. Sie dienen zur Gruppierung logisch zusammengehörender Informationen in einem Unterobjekt. Betrachtet man die komplexen Strukturen als Baum, dann sind die Stuktur-Arrays immer innere Knoten und können niemals Blätter darstellen. Jedem Struktur-Array wird bei seiner Definition eine Menge an Array-Elementen (`<ArrayElements>`) zugeordnet, über die iteriert wird. Dabei wird für jedes Array-Element ein Klon des Array-Körpers (`<ArrayBody>`) erstellt, in dem alle Referenzen auf das aktuelle Array-Element durch dessen konkreten Inhalt ersetzt werden. Ein Beispiel für die Abarbeitung eines Struktur-Arrays wird im Anhang B.2 genauer erläutert. Das Konzept der Struktur-Arrays ist damit vergleichbar mit Schleifen aus Programmiersprachen. Es ist jedoch im Gegensatz zu herkömmlichen Schleifen nicht nur eine Iteration über Ganzzahlwerten möglich, sondern es ist außerdem eine Iteration über Textwerten denkbar[15]. Ist eine dynamische Iteration nicht erwünscht, dann können weitere Strukturobjekte auch direkt als Kindelemente an das `<StructArray>` angefügt werden[16]. Struktur-Arrays können außerdem geschachtelt eingesetzt werden.

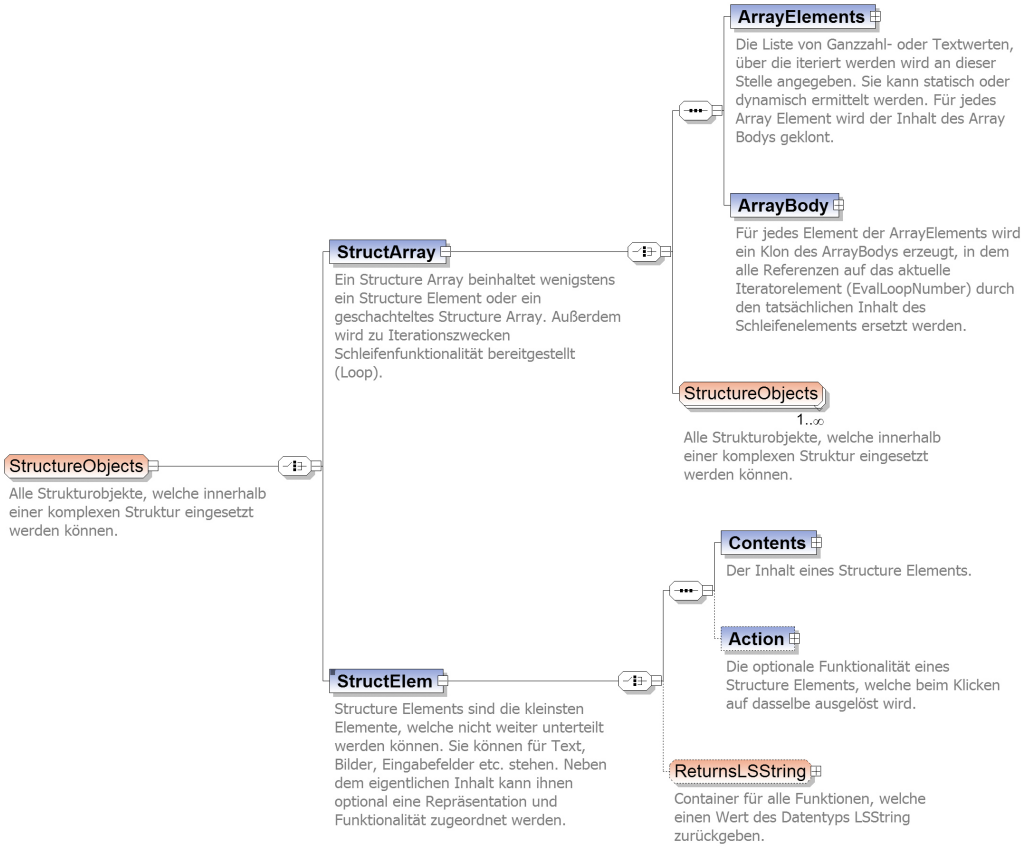

Abbildung 16: *Auszug der Strukturobjekte aus dem XML-Schema*

[14] Der Begriff `Array` wurde gewählt, da die im `ArrayBody` angegebenen Strukturobjekte *im Zusammenhang gesehen* wieder ein Objekt darstellen. Somit bleibt die Eigenschaft von Arrays erhalten, dass ein Array *nur* Objekte gleichen Typs beinhaltet.

[15] Bei Textwerten würden dann alle Referenzen auf das Array-Element durch Text ersetzt werden. Hierbei könnte beispielsweise der Text auch eindeutige Pfadangaben beinhalten, die später für weitere XPath-Anfragen genutzt werden könnten, etc. Die Möglichkeiten sind gut erweiterbar.

[16] Dies ist vor allem für zusätzliche Darstellungsattribute von komplexen Strukturen sinnvoll, welche *direkt* angegeben werden können und nicht dynamisch über einen Iterator bestimmt werden müssen.

Zusammenfassend bestehen also die komplexen Strukturen aus einer bestimmten Kombination von Strukturelementen und Struktur-Arrays. Das entsprechende XML-Schema für den Aufbau der Strukturobjekte zeigt Abbildung 16 auf.

Klassifikation komplexer Strukturen

Nachdem der grundlegende Aufbau von komplexen Strukturen im vorhergehenden Abschnitt dargelegt wurde, kann nun eine Klassifikation der komplexen Strukturen erfolgen. Ebenso wie eine Klassifikation der Strukturelemente im Kleinen notwendig ist, ist sie noch viel bedeutsamer für die gesamte komplexe Struktur. Nur durch Klassifikationen kann eine Schnittstelle geschaffen werden, auf welche im späteren Verlauf XSL-Templates aufsetzen können, um sinnvolle Darstellungen der komplexen Strukturen zu erzeugen.

Die Klassifikation von komplexen Strukturen gibt dabei den Aufbau der Struktur und die Namen der einzelnen Strukturobjekte vor. Der Aufbau kann dabei in drei Komponentenarten unterschieden werden:

- *Obligatorische Komponenten*: Diese Komponentenanteile *müssen* bereitgestellt werden. Sie beschreiben den geringsten Satz an Strukturkomponenten, die benötigt werden, damit eine Interpretation überhaupt erfolgen kann.
- *Optionale Komponenten*: Diese Bestandteile können begleitend angegeben werden und erweitern so die Funktionalität oder das Aussehen der komplexen Strukturen. Die Standard-XSL-Templates werden optionale Komponenten erkennen und diese bei der Darstellung berücksichtigen.
- *Zusätzliche Komponenten*: Diese Komponenten unterscheiden sich von den optionalen Bestandteilen insofern, dass sie nicht von den Standard-XSL-Templates erkannt oder berücksichtigt werden. Die XSL-Templates können deshalb von den Nutzern so erweitert werden, dass eine Integration der zusätzlichen Komponenten möglich ist. Dies ist notwendig, um eine flexible Schnittstelle zu schaffen, die auch auf spezielle Bedürfnisse angepasst reagiert.

Bei der Untersuchung von digitalen Bibliotheken (siehe Kapitel 2.3) haben sich zwei Klassen von komplexen Strukturen als essenziell herausgestellt: Tabellen[17] und Bäume. Deshalb erfolgt die Demonstration der Klassifikation anhand dieser beiden Konstrukte.

Aufbau der komplexen Struktur „Tabelle“

Werden die Listen zur Klasse der Tabellen hinzugerechnet, dann kommt keine digitale Bibliothek ohne adäquate Tabellenkonstrukte aus. Tabellen werden hauptsächlich zur Präsentation von beliebigen Ergebnismengen genutzt. Oftmals dienen sie auch als Ausgangspunkt für eine flache Navigation über Daten. Dazu werden Tabellen mit zusätzlichen Navigationselementen ausgestattet, die eine Verschiebung des Wertebereichs des aktuellen Snapshots erlaubt.

[17] Wobei Listen zur Klasse der Tabellen gezählt werden, da sie im Prinzip Tabellen mit lediglich einer Spalte beschreiben. Listen und Tabellen dürfen dabei *nicht* mit den Konstrukten aus dem Datenbankbereich verwechselt werden! Dort haben beispielsweise nur Listen eine Ordnung und Tabellen nicht und dürften deshalb nicht zu einer Klasse gezählt werden.

```
<!-- Interface fuer die Definition von Tabellen in LibScens -->
<Structure Type="LSTable">

  <!-- Zelleninhalt der einzelnen Spalten -->
  <StructArray ArrayName="TableColumns">
    <StructElem ElemName="Column1">
      <Contents><LSString/></Contents>
    </StructElem>
    ..
    <StructElem ElemName="ColumnX">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>

  <!-- Ueberschriften der einzelnen Spalten -->
  <StructArray ArrayName="TableHeaders">
    <StructElem ElemName="Column1">
      <Contents><LSString/></Contents>
    </StructElem>
    ..
    <StructElem ElemName="ColumnX">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>

  <!-- Wertebereich der Tabelle -->
  <StructElem ElemName="StartRow">
    <Contents><LSInteger/></Contents>
  </StructElem>
  <StructElem ElemName="EndRow">
    <Contents><LSInteger/></Contents>
  </StructElem>

  <!-- Spalte, nach der sortiert wird -->
  <StructElem ElemName="SortColumn">
    <Contents><LSString/></Contents>
  </StructElem>

  <!-- Redundante Zelleninhalte anzeigen? -->
  <StructElem ElemName="ShowRedundantCells">
    <Contents><LSBoolean/></Contents>
  </StructElem>

  <!-- Kopfzeile alle x Zeilen wiederholen -->
  <StructElem ElemName="RepeatHeaders">
    <Contents><LSInteger/></Contents>
  </StructElem>
</Structure>
```

Abbildung 17: *XML-Interface für die komplexe Struktur „LSTable“*

Bezogen auf die komplexen Strukturen benötigt man für Tabellen folgende Eingabeparameter:

- Der eigentliche Tabelleninhalt muss angegeben werden. Dafür muss für jede Spalte und jede Zeile angegeben werden, woher der Wert bezogen wird:

  Element: `<StructArray ArrayName="TableColumns">` (obligatorisch)

- Desweiteren können die Überschriften in der Kopfzeile festgelegt werden:

  Element: `<StructArray ArrayName="TableHeaders">` (optional)

- Es können Werte festgelegt werden, die den Wertebereich der Tabelle bestimmen:

  Element: `<StructElem ElemName="StartRow">` (optional)

  Element: `<StructElem ElemName="EndRow">` (optional)

- Die Tabelle kann über eine bestimmte Spalte sortiert werden:

  Element: `<StructElem ElemName="SortColumn">` (optional)

- Es können redundante Zelleninhalte angezeigt bzw. verborgen werden:

  Element: `<StructElem ElemName="ShowRedundantCells">` (optional)

- Über eine Nummer kann angegeben werden, nach wievielen Zeilen die Kopfzeile wiederholt wird[18]:

  Element: `<StructElem ElemName="RepeatHeaders">` (optional)

Die XML-Schnittstelle für die komplexe Struktur „Tabelle“ wird in Abbildung 17 verdeutlicht. Diese Schnittstelle zeigt jedoch ausschließlich diejenigen Elemente an, auf welche das XSL-Stylesheet reagiert. Es handelt sich dabei also lediglich um Muster, wie die Datenstrukturen im Szenariobaustein aussehen müssen. Die Befüllung der Elemente wird zur Übersichtlichkeit vernachlässigt.

Aufbau der komplexen Struktur „Baum“

Nach Tabellen nehmen Bäume einen zentralen Platz beim Umgang mit digitalen Bibliotheken ein. Sie werden häufig eingesetzt, um hierarchisch geschachtelte Inhalte auch adäquat grafisch zu repräsentieren[19].

Für ***LibScens*** werden ausschließlich *azyklische, gerichtete Bäume* betrachtet. Aus wissenschaftlicher Sicht ist ein gerichteter Baum ein gerichteter Graph mit folgenden Eigenschaften:

1. Es gibt genau einen Anfangsknoten, genannt Wurzel.
2. Jeder Knoten (außer der Wurzel) hat genau einen Vorgänger.
3. Jeder Knoten (außer den Blättern) hat mindestens einen Nachfolger.

Die Knotenmenge von Bäumen kann theoretisch auf unterschiedliche Arten abgelegt werden. Für ***LibScens*** wird jedoch die Speicherungsform von Knoten in Tabellenform bevorzugt. Mögliche Eingabeparameter für Tabellen sind:

- Die Knotenmenge wird als Liste abgelegt:

  Element: `<StructArray ArrayName="Nodes">` (obligatorisch)

- Pro Knoten muss ein Knotenidentifikator angegeben werden:

  Element: `<StructElem ElemName="NodeID">` (obligatorisch)

- Der eigentliche Knoteninhalt ist ebenfalls anzugeben:

  Element: `<StructElem ElemName="NodeContents">` (obligatorisch)

[18] Die Wiederholung der Kopfzeile nach einer gewissen Anzahl von Zeilen macht besonders bei großen Tabellen Sinn. Ein gutes Beispiel für die Verwendung dieses Features findet man unter: http://www.tecchannel.de/tecdaten/show.php3?catid=75&pageid=346.

[19] Ein gutes Beispiel für eine Baumstruktur stellen die Genre-Subgenre-Bäume dar, die z.B. bei http://www.amazon.de eingesetzt werden. Dort werden die Genres beginnend von der Wurzel „Bücher“ hierarchisch geschachtelt. Ein Beispielpfad wäre etwa „Bücher“ → „Fachbücher“ → „Informatik“ → „Praktische Informatik“ → „Datenbanken“.

- Desweiteren ist der Knotenidentifikator des Vaters notwendig[20]:

  Element: `<StructElem ElemName="ParentNodeID">` (obligatorisch)

- Ein Knotentyp kann angegeben werden:

  Element: `<StructElem ElemName="NodeType">` (optional)

- Sinnvoll ist ebenfalls die Angabe eines zugehörigen Bildes:

  Element: `<StructElem ElemName="NodeImage">` (optional)

- Die Liste der aufgeklappten Knoten kann für einige Darstellungsformen notwendig sein:

  Element: `<StructArray ArrayName="ExpandedNodes"` (optional)

- Der Pfad zu dem aktuellen Knoten kann gleichfalls bedeutsam sein:

  Element: `<StructArray ArrayName="CurrentNodes">` (optional)

- Handelt es sich um Warenkorbfunktionen, dann müssen zusätzlich die selektierten Knoten abgelegt werden:

  Element: `<StructArray ArrayName="SelectedNodes">` (optional)

Viele andere Attribute von Bäumen und Knoten im Speziellen sind denkbar. Sie können jederzeit als zusätzliche Komponenten angegeben werden, benötigen in diesem Fall jedoch angepasste XSL-Templates.

Wie bei den Tabellen im vorherigen Abschnitt kann man auch für Bäume eine XML-Schnittstelle angeben (siehe Abbildung 18)[21]:

```
<!-- Interface fuer die Definition von Baeumen in LibScens -->
<Structure Type="LSTree">

  <!-- Menge der Knoten als Liste -->
  <StructArray ArrayName="Nodes">

    <!-- Pro Knoten gibt es einen Knotenidentifikator -->
    <StructElem ElemName="NodeID">
      <Contents><LSString/></Contents>
    </StructElem>

    <!-- Der eigentliche Knoteninhalt -->
    <StructElem ElemName="NodeContents">
      <Contents><LSString/></Contents>
    </StructElem>

    <!-- Die Knoten-ID des Vaters -->
    <StructElem ElemName="ParentNodeID">
      <Contents><LSString/></Contents>
    </StructElem>

    <!-- Der Knotentyp ist optional -->
    <StructElem ElemName="NodeType">
      <Contents><LSString/></Contents>
    </StructElem>
```

[20] Ohne Angabe des Vaterknotens kann der gerichtete Graph nicht erstellt werden. Da es sich bei dem Graph um einen Baum handelt, ist außerdem nur ein Vaterknoten pro Knoten möglich.

[21] Hier gilt ebenfalls: Es handelt sich bei der Abbildung lediglich um Muster, wie die Datenstrukturen im Szenariobaustein aussehen müssen. Die Befüllung der Elemente wird zur Übersichtlichkeit vernachlässigt.

```
    <!-- Eine Bildreferenz pro Knoten -->
    <StructElem ElemName="NodeImage">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>

  <!-- Liste der aufgeklappten Knoten -->
  <StructArray ArrayName="ExpandedNodes">
    <StructElem ElemName="ExpandedNodeID">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>

  <!-- Pfad zum aktuellen Knoten -->
  <StructArray ArrayName="CurrentNodes">
    <StructElem ElemName="CurrentNodeID">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>

  <!-- Liste der selektierten Knoten -->
  <StructArray ArrayName="SelectedNodes">
    <StructElem ElemName="SelectedNodeID">
      <Contents><LSString/></Contents>
    </StructElem>
  </StructArray>
</Structure>
```

Abbildung 18: *XML-Interface für die komplexe Struktur „LSTree“*

Selbstverständlich sind die komplexen Strukturen von ***LibScens*** nicht auf „Baum“ und „Tabelle“ beschränkt. Die beiden vorhergegangenen Komponenten dienen vielmehr als Beispiel für die Erstellung von Schnittstellen für komplexe Strukturen.

3.3.7 Details zum Seitenlayout

Die Erstellung des Seitenlayouts manifestiert den letzten Schritt, der zur Erzeugung des Ausgabeformates erforderlich ist. Dies ist nur dadurch möglich, dass vor diesem Schritt alle nötigen Daten in einer strukturierten Form vorliegen, die nun durch XSL-Prozessoren bearbeitet werden können. Die Definition des Seitenlayouts kann dabei in zwei Schritte unterschieden werden:

1. Aufbauend auf die Definition von derartigen XML-Schnittstellen kann jeder komplexen Struktur nun eine **Repräsentationsform** zugewiesen werden.

2. Anhand von HTML-Code, weiteren Templates und der Einordnung der komplexen Strukturen in die Seite kann nun das **komplette Seitendesign** festgelegt werden.

Repräsentationsformen von komplexen Strukturen

Während die Interpretationsmöglichkeiten der Strukturelemente noch stark begrenzt sind, kommen für komplexe Strukturen eine große Vielfalt an Darstellungsmöglichkeiten in Frage. Dabei gilt es, verschiedene *sinnvolle* Repräsentationen der Strukturen zu ermöglichen.

Die Zuordnung der Repräsentation erfolgt in ***LibScens*** mittels des XML-Attributes `Representation`, das jeder klassifizierten komplexen Struktur in der Phase des Seitenlayouts zugeordenet werden kann. So ist beispielsweise folgender Ausdruck in diesem Abschnitt zulässig[22]:

[22] Dies setzt natürlich voraus, dass die komplexe Struktur im Abschnitt `<ComplexStructures>` mit Hilfe des Interfaces `LSTable` definiert wurde!

```
<Structure Name="Beispieltabelle" Representation="LSTableStandard">
```

Zur Umsetzung werden XSL-Templates eingesetzt. XSL[23] ist eine moderne Methode, um XML-Dateien ein bestimmtes Aussehen, ein Erscheinungsbild zu verleihen. Dabei wird das XML-Dokument auf bestimmte XPath-Ausdrücke hin untersucht und entsprechend ersetzt. Im Fall von ***LibScens*** wird dabei nach der Kombination einer komplexen Struktur einer bestimmten Klasse und deren Repräsentationsform gesucht. Für das Beispiel aus Abbildung 17 würde der entsprechende XPath-Ausdruck etwa nach folgenden Knoten suchen: `//Structure[@Representation="LSTableStandard"]`.

Die verschiedenen Templates für die unterschiedlichen Strukturklassen und zugehörigen Repräsentationsformen werden dabei in separaten Dateien abgelegt und mittels einer globalen Stylesheet-Datei zusammengeführt. Das hat den Vorteil, dass zum einen komplette Repräsentationsklassen problemlos ausgetauscht werden können und zum anderen auch neue Repräsentationen sehr einfach in die speziellen Klassen integrierbar sind.

Da im vorangegangenen Abschnitt der Aufbau der komplexen Strukturen „Tabelle“ und „Baum“ betrachtet wurden, werden diese Beispiele wieder aufgegriffen und entsprechende Repräsentationsformen zu diesen Konstrukten vorgestellt. Es handelt sich dabei jedoch nur um einen Auszug der Interpretationsmöglichkeiten, welche die am häufigsten genutzten Anwendungsfälle abdeckt. Sollten speziell angepasste Repräsentationsformen notwendig sein, kann jederzeit das zugehörige XSL-Template angeglichen werden.

Bibliothek	Signatur	Werktitel	Komponist
D-ROu	Musica Saec. XVIII.-9.^9.10	Trios	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^8	Cantata Core amante di perché libertá	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^7	Cantata Sequir fera che fugge	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^6	Ouverture A	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^5	Ouverture D	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^4	Ouverture g	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^3	Ouverture F	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^2	Chaconne A	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^14	Trio C	Giuseppe Antonio Brescianello
D-ROu	Musica Saec. XVIII.-9.^13	XII Concerti et sinphonie op. 1	Giuseppe Antonio Brescianello

Abbildung 19: *Standardrepräsentation einer Tabelle*

[23] XSL steht für eXtensible Stylesheet Language.

Repräsentationsformen für Tabellen

Genügt eine komplexe Struktur dem Interface der Klasse `LSTable` (Kapitel: *Aufbau der komplexen Struktur „Tabelle“* auf Seite 47), dann sind folgende zwei Interpretationsmöglichkeiten sinnvoll:

- Die standardmäßige Repräsenation der **Tabelle** (Abbildung 19). Dabei wird die Tabelle unverfälscht wiedergegeben. Zusätzliche Darstellungsattribute wie die Umrandung der Zellen oder die Wiederholung der Kopfzeile nach einer bestimmten Anzahl von Zeilen sind möglich. Die räumliche Trennung der unterschiedlichen Attribute von dokumentenähnlichen Objekten in verschiedene Spalten hat den Vorteil, dass sie zum einen direkt mit ihrem Vorgänger und Nachfolger vergleichbar sind und zum anderen auch eine Sortierung nach diesen Attributen vorgenommen werden kann. Dieses Feature würde lediglich eine entsprechende Funktionalität der Überschriften der einzelnen Spaltenkopfzeilen voraussetzen.

Abbildung 20: *Repräsentation einer Tabelle in Listenform*

- Jede Tabelle kann alternativ auch als **Liste** dargestellt werden (Abbildung 20). Diese Form findet in vielen digitalen Bibliotheken ihren Einsatz, da oftmals eine Darstellung der Attribute dokumentenähnlicher Objekte als Ganzes gewünscht wird und keine Trennung der Merkmale auf verschiedene Spalten erfolgen soll. Der unmittelbare Vergleich zwischen den Attributen geht bei dieser Repräsentationsform verloren. Vergleiche sind jedoch auch nur in gewissen Einsatzbereichen von digitalen Bibliotheken sinnvoll. Die Darstellungsform der Liste wird zudem häufig im Bereich von Internetseiten eingesetzt, da sich in diesem Fall eine vertikale Datenrepräsentation besser eignet als die horizontale Form von Tabellen[24].

[24] Dies hängt damit zusammen, dass der horizontale Platz für Internetseiten im Allgemeinen sehr begrenzt ist und nahezu jede Usability-Studie darauf drängt, dass horizontales Scrolling vermieden werden sollte. Dies ist beispielsweise auch ein Grund, warum Internetsuchmaschinen für das Anzeigen ihrer Ergebnisse Listen gegenüber Tabellen bevorzugen.

Repräsentationsformen für Bäume

Auch für die Baumrepräsentation muss die komplexe Struktur passend zur entsprechenden Schnittstelle `LSTree` (Kapitel: *Aufbau der komplexen Struktur „Baum“* auf Seite 49) sein. Für den Prototypen von ***LibScens*** wurden folgende Interpretationsformen implementiert:

- Jeder Baum kann wiederum als **Tabelle** interpretiert werden (Abbildung 21). Dafür wird die geschachtelte Baumstruktur in eine flache Struktur überführt. Dabei geht natürlich ein Großteil der Übersichtlichkeit und Lesbarkeit verloren. Die Funktionalität ist jedoch immer noch vorhanden. So kann ein Aufklappen des Knotens über das Klicken auf die Untergruppen erreicht werden. In digitalen Bibliotheken ist die Repräsenation als Tabellenstruktur immer dann wünschenswert, wenn der Vergleich über die einzelnen Knotenattribute im Vordergrund steht.
- Die häufigste Repräsentationsform des Baumes ist z.B. aus dem **Microsoft Explorer** bekannt (Abbildung 22). Sie ist typisch für die Darstellung von Verzeichnisstrukturen und stellt auch die typische Repräsentation für digitale Bibliotheken dar. Der Kontext wird hierbei viel deutlicher hervorgehoben als bei der flachen Tabellenstruktur. Die erwartete Funktionalität ist beispielsweise, dass der Unterbaum beim Klicken auf das Pluszeichen erweitert und beim Klicken auf das Minuszeichen versteckt wird.

Abbildung 21: *Repräsentation eines Baums als flache Tabelle*

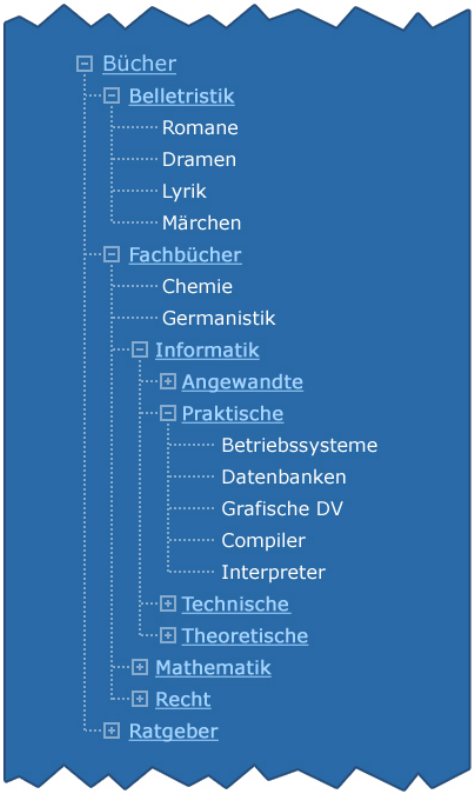

Abbildung 22: *Repräsentation eines Baums als Microsoft-Explorer-Struktur*

- Während bei den vorangegangenen Repräsentationen jeweils die Übersicht über einen größeren Ausschnitt oder die Gesamtheit der Baumknoten im Vordergrund stand, liegt der Fokus der dritten Interpretationsform auf dem **aktuellen Knoten** (Abbildung 23). Er steht deshalb im Zentrum der Darstellung. Über ihm werden seine Väter in hierarchischer Reihenfolge angezeigt, rechts sieht man seine Geschwister und unten werden die Kinder des momentanen Knotens in alphabetischer Reihenfolge angezeigt. Beim Klicken auf einen anderen Knoten wird dieser in den Mittelpunkt gerückt. Diese Form findet ihren Einsatz vor allem in den Bereichen, in dem ein Auswahlprozess stattfindet und lediglich ein kleiner Ausschnitt von Interesse ist, nämlich genau jener Bereich, der sich in direkter Nachbarschaft des aktuellen Knotens befindet.

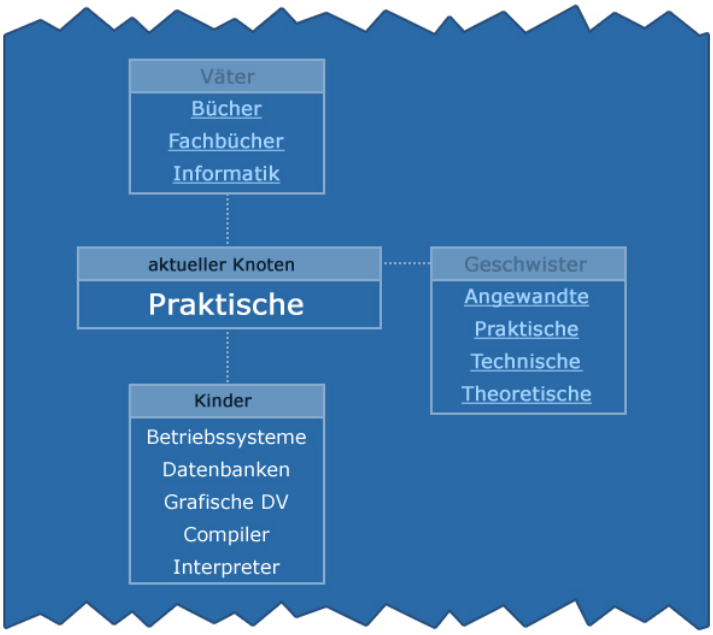

Abbildung 23: *Repräsentation eines Baums fokussiert auf den aktuellen Knoten*

Erstellung des kompletten Seitendesigns

Unter dem XML-Element `<PageLayout>` der XML-Szenariomodule wird im abschließenden Schritt des Erzeugungsprozesses das gültige Ausgabeformat erzeugt. Dieser Schritt ist also auch nur dann notwendig, wenn überhaupt ein Ausgabeformat gewünscht und nicht nur eine Rückgabe der XML-Datei notwendig ist[25].

In diesem XML-Abschnitt können nun alle Bestandteile angegeben werden, die zur Erzeugung eines gültigen Ausgabeformates erforderlich sind. Für die meisten digitalen Bibliotheken ist dies die Generierung einer HTML-Datei. Valide HTML-Dateien können dabei mit Hilfe der unterschiedlichsten Verfahren erzeugt werden, welche natürlich auch geschachtelt verwendet werden können:

[25] Beispielsweise kann die XML-Datei unbearbeitet an die Nutzerschicht weitergegeben werden, falls erst dort die Transformation in ein Ausgabeformat erwünscht ist, z.B. mittels Java Applets.

- Die einfachste Form ist die **direkte Eingabe von HTML-Code**. Diese Form der Eingabe ist vor allem für die Aufteilung der Seite nützlich. Dies kann beispielsweise durch die Gliederung in Tabellen (HTML-Tags: `<table>`, `<tr>` und `<td>`) erreicht werden. Auch andere kleinere, statische Anteile der HTML-Seite können auf diese Weise eingebunden werden.

- Für alle größeren statischen Abschnitte, die eventuell noch einen hohen Wiederverwendungsgrad[26] besitzen, sollten besser **statische Templates** eingesetzt werden. Die einfachste Form der Integration in ***LibScens*** ist die Verwendung von XSL-Templates, da XSL auch für die nächste Form der Eingabe genutzt wird. Diese Art der Templates wird in separaten, projektbezogenen Dateien gespeichert, denn sie wird von Anwendungsbereich zu Anwendungsbereich divergieren.

- Umfangreiche dynamische Anteile werden ebenfalls durch **dynamische XSL-Templates** generiert, benötigen jedoch einen ungemein größeren Erzeugungs- bzw. Verwaltungsaufwand als statische Templates, da sie auf die unterschiedlichsten Eingabewerte reagieren müssen. Zu dieser Art der Templates zählen auch die im Abschnitt 3.3.7 vorgestellten Repräsentationsformen für komplexe Strukturen. Diese Art der Templates besitzen den Vorteil, dass sie häufig projektübergreifend genutzt werden können.

Mit der Erstellung des kompletten Seitendesigns ist die Anfertigung eines XML-Szenariomoduls abgeschlossen und der Szenariobaustein kann nun problemlos dem Pool der anderen Szenariobausteine hinzugefügt werden. Dies ist sogar zur Laufzeit von ***LibScens*** möglich ohne den Webserver zu unterbrechen. So können die Szenariomodule jederzeit geändert, gelöscht oder hinzugefügt werden, ohne dass der Datenfluss für nicht betroffene Bausteine unterbrochen werden muss.

3.4 Integration der Nutzerauthentifizierung

Die Integratation einer fein differenzierbaren Nutzerauthentifizierung gewinnt derzeit, gerade im Bereich der Internetapplikationen, rapide an Bedeutung. Immer mehr Anbieter sind nicht zuletzt dank moderner Sicherheitsverbindungen wie HTTPS[27] gewillt, auch vertrauenswürdige Inhalte online verfügbar zu machen. Dabei reicht jedoch oftmals der Grad der Differenzierung zwischen verschiedenen Nutzern bzw. Gruppen nicht aus. In diesem Abschnitt wird deshalb ein vierschichtiges Authentifizierungskonzept für ***LibScens*** vorgestellt, welches das Erstellen von Sicherheitsbarrieren in vier hierarchisch geschachtelten Niveaustufen ermöglicht (siehe Abbildung 24).

Die oberste Stufe der Hierarchie stellt dabei die Gruppendefinition dar. Eine Gruppe besteht dabei aus einem Namen und Privilegien. Pro Gruppe können verschiedene Nutzer registriert werden. Diese erben zum einen die Eigenschaften der Gruppe, können zum anderen jedoch zusätzliche Privilegien besitzen. Die nächste Stufe bilden die Szenarioauthentifizierungen. Dabei wird je nach Ansatz der Zugriff auf zusätzliche Szenarios freigegeben bzw. verboten. Pro Szenario kann

[26] Hierzu zählen vor allem mehr oder weniger komplexe Designelemente wie Abfolgen von Bildern, welche in den unterschiedlichsten Szenariobausteinen zum Einsatz kommen. Sie sollten als Template angegeben werden und nicht direkt als HTML-Quellcode. Der Vorteil darin ist zum einen der verringerte Schreibaufwand und zum anderen der geringe Verwaltungsaufwand, denn sollte sich ein Grundbaustein ändern, dann muss nur das eine Template angepasst und nicht *jeder* XML-Szenariobaustein einzeln angeglichen werden.

[27] HTTPS: HTTPS - Hypertext Transport Protocol Secure

wiederum eine Restriktion der Attribute von Eingabetabellen auf bestimmte Wertebereiche definiert werden. Dabei gibt es statische und dynamische Attribute. So können statische Attribute beispielsweise zur Fallunterscheidung während der Szenarioabarbeitung genutzt werden. Es ließe sich zum Beispiel ein Attribut `allowHighQualityImages` festlegen, über welches entschieden werden kann, ob einem Nutzer der Zugriff auf qualitativ hochwertige Bilder eines Szenarios gestattet wird oder nicht. Während der Abarbeitung eines XML-Szenariobausteins kann dann auf die konkrete Belegung dieses Attributs reagiert werden. Dynamische Attribute hingegen können neben einzelnen Werten auch komplette Wertebereiche enthalten, die den Zugriff auf Tabellenattribute bestimmen. So könnte beispielsweise Gastnutzern eines Szenarios nur ein eingegrenzter Wertebereich aller möglichen Bibliotheksobjekte zugewiesen werden. In Bezug auf die Navigation durch bestimmte Buchmengen würde dies bedeuten, dass dem Gastnutzer z.B. nur die Navigation über die ersten 20 Bücher erlaubt wäre. Durch Definition von Attributrestriktionen kann so ein optimaler Grad an Nutzerdifferenzierung erfolgen.

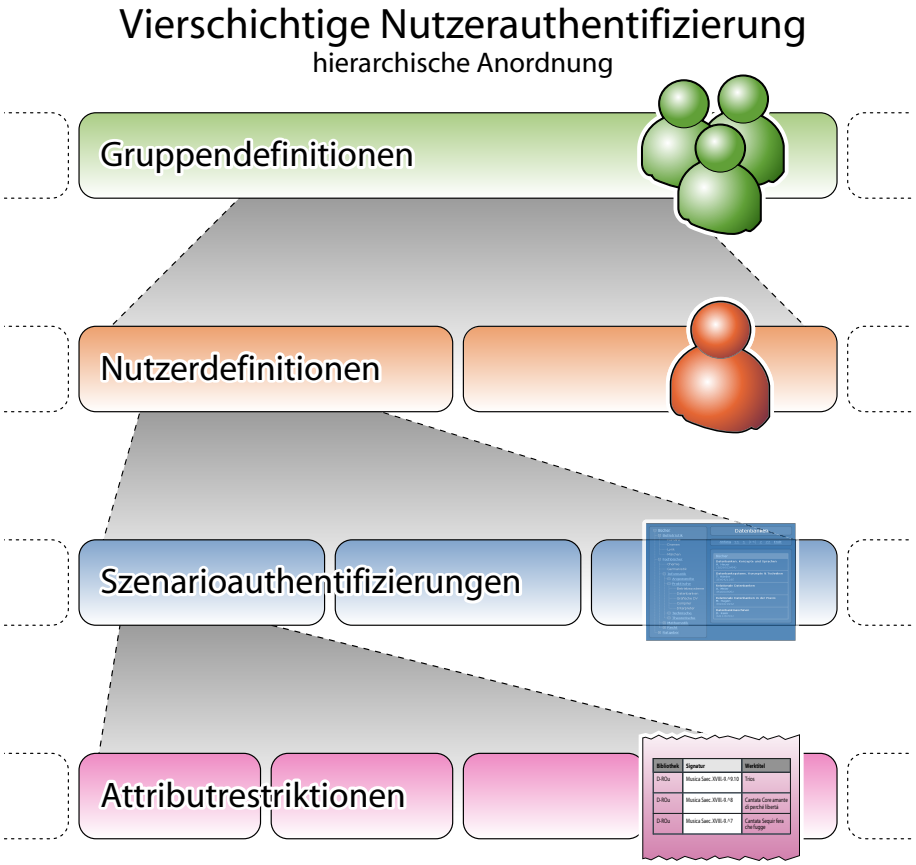

Abbildung 24: *Hierarchische Anordnung der vierschichtigen Nutzerauthentifizierung*

Generell gibt es für die Festlegung von Authentifizierungsprivilegien zwei unterschiedliche Ansätze:

1. Der *defensive* Ansatz verbietet von vornherein komplett den Zugriff auf alle Attribute. Sollen einer Gruppe oder einem Nutzer Privilegien gewährt werden, müssen diese zur Liste der freigegebenen Attribute hinzugefügt werden.

2. Der *offensive* Ansatz hingegen erlaubt anfänglich den Zugriff auf sämtliche Attribute. Dabei werden nur diejenigen Attribute in einer Liste gespeichert, auf welche der Zugriff nicht gestattet ist.

Beide Methoden haben ihre Berechtigung, jedoch ist der defensive Ansatz aufgrund des verminderten Sicherheitsrisikos der offensiven Strategie vorzuziehen.

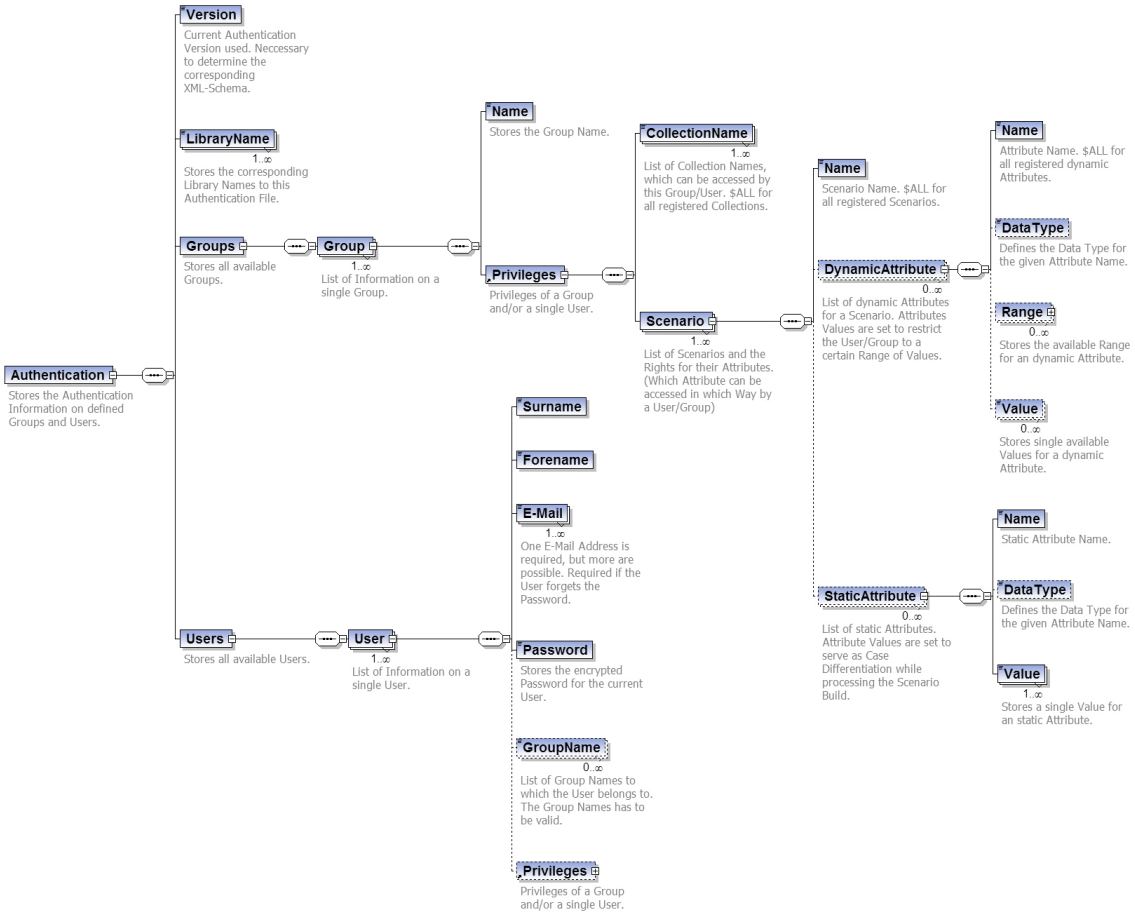

Abbildung 25: *XML-Schema einer vierschichtigen Nutzerauthentifizierung*

Als Ablageform der Nutzerauthentifizierung bietet sich, nicht zuletzt aus Konsistenzgründen zu den anderen Konzepten von ***LibScens***, die Speicherung als XML-Dateien an. Um die Erstellung dieser XML-Authentifizierungsdatei zu erleichtern und zu Validierungszwecken ist die Angabe einer DTD oder eines XML-Schemas unerlässlich. Abbildung 25 zeigt die adäquate Umsetzung des vierschichtigen Authentifizierungskonzepts mittels XML-Schema.

Kapitel 4

Implementierung

4.1 Techniken zur Umsetzung von *LibScens*

Neben der Erstellung des Konzeptes zählt die prototypische Implementierung zum Aufgabenbereich dieser Arbeit. Hierbei muss abseits der Funktionsvielfalt ein hoher Grad an Performance geboten werden. Die im folgenden Abschnitt vorgestellten Technologien wurden deshalb auf diese beiden Gesichtspunkte hin untersucht. Die Techniken können dabei je nach Anwendungsbereich in eine der drei Schichten eingeordnet werden:

- In das **Front-End** werden diejenigen Techniken eingeordnet, welche zur direkten Repräsentation des Ausgabeformats eines Szenariobausteins erforderlich sind.
- Dem **Interface** sind die Implementierungstechniken zuzuordnen, die zur Speicherung und Konvertierung des Szenariobausteins in das Ausgabeformat notwendig sind.
- Zum **Back-End** zählen alle Datenquellen, die zur Befüllung der Szenariobausteine genutzt werden.

4.1.1 Front-End

Für die konkrete Darstellung des Ausgabeformates wurde **HTML** gewählt. Wie im Konzept (Kapitel 3.2) beschrieben, sind zwar auch andere Ausgabeformate denkbar, jedoch stellt HTML derzeit die Standardrepräsentation für digitale Bibliotheken dar. Zu den Gründen ist zu zählen, dass HTML einerseits auf praktisch jedem modernen Rechner mittels eines Internet-Browsers angezeigt werden kann und andererseits ein hoher Interaktionsgrad mit der digitalen Bibliothek gewährleistet ist. Weitere statische Ausgabeformate wie PDF[1] oder Postscript-Dateien sind jedoch ohne weiteres implementierbar.

Die standardmäßigen Interaktions- und Darstellungsmöglichkeiten von HTML werden durch **JavaScript** erweitert. Die dynamischen und interaktiven Funktionen von JavaScript werden eingesetzt, um beispielsweise auf bestimmte Ereignisse zu reagieren. Derartige Ereignisse sind z.B. MouseOver-Events. Durch das Abfangen und Reagieren auf MouseOver-Events können so ver-

[1] PDF: Portable Document Format

schiedene Zustände von Buttons hervorgerufen[2] oder Popup-Fenster als Hilfe angezeigt werden[3]. Die JavaScript-Anteile werden bei ***LibScens*** unmittelbar in die XSL-Templates integriert.

Eine weitere Erweiterung von HTML sind **Cascading Stylesheets (CSS)**. Sie erweitern im Gegensatz zu JavaScript jedoch nicht den Funktionsumfang, sondern die grafischen Repräsentationsmöglichkeiten von HTML. Ihr großer Vorteil liegt in der Definition von globalen Stylesheets[4]. So können über eine einzige globale CSS-Datei sämtliche Darstellungsattribute der digitalen Bibliothek festgelegt und geändert werden. Nahezu jede moderne Internetseite und deshalb auch alle untersuchten digitalen Bibliotheken nutzen CSS zur einheitlichen Gestaltung ihrer Onlinepräsenz. Für ***LibScens*** gibt es zum einen eine globale CSS-Datei und zum anderen werden CSS-Informationen direkt innerhalb der XSL-Templates angegeben.

4.1.2 Interface

LibScens wurde komplett in **Java** programmiert. Der größte Vorteil von Java ist dessen Systemunabhängigkeit. Dadurch wird eine problemlose Anwendung unter verschiedenen Betriebssystemen gewährleistet. Ein weiterer Vorteil von Java ist die kostenlose Nutzung, solange es sich nicht um ein kommerzielles Produkt handelt. Die konkrete Klassenstruktur von ***LibScens*** wird dabei in Kapitel 4.3 genauer erläutert. Als JDK[5] wurde die Version 1.4.2 genutzt, ein Umstieg auf das neue JDK 1.5 ist mit geringem Aufwand jederzeit möglich. Für die Web-Anbindung werden Java-Servlets genutzt. Diese empfangen einerseits die Eingabeparameter von der Nutzerschnittstelle und geben andererseits die endgültige HTML-Datei an den Nutzer bzw. Browser zurück.

Der Prototyp wird standardmäßig auf dem Webserver **Apache Tomcat** in der Version 4.1. betrieben. Da es derzeit eine große Auswahl an verschiedenen Webservern gibt, waren hier die freie Verfügbarkeit, die unkomplizierte Konfiguration und nicht zuletzt die gute Dokumentation die Kriterien, weshalb Tomcat bevorzugt wurde. Auch in diesem Bereich ist ein problemloser Wechsel auf Alternativen ohne weiteres möglich.

Wie im Kapitel 3 bereits beschrieben wurde, dient **XML** zur Speicherung der Szenariobausteine. Desweiteren können auch andere XML-Dokumente, wie Authentifizierungsdaten, jederzeit in ***LibScens*** importiert werden. XML wurde gewählt, weil damit eine eine gut strukturierte Speicherung der Daten realisierbar ist und es inzwischen eine Vielzahl von überzeugenden Werkzeugen gibt, welche die Erstellung, Verarbeitung und Modifikation von XML-Dateien ermöglichen.

Ein weiterer Vorteil von XML ist die Möglichkeit der Validierung der XML-Dokumente mittels **XML-Schema**. Dies ist außerordentlich sinnvoll, um syntaktische Fehler bei der Erstellung der XML-Szenariobausteine oder Authentifizierungsdateien von vornherein auszuschließen. Desweiteren können die XML-Dateien zum Laufzeitbetrieb überprüft werden und ungültige Dateien, die Fehler innerhalb der Laufzeitumgebung hervorrufen würden, damit ignoriert werden. Da mit Hilfe von XML die Daten sehr feingranular strukturiert abgelegt und mittels XML-Schema grundlegende Strukturen definiert und erkannt werden, unterstützt XML-Schema auf diese Weise eine spätere Erweiterung von ***LibScens*** um grafische Schnittstellen.

[2] Die unterschiedlichen Zustände von Buttons werden im Allgemeinen durch unterschiedliche Bilder repräsentiert. Beispielsweise sind verschiedene Bilder für folgende Zustände denkbar: MouseOver, MouseUp, MouseClick.

[3] Popup-Fenster stellen eine moderne Lösung dar, um dem Nutzer schnell und intuitiv Hilfe anzubieten. Sie reagieren damit unmittelbar auf die Verhaltensweise von Nutzern, die in ungewohnten oder unschlüssigen Situationen dazu neigen, den Mauszeiger nicht zu bewegen.

[4] Stylesheets trennen die Form der Darstellung von den Inhalten eines strukturierten Dokuments.

[5] JDK: Java Developer Kit

Da jedes valide XML-Dokument auch als Baumstruktur interpretiert werden kann, ist eine Konvertierung der Datei in die entsprechende DOM-Struktur[6] notwendig. Die beiden führenden XML-Parser dieses Bereichs sind Apache Xerces und das Open-Source-Projekt DOM4J. Da, wie in Kapitel 3.3.3 genauer beschrieben, das Auslesen und die Modifikation der Baumstruktur notwendig sind, muss das zu verwendende Werkzeug in der Lage sein, **XPath**-Ausdrücke schnell und effizient auszuwerten. Wie im Kapitel 5 dargelegt, hat die DOM4J-Bibliothek in diesem Bereich wesentliche Performance-Vorteile. Außerdem weist die XPath-API[7] von Apache Xerces in der vorliegenden Version 2.6 erhebliche Schwachstellen im Umgang mit XPath auf (siehe [BOE04]). Deshalb wird in dieser Arbeit **DOM4J** bevorzugt.

Für die Konvertierung der XML-Dokumente in das Ausgabeformat werden desweiteren **XSL-Stylesheets** und ein adäquater XSLT-Prozessor benötigt. Die vorherrschenden Prozessoren sind hierfür Apache Xalan und SAXON. Wie bei der Evaluation des Prototyps von ***LibScens*** festgestellt wurde (siehe Kapitel 5), ist die Bearbeitung mittels SAXON erheblich performanter. Deshalb wird standardmäßig **SAXON** eingesetzt. Da mit dem JAXP-Interface[8] eine Schnittstelle für die Bearbeitung von XML-Dateien in Java geschaffen wurde, kann jedoch auch an dieser Stelle ein Austausch des XSLT-Prozessors vorgenommen werden[9].

4.1.3 Back-End

Da der Prototyp von ***LibScens*** aus dem eNoteHistory-Musikarchiv hervorgegangen ist, welches seine Daten derzeit in **IBM DB2** ablegt, wurde auf die Implementierung einer Schnittstelle zu DB2 besonderen Wert gelegt. Die Beispiele der XML-Szenariobausteine nutzen deshalb ebenfalls Konnektoren zu DB2. Jedes weitere Datenbanksystem erfordert eine angepasste Implementierung einer entsprechenden Hilfsprozedur wie `<FillTableDB2SQL>` (siehe auch Kapitel 4.5.1).

4.2 Dreistufiger Transformationsprozess

Betrachtet man die Implementierung aus Sicht der verschiedenen abgekapselten Umformungsschritte, dann ergibt sich ein dreistufiger Transformationsprozess. Abbildung 26 verdeutlicht die drei Schritte.

Begonnen wird mit dem Überführen der Ergebnisse der Anfrage aus der Quelldatenbank in die Eingabetabellen von ***LibScens***. Dies geschieht aufgrund der Beschreibung des Elements `<InputTables>` des zugehörigen XML-Szenariobausteins. Als zusätzliche, optionale Eingabeparameter kommen für diesen Transformationsschritt die Post- und Get-Attribute der HTML-Anfrage in Betracht. Als technische Hilfsmittel für die Umsetzung sind einerseits die Zielanfragesprache zu nennen, welche im Fall des Prototypen SQL ist, und andererseits wird die DOM-Struktur des XML-Szenariobausteins mittels DOM4J abgearbeitet.

[6] DOM: Document Object Model
[7] API: Application Programming Interface
[8] JAXP: Java API for XML Processing
[9] Der Austausch zwischen verschiedenen XSLT-Prozessoren gestaltet sich als äußerst einfach. Dazu muss lediglich das zu verwendende XSLT-Prozessoren-Package der Laufzeitumgebung ausgetauscht werden.

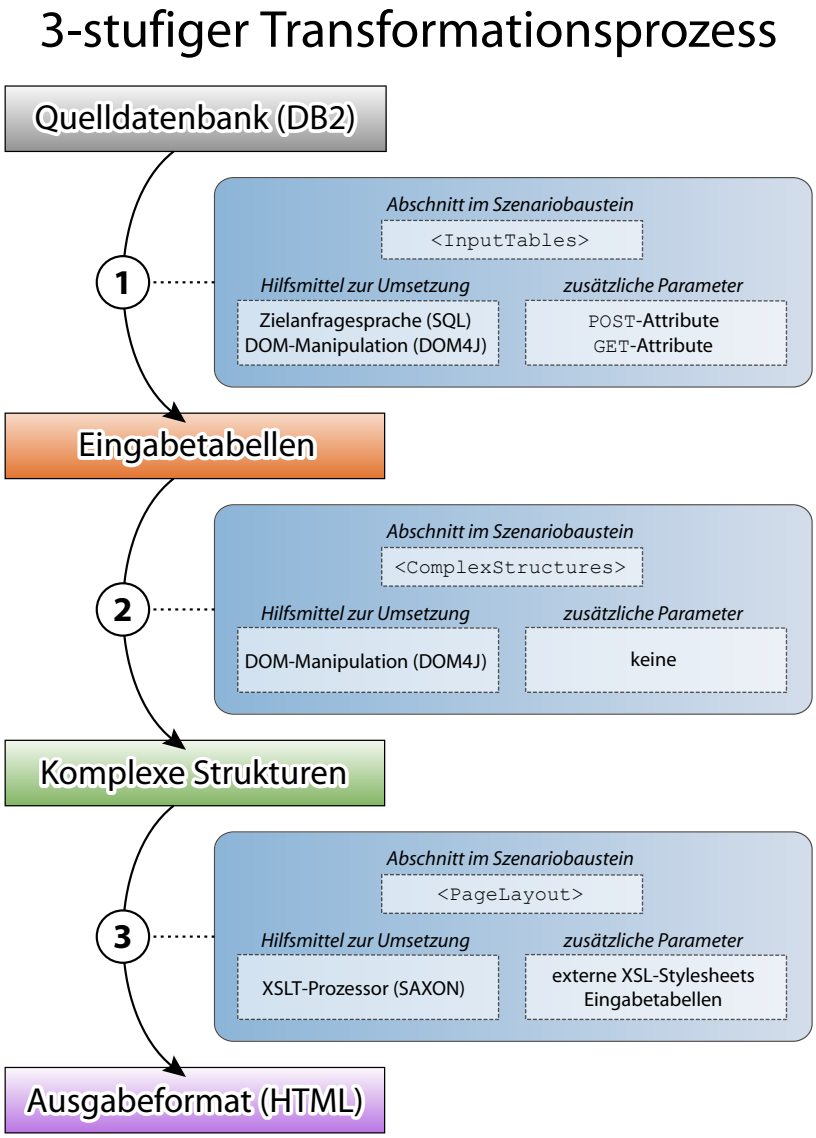

Abbildung 26: *Dreistufiger Transformationsprozess*

Danach werden die Eingabetabellen durch die Beschreibung des Abschnitts `<ComplexStructures>` in komplexe Strukturen konvertiert. Hierfür sind außer den Eingabetabellen an sich keine weiteren zusätzlichen Parameter erforderlich. Mittels DOM-Manipulation (DOM4J) und der im Kapitel 3.3.3 beschriebenen Techniken „Entfernung von Knoten“ und „Hinzufügen von Knoten“ werden die Beschreibungen der komplexen Strukturen mit dem konkreten Inhalt der Eingabetabellen befüllt.

Anhand der im Szenariobaustein unter dem Element `<PageLayout>` abgelegten Beschreibung wird das aktuelle Ausgabeformat (HTML) erzeugt. Als zusätzliche Parameter müssen hier zum einen die externen XSL-Stylesheet-Dateien angegeben werden und zum anderen können optional noch Werte aus den Eingabetabellen für die Konstruktion der HTML-Datei genutzt werden. Umgesetzt wird dies mittels des XSLT-Prozessors SAXON. Mit der Erstellung des Ausgabeformates sind alle Transformationsschritte abgeschlossen und das Servlet versendet den HTML-Quellcode an den Klienten zurück.

4.3 Aufbau der Java-Klassenstruktur

Bevor ein detaillierter Programmablaufplan vorgestellt wird, müssen die verwendeten Klassen von ***LibScens*** und deren Bedeutung aufgelistet werden. Der folgende Abschnitt fasst die Klassen des Prototyps zusammen. Dabei werden sie in der logischen Reihenfolge der Packages[10] aufgezählt.

Package DE.LibScens

Das Package `DE.LibScens` beinhaltet die Hauptkomponenten von ***LibScens***. Diese stellen einerseits die unmittelbare Schnittstelle zwischen dem Nutzer und der Laufzeitumgebung her und definieren andererseits den konkreten Ablaufplan für die Abarbeitung der XML-Szenariobausteine. Abbildung 27 listet die Klassen auf.

Klassenname	kurze Inhaltsbeschreibung
`MainServlet`	Von diesem Servlet werden alle HTML-Anfragen angenommen und HTML als Ausgabeformat zurückgeschickt. Damit repräsentiert diese Klasse die Schnittstelle zwischen Nutzer und ***LibScens***.
`ScenarioProcessor`	Jede unterschiedliche HTML-Anfrage (sessionabhängig) [11] erzeugt eine neue Instanz eines `ScenarioProcessors`. Dieser beinhaltet den konkreten Ablaufplan für einen Szenarioaufruf und bearbeitet die einzelnen Teilschritte sequenziell, wie in Kapitel 4.4 beschrieben. Dafür werden u.a. Hilfsmethoden aus der Klasse `Routines` eingebunden. Pro Session existiert maximal eine Instanz.
`GlobalCache`	Die „globale Speicherstruktur“ speichert sessionunabhängige, klassenübergreifende, persistente Objekte wie Initialdaten, XML-Szenariodateien und globale Eingabetabellen bzw. komplexe Strukturen, die zwischengespeichert werden können. Es handelt sich hierbei um ein Singleton-Objekt[12].
`SessionCache`	Alle Objekte (Eingabetabellen, komplexe Strukturen), die abhängig von der Session gecached werden müssen, werden innerhalb dieser Instanz gespeichert. Pro Session existiert genau eine Instanz, welche auch nach Absterben des aktuell zugehörigen `ScenarioProcessors` aktiv bleibt.
`Routines`	Diese Klasse enthält unterschiedliche Hilfsroutinen, die u.a. von den einzelnen Abschnitten des `ScenarioProcessors` genutzt werden. Dazu zählt beispielsweise das Parsen von XML-Dateien, das Abarbeiten von DOM-Knoten oder das Überprüfen bestimmter Parameter.

Abbildung 27: *Package DE.LibScens*

[10] Um große Projekte strukturieren zu können und saubere Schnittstellen zu Bibliotheken zu schaffen, können die Klassen zu Paketen (englisch *Packages*) zusammengefasst werden.

[11] Durch mehrere Nutzer können dementsprechend viele HTML-Anfragen parallel eintreffen und müssen dann entweder multi-threaded oder sequenziell bearbeitet werden.

[12] Singleton-Objekte sind Objekte, welche nur eine Instanz haben können. Jeder weitere Versuch einer erneuten Instanziierung referenziert die erste und einzige Instanz.

Package: DE.LibScens.AssistanceClasses

Die Klassen dieses Packages dienen als Container bzw. Hülle für Objekte, die vom Package `DE.LibScens` eingesetzt werden. Darunter befinden sich beispielsweise die Behälter, in denen die Eingabetabellen gespeichert werden (Klassen: `TableInput`, `TableColumn` und `TableCell`) oder auch simple Objekte, welche zu Logging-Zwecken (Klasse: `LoggedTime`) verwendet werden. Da es sich nur um Unterstützungsfunktionen handelt, wird dieses Package nicht ausführlich aufgelistet.

Package: DE.LibScens.HelpFunctions

Das Konzept der Hilfsfunktionen wurde bereits in Kapitel 3.3.3 ausführlich vorgestellt. In diesem Package werden alle zugehörigen Hilfsfunktionen bzw. Hilfsprozeduren abgelegt. Die Subpackages sind dabei nach den Rückgabewerten der Hilfsfunktionen benannt worden: `LSBoolean`, `LSInteger`, `LSIntegerArray`, `LSString`, `LSStringArray` und `Procedures`. Die für den Prototyp implementierten Hilfsfunktionen werden in der beiliegenden CD-ROM detailliert vorgestellt.

4.4 Ablaufplan eines Szenarioaufrufs

Die im Kapitel 4.2 vorgestellten Transformationsschritte können als Abstraktion des Ablaufplans angesehen werden. Ein wesentlich detaillierterer Programmablaufplan, der auch bereits die konkreten Methodenaufrufe beinhaltet, wird in Abbildung 28 aufgezeigt. Wie aus der Abbildung ersichtlich wird, können die groben Transformationsschritte in kleinere Module aufgesplittet werden, welche seriell abzuarbeiten sind. Dadurch besteht die Möglichkeit, bestimmte Module zu einem späteren Zeitpunkt zu implementieren, auszutauschen oder anzupassen[13].

Begonnen wird mit dem Empfang der Szenarioanfrage durch die Methode `doGet(..)` des Servlets `MainServlet.java` (Schritt 1). Danach wird überprüft, ob sich die INI-Datei[14] geändert hat und diese bei Bedarf neu zwischengespeichert (Schritt 2). Haben sich die XML-Szenariobausteine (Schritt 3) oder die XML-Authentifizierungsdatei (Schritt 4) seit dem letzten Aufruf geändert, dann müssen sie erneut in den Speicher geladen werden. Die Methode `updateSessionCache()` überprüft dann, ob zu der aktuellen Anfrage bereits ein Session-Objekt im globalen Cache existiert (Schritt 5). Existiert eine aktive Session, dann wird sie fortgesetzt, ansonsten wird ein neues Objekt dafür erzeugt. In Schritt 6 wird der Szenarioname aus der Anfrage extrahiert, um zu überprüfen, ob ein entsprechender XML-Szenariobaustein gecached wurde. Falls ja wird fortgefahren, falls nein muss abgebrochen und eine Fehlermeldung an den Nutzer geschickt werden. Darauf folgt das Entfernen aller temporären bzw. aller nicht mehr gültigen Eingabetabellen[15] (Schritt 7). In Schritt 8 und 9 werden die Get- bzw. Post-Parameter in die

[13] Beispielsweise wird im Prototypen im Programmschritt 4. — *„Cachen der XML-Authentifizierungsdatei“* — derzeit eine XML-Datei eingelesen und es werden zwecks Caching-Optimierung die entsprechenden Nutzerinformationen in die betreffenden Objekte geparst. Stattdessen ist auch eine Implementierung dieser Methode möglich, welche die Authentifizierungsdaten aus einer DB2-Datenbank liest oder eine Verbindung zu einem LDAP-Server herstellt.

[14] Die Initialisierungsdatei `GlobalSettings.ini` befindet sich standardmäßig im Unterordner der Tomcat-Webdateien und beinhaltet Logbucheinstellungen, Pfade zu der Authentifizierungsdatei, Pfade und Debuginformationen zu den XML-Szenariodateien und Einstellungen für den XSL-Transformationsprozess.

[15] Tabellen und komplexe Strukturen, welche als cacheable deklariert wurden, sind ungültig, sobald die INI-Datei, eine XML-Szenariodatei oder die Authentifizierungsdaten geändert wurden.

entsprechenden temporären Eingabetabellen überführt. Dann wird mittels der Datenbankkonnektoren eine Verbindung zu den genutzten Datenbanken hergestellt (Schritt 10), sofern diese nicht bereits im `GlobalCache` existiert. Mit Hilfe der Konnektoren und der Beschreibung des Elements `<InputTables>` der zwischengespeicherten XML-Szenariodatei kann nun die Befüllung der Eingabetabellen erfolgen (Schritt 11). Ist im späteren Verlauf der Einsatz von XPath auf diese Eingabetabellen erwünscht, dann werden diese Tabellen als strukturierte XML-Elemente in die derzeitigen XML-Szenariobaustein eingefügt[16] (Schritt 12). Erst nachdem alle Eingabetabellen befüllt wurden, kann die Überprüfung der Nutzerprivilegien erfolgen (Schritt 13), da in den Authentifizierungsdaten auch Einschränkungen auf bestimmte Wertebereiche der Eingabetabellen zulässig sind.

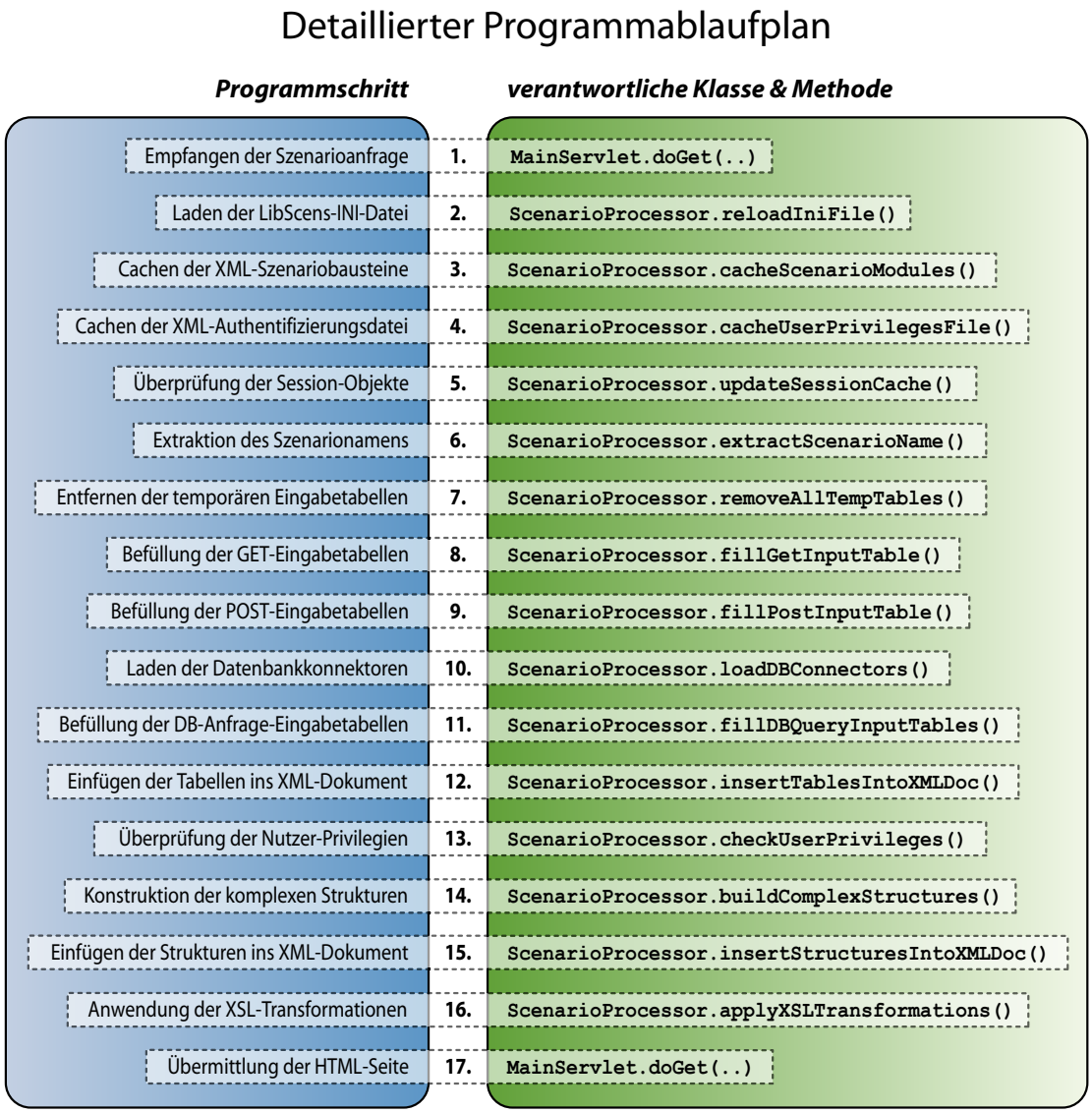

Abbildung 28: *Detaillierter Programmablaufplan von* ***LibScens***

[16]Dafür muss in der INI-Datei das Flag `SCENARIO_CONTAINS_XPATH` auf `true` gesetzt werden.

Anschließend ist die Methode `buildComplexStructures()` für die Konstruktion der komplexen Strukturen verantwortlich (Schritt 14), die im Schritt 15 an die entsprechende Position des Elements `<PageLayout>` der XML-Szenariodatei verschoben werden müssen. Da nun alle erforderlichen Vorbereitungen für die XSL-Transformation abgeschlossen wurden, komplettiert Schritt 16 den Konvertierungsprozess in das Ausgabeformat. Abschließend erfolgt die Rücksendung der erzeugten HTML-Datei mittels HTTP[17] oder HTTPS[18] an den Klienten.

4.5 Ausgewählte Implementierungsdetails

In diesem Kapitel werden wichtige, ausgewählte Implementierungsdetails zu verschiedenen Arbeitsschritten aus Kapitel 4.4 angerissen, welche die Umsetzung unterschiedlicher Konzepte verdeutlichen soll. Es handelt sich hierbei um ausgewählte Stichproben.

4.5.1 Details des Deskriptorenmoduls

Die Deskriptoren dienen aus Programmierungssicht zur späteren Identifikation bzw. Referenzierung und temporären Ablage der Szenariobausteine im `GlobalCache`. Die kompletten XML-Szenariodateien werden beim ersten Aufruf eines **beliebigen** Szenarios in dem `HashVector`-Objekt[19] `scenarioFilesVector` gespeichert.

Hashing-Vektoren eignen sich hervorragend zur Ablage von Daten, die später durch ihren Namen referenziert werden. Sie bieten neben der Identifikation der `Array`-Elemente durch deren Schlüsselworte die Möglichkeit des Zugriffs durch den Positionswert in der Liste. Es sind also folgende Zugriffe möglich: `getObject(String scenarioName)` oder `getObject(int scenarioNumber)`. Beide Methoden haben ihre Berechtigung. Während auf einzelne Objekte des `HashVector`s mittels Namen zugegriffen wird, bietet sich für die iterative Verarbeitung der Zugriff durch deren Zeilennummer im Vektor an. Die unterliegende Hash-Funktion optimiert dabei speziell den Zugriff auf *einzelne* Objekte des Vektors. Diese Optimierung geht zu Lasten einer geringfügig erhöhten Speicherauslastung, welche aber im Allgemeinen vernachlässigbar gering ausfällt, da der sekundäre Hash-Index lediglich Referenzen auf die originalen Objekte verwaltet. Aufgrund dieser Tatsache werden Hashing-Vektoren bei der Umsetzung von ***LibScens*** sehr häufig und in verschiedenen Modulen genutzt.

Wie im Kapitel 3.3.4 bereits erwähnt, werden im Deskriptor ebenfalls die Konnektoren zu verschiedenen Datenbanken erstellt. Für den Prototyp ist eine Anbindung an IBM DB2 geplant, da dieses Datenbanksystem derzeit als Back-End für eNoteHistory dient. Wünscht man alternative Konnektoren, muss die Methode `loadDBConnectors()` der Klasse `ScenarioProcessor.java` entsprechend angepasst und das zugehörige XML-Schema erweitert werden, beispielsweise um die Hilfsprozedur `<FillTableContentManager>`[20] für eine Anbindung des IBM Content Managers. Die Datenbankanbindungen werden ebenfalls global in einem Hashing-Vektor abgelegt und beim ersten

[17] HTTP: Hypertext Transfer Protocol

[18] HTTPS: Hyptertext Transfer Protocol Secure

[19] Vektoren stellen in Java eine listenförmige Speicherungsform für Objekte dar und sind vergleichbar mit `Arrays`. Sie sind durch den Einsatz von Objektzeigern jedoch wesentlich flexibler und leichter erweiterbar (z.B. `HashVector`).

[20] Selbstverständlich muss in diesem Fall auch eine angepasste Hilfsprozedur mit dem Namen `FillTableContentMangager` implementiert werden. Diese muss dann lediglich, wie im Kapitel 4.5.2 beschrieben, in das Package `DE.LibScens.HelpFunctions` eingeordnet werden.

Szenarioaufruf initialisiert. Dabei versucht ***LibScens*** die Verbindung zur Datenbank solange wie möglich aktiv zu halten, denn der Aufbauprozess der Verbindung benötigt, je nach Verbindungsart und Netzwerkkonfiguration, eine nicht unerhebliche Zeitspanne. Sollte die Verbindung getrennt werden, dann wird das `Connection`-Objekt aus dem Vektor gelöscht und bei der nächsten Anfrage wird das Programm erneut versuchen, diese spezielle Verbindung wiederherzustellen. Die Konnektoren werden später im Eingabetabellenmodul durch deren Namen (Element `ConnectorName`) referenziert.

4.5.2 Interface und dynamisches Laden der Hilfsfunktionen

Die zentrale Rolle der Hilfsfunktionen und -prozeduren wurde bereits in Kapitel 3.3.3 ausführlich erläutert. Da die Hilfsfunktionen leicht erweiterbar und anpassbar sein müssen, ist eine Schnittstellendefinition einer Hilfsfunktion unabdingbar. Nur so kann gewährleistet werden, dass alle Methoden, welche Hilfsfunktionen implementieren, auf die gleiche Art und Weise aufgerufen werden können. Das dazugehörige Gerüst zeigt Abbildung 29.

```
public class HelpFunctionName {

  // Die Methode, welche fuer die Abarbeitung des Knotens verantwortlich ist,
  // besitzt den gleichen Namen wie die zugehoerige Klasse bzw. Hilfsfunktion.
  public static void HelpFunctionName (Logger logger, Node node, SessionCache
      sessCache) {

    if (node != null) {

      // Bearbeitung des aktuellen DOM-Knotens und entsprechende Konvertierung
      // und Ersetzung des Knotens mit dem Rueckgabedatentyp und Rueckgabewert
      // der Hilfsfunktion. Bei Hilfsprozeduren wird der Knoten im Allgemeinen
      // abgearbeitet und danach geloescht.
      ..

    }
  }

  // Die init-Methode dient zur Festlegung der Prioritaet und gibt ausserdem
  // bei erfolgreicher Initialisierung eine kurze Beschreibung der Funktions-
  // weise der Hilfsfunktion aus.
  public static Integer init (Logger logger) {

    logger.debug ("Kurze Beschreibung der Funktionsweise");

    // Moegliche Prioritaetswerte sind je nach Abarbeitungsreihenfolge
    // Routines.PRIORITY_BOTTOM_UP oder Routines.PRIORITY_TOP_DOWN.
    return priority;
  }
}
```

Abbildung 29: *Schnittstellenbeschreibung für eine Hilfsfunktion bzw. -prozedur*

Alle Methoden, die dem Gerüst genügen, müssen in dem Package `DE.LibScens.HelpFunctions` abgelegt werden, denn ***LibScens*** wird mittels der Methode `Routines.registerHelpFunctions(..)` versuchen, alle dort gespeicherten Hilfsfunktionen innerhalb der globalen Speicherstruktur zu registrieren. Dies wird dynamisch zur Laufzeit mit Hilfe der `Reflection`-API von Java realisiert. Der große Vorteil ist, dass dann für das Einfügen neuer Hilfsfunktionen keine Anpassung des Quelltextes von ***LibScens*** erforderlich ist. Das dynamische Auslesen wird exemplarisch in Abbildung 30 deutlich.

```
// Lege alle Dateien des Verzeichnisses, in dem die Hilfsfunktionen
// gespeichert sind, in einem Vector ab.
Vector allRecurseFiles = getRecursiveFiles (helpFunctionsPath);

// Lade jede Klasse in den HashVector registeredHelpFunctions.
for (int i = 0; i < allRecurseFiles.size (); i++) {

  String curClassName =
    ( (PackageClassPriorityTriple) allRecurseFiles.get (i)).functionName;
  String curPackageName =
    ( (PackageClassPriorityTriple) allRecurseFiles.get (i)).packageName;

  try {

    // Versuche die aktuelle Hilfsmethode durch die Reflection-API
    // einzubinden.
    Class curClass = Class.forName (curPackageName + "." + curClassName);

    // Initialisiere die Klasse.
    Method initMeth = curClass.getMethod ("init", new Class[] {
      Class.forName ("org.apache.log4j.Logger")});

    // Versuche die zugehoerige Methode init() aufzurufen und speichere
    // die zugehoerige Prioritaet ab.
    Integer priority = (Integer) initMeth.invoke (null, new Object[]{logger});
    ( (PackageClassPriorityTriple) allRecurseFiles.get (i)).priority =
        priority.intValue ();

    // Registriere die Hilfsfunktion im globalen Cache.
    globCache.functionAdd (curPackageName, curClassName, priority.intValue());
  }

  catch (Exception ex) {
    logger.error ("Error while registering a Help Function!", ex);
  }
}
```

Abbildung 30: *Dynamisches Einbinden von Klassen durch die Reflection-API*

4.5.3 Rekursive Abarbeitung von DOM-Knoten

Da im vorherigen Abschnitt beschrieben wurde, wie die Hilfsfunktionen geladen werden, kann nun die Abarbeitung der Elemente des XML-Szenariobausteins erfolgen. Dies geschieht beginnend von dem Wurzelelement des XML-Szenariobausteins. Jedes aktuelle XML-Element wird dafür mit dem Pool aller registrierten Hilfsfunktionen abgeglichen. Sollte es sich also bei dem aktuellen XML-Element um eine Hilfsfunktion handeln, dann wird diese entsprechend ihrer Priorität (Bottom-Up oder Top-Down) abgearbeitet. Den entsprechenden Quelltext für die Methode `processNode(..)` zeigt Abbildung 31.

```
public static synchronized Node processNode (Logger logger, Node node,
    SessionCache sessCache) {

  boolean childrenAlreadyProcessed = false;

  // Ueberpruefe den Abarbeitungstyp (Bottom-Up or Top-Down).
  for (int k = 0; k < c.functionGetVectorSize (); k++) {
    String testFunctionName = c.functionGet (k).functionName;
    String testPackageName = c.functionGet (k).packageName;
    int testPriority = c.functionGet (k).priority;
    String curNodeElemName = node.getName ();
```

```
    // Falls die Abarbeitung Top-Down ist, wird sofort die Hilfsfunktion
    // aufgerufen. Die Methode des Knotennamens ist dann fuer die Verarbeitung
    // der Kindelemente des Knotens verantwortlich!
    if (testFunctionName.equals (curNodeElemName) && testPriority == Routines.
        PRIORITY_TOP_DOWN) {

      callHelpFunction (logger, testPackageName, testFunctionName, node,
          sessCache);
      childrenAlreadyProcessed = true;
    }
  }

  // Nur falls der Knoten vom Typ Bottom-Up oder nicht bekannt ist (Unter-
  // element eines Vaterknotens), muessen die Kinder weiter bearbeitet werden.
  if (!childrenAlreadyProcessed) {

    // Falls der Knoten Kinder hat...
    List nodes = node.selectNodes ("./*");

    // ... werden diese zuerst rekursiv abgearbeitet.
    for (int i = 0; i < nodes.size (); i++) {
      processNode (logger, ( (Node) nodes.get (i)), sessCache);
    }

    // Erst nachdem alle Kinder bearbeitet wurden, wird die entsprechende
    // Hilfsmethode fuer den Knoten selbst aufgerufen.
    for (int k = 0; k < c.functionGetVectorSize (); k++) {
      String testFunctionName = c.functionGet (k).functionName;
      String testPackageName = c.functionGet (k).packageName;
      String curNodeElemName = node.getName ();

      if (testFunctionName.equals (curNodeElemName)) {
        callHelpFunction (logger, testPackageName, testFunctionName, node,
            sessCache);
      }
    }
  }

  // Der abgearbeitete Knoten wird zurueckgegeben.
  return node;
}
```

Abbildung 31: *Rekursive Abarbeitung der DOM-Knoten des XML-Szenariobausteins*

4.5.4 Integration eines Logbuchs

Wie wichtig die Einbindung von Logbüchern in modernen Implementierungen geworden ist, wird z.B. durch die Integration der Logging-API in das JDK 1.4 deutlich. Die rudimentären Loggingmechanismen über die Standardausgabe `System.out.println(..)` genügen den aktuellen Ansprüchen bei weitem nicht mehr. Progressive Logger können geschachtelt werden, bedienen unterschiedlichste Ausgabemedien und besitzten ein flexibles Layout. Für den Prototypen von ***LibScens*** wird Logging hauptsächlich zu Debug- und Kontrollzwecken eingesetzt. Desweiteren wird das Logbuch für die Messungen bestimmter Laufzeiten zu Evaluationszwecken des Prototyps benötigt (siehe Kapitel 5).

Als Quasi-Standard dieser Kategorie hat sich das Open Source Projekt Log4J von Apache etabliert. Log4J besteht aus drei Hauptkomponenten:

- dem Logger,
- dem Appender und
- den Layouts.

Diese drei Komponenten ermöglichen Entwicklern, Logginginformationen entsprechend ihres Typs und Levels in den Quellcode zu integrieren. Desweiteren kann zur Laufzeit kontrolliert werden, wie diese Logmeldungen formatiert und wo sie ausgegeben werden.

Jedem Loggingprozess geht eine bestimmte Instanz des `Logger`-Objekts voraus. Der Logger ist somit die Basis, an die sämtliche Loginformationen gesendet werden. Die hierarchische Schachtelung des Loggers kommt bei ***LibScens*** vor allem zur Unterscheidung der unterschiedlichen Sessions zum Tragen. So ist auch im Nachhinein eindeutig festzustellen, durch welche Session-ID und Programmabfolgen beispielsweise Fehler ausgelöst wurden. Den Nachrichten, welche an den Logger gesendet werden, muss außerdem ein bestimmter Dringlichkeitslevel zugeteilt werden: `debug`, `info`, `warn`, `error` oder `fatal`. Dadurch können Nachrichten später auch auf bestimmte Dringlichkeitsklassen hin gefiltert werden. Das Setzen eines NDC-Wertes[21] erleichtert die Nachverfolgung geschachtelter Inhalte. So kann beispielsweise die Baumtiefe von XML-Elementen in diesem Feld gespeichert werden und somit Aufschluss über die rekursive Abarbeitungsreihenfolge der Hilfsfunktionen geben (siehe Kapitel 4.5.3).

Das Ausgabemedium des Loggers wird durch einen so genannten Appender bestimmt. Mögliche Arten von Appendern sind z.B. `ConsoleAppender`, `SocketAppender`, `FileAppender` oder `HTMLAppender`. Einem Logger können dabei auch mehrere Appender zugeordnet werden. Der `ConsoleAppender` ersetzt beispielsweise die Standardausgabe `System.out` und über den `SocketAppender` können die Logginginformationen über eine Standard-Socket-Verbindung versendet werden. Für den Empfang solcher Socket-Nachrichten gibt es wiederum eine Vielzahl von grafischen Front-Ends. Eines der interessantesten Projekte in diesem Bereich ist **Chainsaw V2**. Abbildung 32 zeigt den Einsatz dieser grafischen Logging-Schnittstelle.

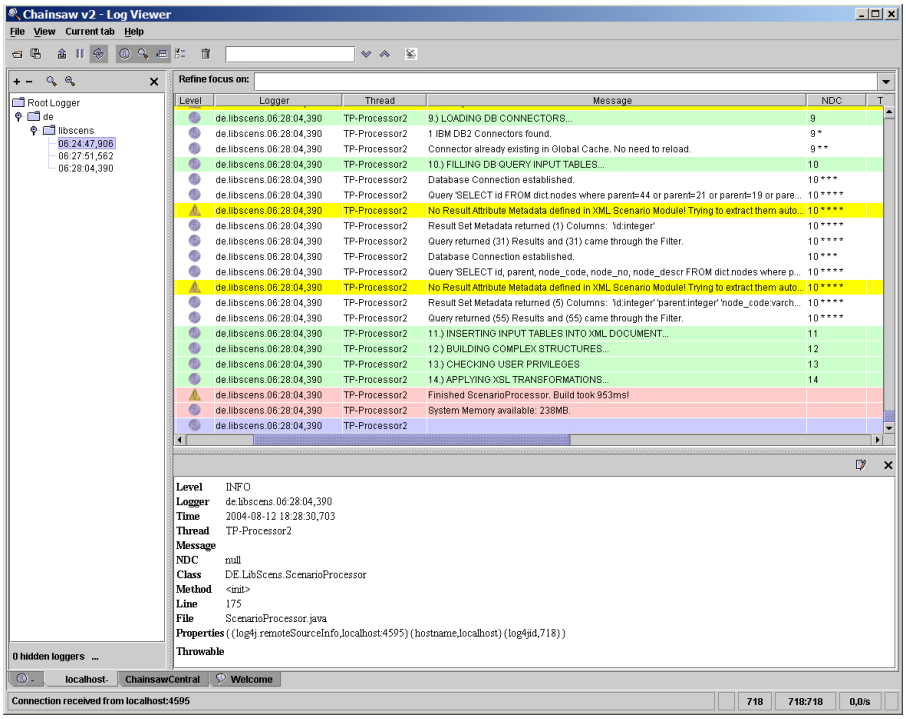

Abbildung 32: *Beispiel für ein grafisches Front-End für Log4J (Chainsaw V2)*

[21] NDC: Nested Diagnostic Context

Die letzte Komponente von Log4J ist das Hinzufügen von Layouts zu bestimmten Appendern. Dies macht allerdings nur bei direkten Ausgabemedien Sinn, beispielsweise bei Einsatz des `HTMLAppenders` oder des `FileAppenders`.

Da Log4J in allen Projektklassen von ***LibScens*** konsequent integriert wurde, sind detaillierte Informationen über alle Programmschritte in variabler Granularität verfügbar. Über die Initialisierungsdatei `GlobalSettings.ini` können die unterschiedlichsten Einstellungsmöglichkeiten für den Logger vorgenommen werden. Sollte aus Performance-Gründen kein Logging gewünscht werden, können auch alle Appender entfernt werden.

4.5.5 Implementierungsdetails der Eingabetabellen

Da die Eingabetabellen einen weiteren zentralen Punkt des Konzeptes darstellen, wird in diesem Abschnitt auf die Speicherung, den Zugriff und die Caching-Mechanismen der Tabellen eingegangen.

Während die Definition von Tabellen eindeutig ist, kann die Implementierung auf unterschiedlichste Art und Weise durchgeführt werden. Bei ***LibScens*** wird dabei eine dreistufige Objekthierarchie bevorzugt. Hierbei besteht eine Tabelle (Klasse `TableInput`) aus mehreren Tabellenspalten (Klasse `TableColumn`), welche wiederum eine bestimmte Anzahl von Tabellenzellen beinhaltet (Klasse `TableCell`). Während die Tabellenspalten immer durch einen eindeutigen Namen beschrieben werden und deshalb in einem `HashVector` abspeicherbar sind, können konkrete Tabelleninhalte mehrfach vorkommen und müssen daher in einem `Vector` abgelegt werden. Eine Zelle besteht dabei aus dem eigentlichen Zelleninhalt (Objekt `contents`) und einem zugewiesenen Datentyp (Objekt `dataType`). Bei den Datentypen sind alle definierten Typen zulässig, die in Kapitel 3.3.3 beschrieben wurden. Wird kein expliziter Datentyp angegeben, wird der Standardwert `LSString` zugewiesen.

Die drei unterschiedlichen Caching-Methoden für die Eingabetabellen (wie in Kapitel 3.3.5 dargestellt) sind: globale Eingabetabellen, temporäre Eingabetabellen und Zustandstabellen. Abhängig von der Zuweisung des Caching-Typs werden die Eingabetabellen in unterschiedlichen Objekten abgelegt, die dadurch unmittelbar die Lebensdauer und den Geltungsbereich der Tabellen bestimmen.

Globale Tabellen sind aufgrund ihrer allgemeinen Gültigkeit dem Singleton-Objekt `GlobalCache` zuzuordnen und werden in dem `HashVector globalCacheableInputTables` abgespeichert. Um komfortabel auf die einzelnen Tabellen dieses Objektes zuzugreifen, wurden verschiedene Zugriffsmethoden auf den `HashVector` implementiert. Zur Definition globaler Eingabetabellen im Untermodul `<InputTables>` des zugehörigen XML-Szenariobausteins muss das XML-Attribut `CacheType` auf den Wert „`GLOBAL`“ gesetzt werden[22]. Globale Eingabetabellen werden nicht durch die Serviceroutine `removeAllTempTables()` gelöscht, es sei denn es erfolgt eine zwischenzeitliche Änderung des Szenariobausteins oder der INI-Datei.

Temporäre Eingabetabellen können immer nur einem konkreten Session-Objekt zugewiesen werden und sind deshalb in einer Instanz der Klasse `SessionCache`, ebenfalls in einem `HashVector`, gespeichert (Objekt `sessionCacheableInputTables`). Für die Definiton einer temporären Eingabetabelle ist keine Angabe eines speziellen Caching-Typs erforderlich, da der `CacheType` standardmäßig immer „`TEMP`“ ist. Nach jedem Abarbeiten eines Szenariobausteins werden die temporären Eingabetabellen durch die Methode `removeAllTempTables()` bereinigt. Zu den tem-

[22] Beispielsweise in der Hilfsfunktionsdefinition `<FillTableDB2SQL CacheType='GLOBAL'>`.

porären Eingabetabellen zählen auch diejenigen Tabellen, welche die `POST`- und `GET`-Werte beinhalten (Tabellennamen „`table_get_attr`“ bzw. „`table_post_attr`“).

Die beiden unterschiedlichen Zustandstabellen „Zustandstabelle für Session“ (Tabellenname „`table_sess_states`“) und „Zustandstabelle für Szenario“ (Tabellenname „`table_scen_states`“) müssen ebenfalls einer konkreten Session unterstellt werden, da sie direkt vom Nutzer abhängig sind. Zustandstabellen müssen im Gegensatz zu den beiden vorherigen Caching-Typen nicht explizit definiert werden, da sie automatisch bei Erstellung eines Session-Objekts bereitgestellt werden. Innerhalb des `SessionCaches` werden diese Eingabetabellen im Objekt `stateCacheableInputTables` abgelegt. Die Zustandstabellen unterscheiden sich in ihrer Lebensdauer. Während „`table_sess_states`“ die gesamte Session über erhalten bleibt, wird „`table_scen_states`“ durch die Serviceroutine `removeAllTempTables()` bei Eintritt in ein neues Szenario gelöscht.

4.6 Layoutumsetzung mittels XSL-Templates

Die Layoutumsetzung durch XSL-Templates stellt den letzten Schritt zur Generierung des endgültigen Ausgabeformats dar. Dazu wird das XML-Szenario benutzt, das inzwischen durch die Befüllung der Eingabetabellen und Konvertierung in komplexe Strukturen für die Transformation mittels XSL-Templates vorbereitet wurde. Unter Verwendung der Layoutbeschreibung des XML-Elements `<PageLayout>` werden nun alle verfügbaren XSL-Regeln der Datei `LibScensMainTemplate.xsl`[23] angewendet. Innerhalb der Hauptvorlage werden nun alle Repräsentationsformen der konkreten Datentypen und komplexen Strukturen eingebunden. Diese befinden sich in separaten Dateien, welche im gleichen Verzeichnis wie die Hauptvorlage abgespeichert sind. Zur Vereinfachung wird davon ausgegangen, dass nicht nur die Standard-XSL-Templates, sondern auch alle projektspezifischen XSL-Templates in dem gleichen Verzeichnis abgelegt werden, in dem sich die Datei `LibScensMainTemplate.xsl` befindet. Dies liegt daran begründet, dass ***LibScens*** sämtliche relativen XSL-Referenzen innerhalb des `MainTemplates` in absolute Referenzen auflösen muss. Die Hauptvorlage wird in Abbildung 33 exemplarisch aufgelistet.

Innerhalb des Templates können nun alle beliebigen Repräsentationsformen für Datentypen und komplexe Strukturen eingebunden und bei Bedarf an das konkrete Projekt angepasst werden. Mittels XPath-Ausdrücken werden dafür die Kombinationsformen von Datentyp und Repräsentationsform überprüft und entsprechend ausgewertet. Für die komplexe Struktur `LSTree` und die Repräsentationsform `LSTreeExplorer`[24] lautet die entsprechende Anfrage beispielsweise:

```
<xsl:template
  match="Structure[@Type='LSTree' and @Representation='LSTreeExplorer']">
```

Wurde ein komplexer Datentyp auf diese Weise ermittelt, dann erfolgt die Abarbeitung entsprechend der Beschreibung des XML-Elements `<xsl:template>`. Auf die gleiche Weise werden auch die Basisdatentypen (siehe Kapitel 3.3.3) abgefangen. Die Standardrepräsentation vom Basisdatentyp `LSText` verdeutlicht Abbildung 34.

[23] Die Datei `LibScensMainTemplate.xsl` ist standardmäßig in dem Ordner `LibScens/XSL Templates/` zu finden.

[24] Die Repräsentationsform `LSTreeExplorer` stellt die komplexe Struktur `LSTree` in der vom Microsoft Explorer bekannten Form dar und bezieht daher auch ihren Namen.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="www.w3.org/1999/XSL/Transform">
  <!--

  Import der externen XSL-Templates -->
  <xsl:import href="LSTreeExplorer.xsl"/>
  <xsl:import href="LSTreeCurrent.xsl"/>
  <xsl:import href="LSText.xsl"/>
  ..
  <!--

  Variablendeklaration -->
  <xsl:variable name="NET_ROOT_IMAGES" select="'C:/LibScens/Img'"/>
  ..
  <!--

  -->
  <xsl:output method="html"/>
  <!-- Die Hauptvorlage -->
  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="/Scenario/Descriptors/@ScenarioName"/>
        </title>
      </head>
      <body>
        <xsl:apply-templates select="/Scenario/PageLayout"/>
      </body>
    </html>
  </xsl:template>
  <!--

  -->
</xsl:stylesheet>
```

Abbildung 33: *Listing des XSL-Templates LibScensMainTemplate.xsl*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="www.w3.org/1999/XSL/Transform">
  <!--

  LSText -->
  <xsl:template match="StructElem[@ElemType='LSText'␣and␣@Representation='
      LSTextStandard']">

    <xsl:choose>
      <xsl:when test="current()/@Action">
        <a href="{current()/@Action}">
          <xsl:value-of select="current()/@Contents"/>
        </a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="current()/@Contents"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!--

  -->
</xsl:stylesheet>
```

Abbildung 34: *Standardrepräsentation des Basisdatentyps LSText*

Wesentlich komplexere Implementierungen von Repräsentationsformen findet man z.B. bei der Abarbeitung der Baumrepräsentationen. Diese nutzen die Möglichkeiten von XSLT 1.0 beispielsweise durch die Anwendung von Variablen und rekursiven Template-Aufrufen nahezu vollständig aus. Für die Auflistung und detaillierten Erklärungen derartiger XSL-Templates wird auf die kommentierten XSL-Projektdateien der beiliegenden CD-ROM verwiesen.

Kapitel 5

Evaluierung des Prototyps

Der im Rahmen dieser Diplomarbeit entwickelte Prototyp von ***LibScens*** muss abschließend auf bestimmte Softwaremerkmale hin ausgewertet werden. In Bezug auf den Entwicklungszyklus des User Centered Designs (vorgestellt in Kapitel 2.1) geht dieses Kapitel mit dem letzten Punkt, *„Auswerten von Designvorschlägen“*, einher. Die Auswertung von Software lässt sich grundlegend in zwei Abschnitte unterteilen:

1. Die **Performance-Messung** stellt den umittelbar messbaren Anteil der Evaluation dar.
2. Obwohl es immer wieder Versuche gibt, die **Usability-Auswertung** zu automatisieren, verkörpert sie hauptsächlich den nicht bzw. nur bis zu einem bestimmten Grad messbaren Anteil der Softwareauswertung und ist damit größtenteils subjektiv.

5.1 Usability-Betrachtungen des Prototyps

Die Usability von ***LibScens*** muss wiederum in zwei Abschnitte unterteilt werden:

1. Einerseits muss der **administrative Gebrauch von *LibScens*** an sich, einhergehend mit der Erstellung der XML-Szenariobausteine, untersucht werden.
2. Andererseits müssen die **konkreten, projektspezifischen Ausgabeformate** auf deren Nutzbarkeit hin analysiert werden.

An dieser Stelle können jedoch lediglich Methoden zur Untersuchung von Usability-Merkmalen angegeben werden, da die Umsetzung einer grafischen Oberfläche einen sehr hohen Zeitanteil des gesamten Softwareentwicklungsprozesses einnimmt. Aktuelle Studien beziffern den Zeitaufwand, der in das Design und die Evaluation einer Oberfläche investiert werden muss, auf ca. 50 Prozent [KOE04] des Gesamtaufwands. Aus diesem Grund kann dieser Aspekt keinen Schwerpunkt dieser Arbeit darstellen. Bei dem Konzept und der Implementierung von ***LibScens*** wurde jedoch vor allem Wert auf den modularen Entwurf gelegt, damit grafische Oberflächen nahtlos in das Projekt integriert werden können. Nicht zuletzt durch den Einsatz von XML-Schema, welche gleichzeitig eine kommentierte Beschreibung für die Szenariobausteine liefern, wird der Entwurf von GUIs[1] adäquat unterstützt. Ein Beispiel, wie eine grafische Schnittstelle aussehen könnte, liefert Abbildung 35.

[1] GUI: Graphical User Interface

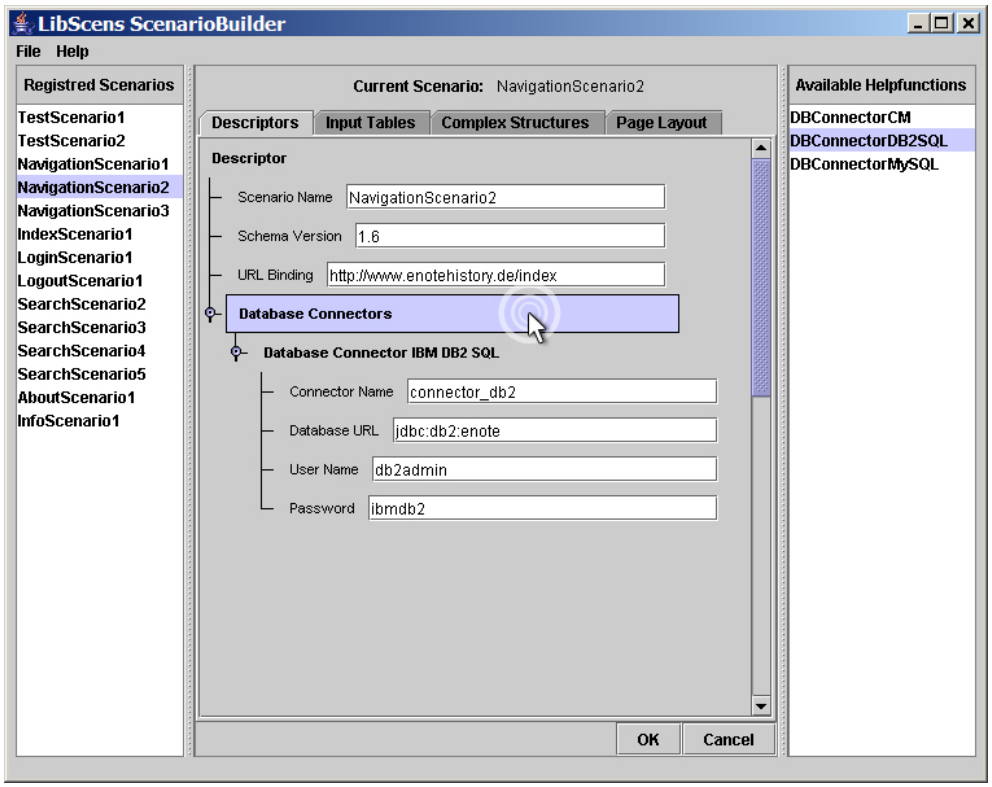

Abbildung 35: *Entwurf einer grafischen Nutzerschnittstelle für* ***LibScens***

Die entwickelten grafischen Interfaces müssen sich dann selbstverständlich den in Kapitel 2.2 angesprochenen Kriterien der Software-Ergonomie stellen. Bei den darauf anzuwendenden Usability-Evaluierungsmethoden unterscheidet man zwischen den folgenden zwei Hauptansätzen:

- Testmethoden,
- Inspektionsmethoden.

Die wichtigsten konkreten Methoden werden in den nächsten Abschnitten kurz vorgestellt. Für einen wesentlich detaillierteren Überblick sei auf das Studium von [HEG03], [LIN94] und [EIC04] verwiesen.

5.1.1 Testmethoden

Testmethoden stellen die fundamentalste Evaluationsmethode dar. Dabei werden Experimente mit realen Nutzern durchgeführt, um dadurch spezifische Informationen über ein bestimmtes Oberflächendesign zu gewinnen. Ihre Auswertung liefert direkte Erkenntnisse über den Umgang der Menschen mit dem Computer und darüber, was ihre exakten Probleme bei Nutzung der konkreten, getesteten Schnittstelle sind. Die Ergebnisse zeigen, inwieweit die Nutzerschnittstelle den Anwender bei der Lösung seiner Aufgabe unterstützt. Die Testmethoden basieren dabei auf der experimentellen Psychologie.

Betrachtet man die Zuverlässigkeit der Ergebnisse, muss man sich die Frage stellen, ob dasselbe Resultat auch erreicht wird, wenn der Test wiederholt wird. Dort werden zwangsläufig, bedingt durch die individuellen Unterschiede der Testpersonen, Probleme auftreten. Hier hat sich jedoch die Maxime durchgesetzt: *„Some data is better than no data."* Anhand mathematischer Nachbetrachtungen, wie statistischen Tests oder der Bildung des Konfidenzintervalls, wird versucht, die Ergebnisse weitgehend zu normalisieren.

Die Gültigkeit der Ergebnisse kann ebenfalls durch die falsche Auswahl der Testpersonen und Testaufgaben zu weiteren Problemen führen. Auch die fehlende Berücksichtigung von Zeitbeschränkungen und sozialen Einflüssen spielt hier hinein.

Testvorbereitung

Neben der Erstellung der Testpläne (Was ist der Zweck dieses Tests? Wo soll er stattfinden? Wie lange soll er dauern?) müssen die Testpersonen, Experimentatoren und Testaufgaben ausgewählt werden.

Bei der Wahl der Testpersonen gilt die Regel, dass die Testpersonen so repräsentativ wie möglich ausgewählt werden sollten, oder, falls dies möglich ist, gleich der eigentliche Endverbraucher bemüht wird. Führt man kleinere Tests durch, dann ist lediglich der durchschnittliche Tester zu überprüfen, bei größeren Tests sollten jedoch mehrere verschiedene Nutzergruppen integriert und auch Randgruppen einbezogen werden.

Die Experimentatoren bzw. Testdurchführer sind bei jeder Methode notwendig. Je erfahrener die Experimentatoren, desto mehr Usability-Probleme sind identifizierbar. Ein Testdurchführer muss dabei selbstverständlich intensive Kenntnisse über die zu testende Applikation und Nutzerschnittstelle besitzen. Deshalb bieten sich auch die GUI-Designer direkt als Experimentatoren an.

Die Testaufgaben sind wiederum so repräsentativ wie möglich bezüglich der Anwendungsdomäne des Systems zu wählen. Sie müssen dabei die wichtigsten Teile der Nutzerschnittstelle abdecken und dürfen nicht trivial sein, jedoch einfach genug, um in der vorgegebenen Zeit gelöst werden zu können. Allgemein gilt, dass sie sehr realistisch und anwendernah sein müssen. Die erste Aufgabe sollte so simpel wie möglich gehalten werden, um dadurch die Motivation und Zuversicht der Testperson zu steigern. Die letzte Aufgabe sollte dann ein greifbares Ergebnis erzeugen, um der Testperson das Gefühl zu geben, etwas Sinnvolles vollbracht zu haben.

Testablauf

Um die Testläufe durchzuführen, muss der Testraum, die Technik und das Material vorbereitet werden. Anweisungen und Fragebögen müssen verfügbar sein und alle potenziellen Unterbrechungsquellen gilt es zu eliminieren. Dann sollte die Testperson in ihren Aufgabenbereich eingeführt werden. Nach einer kurzen Erklärung des Testziels und -zwecks müssen alle nötigen Maßnahmen getroffen werden, um der Testperson jegliche Nervosität zu nehmen. Der Arbeitsplatz muss an die Testperson angepasst werden und der Tester sollte die Möglichkeit für Rückfragen haben.

Bei der Testdurchführung sollte die Testperson nicht alle Aufgaben im Voraus kennen, sondern sie sollten ihr einzeln gegeben werden. Der Experimentator darf dabei nicht mit der Testperson interagieren und sich nicht anmerken lassen, ob die Testperson die Aufgabe gut oder schlecht gelöst hat. Nur wenn die Testperson ohne fremde Hilfe nicht mehr weiterarbeiten kann, darf eingegriffen werden. Bei mehreren Beobachtern sollte dann jedoch auch immer nur einer sprechen.

Bevor die abschließende Befragung der Testperson erfolgt, ist jedoch der Fragebogen auszufüllen. Danach sollten die Kommentare der Testpersonen aufgenommen werden, wobei dem Experimentator die Möglichkeit zum Nachfragen gegeben wird. Die aufgezeichneten Dateien und Fragebögen sind zu nummerieren und der zugehörige Bericht muss erstellt werden. Durch die Nummerierung wird verhindert, dass die Testperson später mit den Ergebnissen in Zusammenhang gebracht werden kann, wodurch die Anonymität gewahrt bleibt.

Konkrete Testmethoden

- **„Thinking Aloud“**
  Bei dieser konkreten Testmethode formuliert die Testperson ihre Gedanken, Gefühle und Meinungen. Dabei sollte sie kritische Antworten geben und periodische Berichte abliefern. Die Aufzeichnung der Daten erfasst dabei eine breite Auswahl an kognitiven[2] Aktivitäten. Dadurch wird das Verständnis bei den Experimentatoren erhöht und ein Einblick in die Terminologie der Testperson geboten. Diese sollten dann Verwendung im Produktdesign und der späteren Dokumentation finden. Thinking Aloud ist eine sehr kostengünstige Methode.

- **„Co-discovery Learning“**
  Hierbei lösen zwei Testpersonen gemeinsam Aufgaben unter Beobachtung. Dabei helfen sich die Testpersonen gegenseitig. Testpersonen sind zu zweit oft unbefangener, wenn sie ihre Gedanken laut äußern sollen. Die Ergebnisse sind dabei häufig realitätsnäher, da sich die Nutzer am Arbeitsplatz ebenfalls gegenseitig helfen. Diese Methode wird vor allem zur Gewinnung quantitativer Daten bevorzugt und häufig in Kombination mit retrospektiven Methoden (z.B. Fragebögen, Post-Test-Interview) eingesetzt.

- **„Question Asking Protocol“**
  Der Experimentator befragt bei dieser Methode die Testperson direkt während des Tests zum Produkt. Das Protokoll stellt also eine Kombination mit „Thinking Aloud“ dar. Dies kann das Verständnis des Experimentators bezüglich des mentalen Musters der Testperson fördern und zeigt Benutzungs- und Verständnisprobleme der Testperson auf. Oftmals wirkt diese Methode natürlicher als „Thinking Aloud“, da die Testpersonen nicht „mit sich selbst“ reden müssen.

Ethische Aspekte

Alle aufgelisteten Testmethoden sind mit tiefem Respekt für die Emotionen und das Wohlsein der Testperson durchzuführen, da die Testpersonen unter enormen Druck stehen. Die Experimentatoren sind unmittelbar für dieses Wohlbefinden vor, während und nach dem Test verantwortlich. Wie schon angesprochen sollten die ersten Aufgaben einfach sein, um die Testperson zu motivieren. Desweiteren sollte der Testperson nach dem Test die Möglichkeit gegeben werden, Anmerkungen machen zu dürfen. Die Testperson sollte über Irreführungen des Systems nach dem Test aufgeklärt werden, damit der fehlerhafte Eindruck über das System auf Seiten der Testperson vermieden wird.

[2] „Kognitiv“ bedeutet soviel wie „die Erkenntnis betreffend, auf ihr beruhend, erkenntnismäßig“.

5.1.2 Inspektionsmethoden

Im Gegensatz zu den Testmethoden bezeichnen die Inspektionsmethoden eine Sammlung von Verfahren, die darauf basieren, dass Inspektoren usability-bezogene Aspekte einer Nutzerschnittstelle untersuchen. Die Inspektoren sind in diesem Fall Usability-Spezialisten, Softwareentwicklungsberater mit fundierten Fachkenntnissen oder Endanwender mit Inhalts- oder Aufgabenkenntnis. Die eigentliche Evaluation beruht dabei auf der Beurteilung bzw. Meinung der Inspektoren.

Vorab müssen bestimmte Fragen, welche den Betrachtungsrahmen definieren, beantwortet werden: Wie evaluiert ein Inspektor tatsächlich eine Nutzerschnittstelle? Denn jede Methode benutzt verschiedene Richtlinien oder Prinzipien (Heuristiken, Checklisten, Styleguides). Sollen die Inspektoren ein Interface auf spezielle Endanwenderaufgaben hin eruieren oder soll das gesamte System aufgrund offener Anweisungen ausgewertet werden?

Die Erfahrungen des Inspektors müssen ebenfalls abgeklärt werden. Welche Kenntnisse sollte ein Inspektor haben und wer soll die Inspektionen durchführen? Usability-Spezialisten besitzen viel Fachkenntnis, Softwareentwickler benötigen eine Einführung in nutzerorientierte Aspekte und Nichtexperten (eventuell Endanwender) haben nur eine geringe Kenntnis der konkreten Methoden.

Auch unter den Inspektionsmethoden unterscheidet man zwischen einzelnen Inspektionen oder Gruppeninspektionen. Bei einzelnen Inspektionen arbeiten die Inspektoren unabhängig voneinander und führen ihre Berichte erst hinterher zusammen. Die Inspektoren beeinflussen sich nicht gegenseitig und sind unbefangener. Gruppeninspektionen haben den Vorteil, dass sich Inspektoren wechselseitig bekräftigen und helfen, allgemeine Probleme zu formulieren und zu notieren. Eine Kombination beider Arten ist oftmals aufgrund des Zeit- und Personalmangels problematisch.

Wie voraussagend sind nun die Ergebnisse für die Endanwenderprobleme? Oftmals werden Probleme bei szenarienbasierten Inspektionen nicht erkannt, andere Probleme sind für Inspektoren schwerer zu erkennen und zu beschreiben, wie z.B. motorische und kognitive Probleme. Bei Inspektionen ohne Endanwender spielt das Verhältnis der Intuition von Inspektor und Endanwender eine große Rolle.

Die Wirksamkeit der gewonnenen Informationen beginnt mit der Liste von Berichten. Probleme zu kennen reicht oftmals aus, um einen einfachen Weg der Behebung zu finden. Inspektoren sollten jedoch auch spezifische Verbesserungsvorschläge beschreiben. Die Probleme müssen dabei priorisiert und ihre Kosten eingeschätzt werden. So wird ein unverhältnismäßig hoher Aufwand für die Fixierung nicht schwerwiegender Fehler vermieden. Probleme können jedoch auch komplexe Auswirkungen haben und mehrere Designkonzepte und Schnittstellenmerkmale umfassen.

Die Methode der Inspektion ist erst dann möglich, wenn bereits eine Nutzerschnittstelle existiert, die jedoch noch nicht implementiert sein muss. Sie ist deshalb nicht für den Einsatz in sehr frühen Phasen des Entwicklungszyklus geeignet. Auch für sehr späte Entwicklungsphasen ist dieses Verfahren nicht zu empfehlen. Wurde das System schon an Kunden ausgegeben, dann sind eher Feldstudien angebracht. Sie sind ideal als Teil eines iterativen Designprozesses, in dem Inspektionsmethoden mit anderen kombiniert werden können.

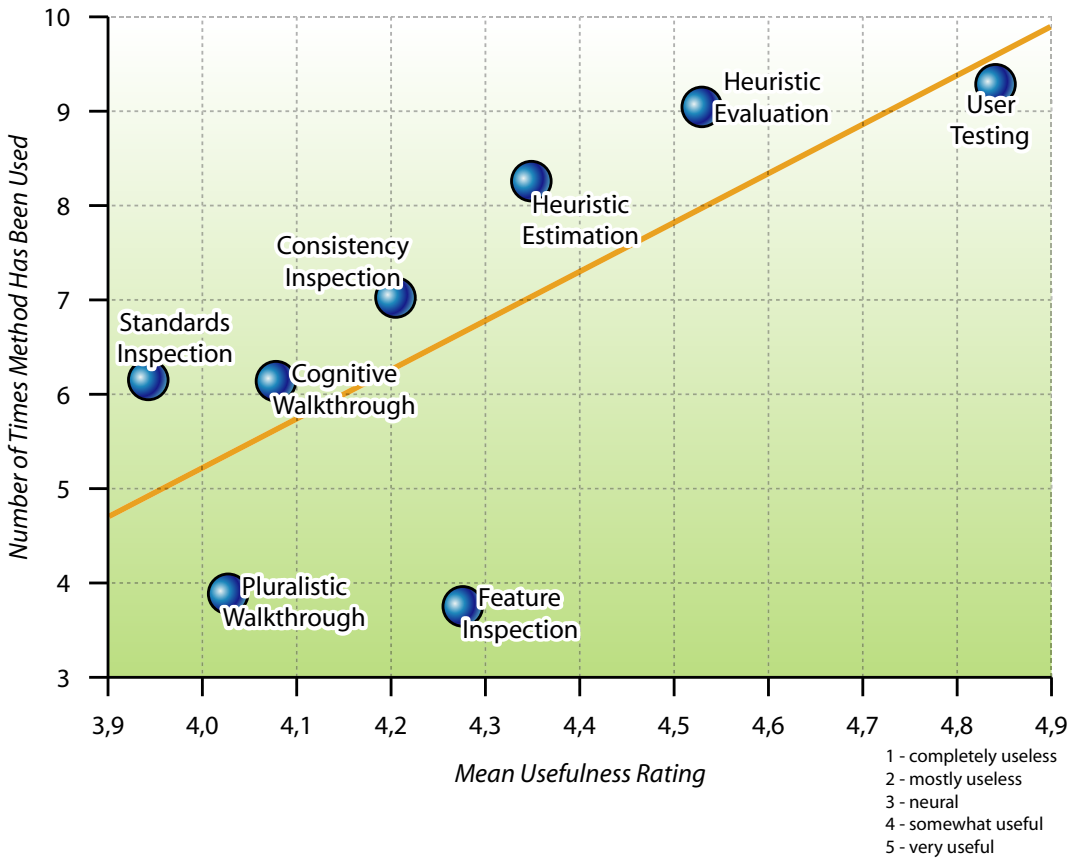

Abbildung 36: *Bezug zw. Häufigkeit und eingeschätztem Nutzen der Methoden [NIE95]*

Inspektionsmethoden

- **„Heuristische Auswertung“**
  Eine Heuristik kann als Guideline, generelles Prinzip oder Daumenregel aufgefasst werden, die eine Designentscheidung lenken oder kritisch beurteilen kann[3]. Zusammenhänge zwischen der Systemsprache und der realen Welt sind herzustellen, systemspezifische Ausdrücke sollten ausdrücklich vermieden werden. Die Nutzerkontrolle kann überprüft werden und das ästhetische und minimalistische Design ist ebenfalls evaluierbar. Es sollte eine kritische Abhandlung über ein System mit Hilfe von relativ simplen und allgemeingültigen Heuristiken erfolgen. Diese Methode ist in jeder Softwareentwicklungsphase einsetzbar, jedoch in der Designphase am effektivsten.

- **„Kognitiver Walktrough“**
  Hierbei untersucht der Inspektor oder eine Gruppe von Inspektoren eine Nutzerschnittstelle bezüglich Verständlichkeit und Erlernbarkeit, indem sie einen Satz von Aufgaben lösen. Das Interface ist dabei entweder ein Papier-Modell, ein funktionstüchtiger Prototyp oder eine voll entwickelte Schnittstelle. Auch diese Methode ist am effektivsten im Designprozess.

[3]Beispielsweise muss ein System den Nutzer durch ein vernünftiges Feedback darüber informieren, welche Bearbeitungsschritte das System gerade erledigt.

Es handelt sich um eine aufgabenorientierte Inspektionsmethode. Der Usability-Experte erforscht dabei die Funktionalität in Form eines imaginären Benutzers. Dabei geht er von der Annahme aus, dass der Nutzer beim Erkunden der Software den Weg des geringsten kognitiven Aufwands gehen wird. Der kognitive Walkthrough stützt sich dabei auf Theorien des explorierenden Lernens und Problemlösens. Damit sollte diese Methode einem Usability-Test vorausgehen, um auf diese Weise offensichtliche Usability-Probleme vor dem Test zu beheben.

- **„Pluralistischer Walkthrough“**
  Im Gegensatz zum kognitiven Walkthrough werden hier die Aufgaben von einer Gruppe von Nutzern, Entwicklern und Usability-Experten am System gelöst. Dabei muss eine Person die Rolle des Koordinators innehaben. Das System liegt hierbei lediglich in Form eines Papierprototypen vor. Bei jedem Schritt der Lösung der Aufgaben schreibt jeder Inspektor zum gerade dargestellten Panel seine Aktionen auf, die er in diesem Fall ausführen würde. Danach beginnt die Diskussion, wobei die Nutzer zuerst sprechen sollten. Letztendlich beschreibt der Koordinator, welche Aktion der Anwender hätte ausführen müssen. Die Gruppenarbeit deckt dabei eine größere Bandbreite von Fähigkeiten und Perspektiven ab und beschleunigt somit die Beschreibung der Probleme.

- **„Feature Inspection“**
  Hierbei liegt die Konzentration auf einer Menge von Charakteristika eines Produktes. Dem Inspektor wird ein Anwendungsfall mit dem Endergebnis gegeben, das erreicht werden soll. Jedes Merkmal wird dann bezüglich seiner Verfügbarkeit, Verständlichkeit und anderen Usability-Aspekten analysiert. Die Merkmale sind in der Reihenfolge aufgelistet, wie sie bei der Durchführung der Aufgaben erscheinen. Mit dieser Methode können typische, immer wiederkehrende Fehler gefunden werden: zu lange Sequenzen, Schritte, die ein erhebliches Vorwissen voraussetzen, oder Dialogschritte, welche Standardnutzer niemals benutzen würden.

Die hier aufgeführten Methoden stellen selbstverständlich nur einen Auszug der existierenden Ansätze dar. Abbildung 36 zeigt noch einmal den Zusammenhang zwischen Häufigkeit und eingeschätztem Nutzen der Methoden.

Für das eNoteHistory-Projekt würde sich als Inspektionsmethode der Pluralistische Walkthrough empfehlen. Die Methode ist in diesem speziellen Fall besonders geeignet, da ein intensiver Gedankenaustausch zwischen Entwicklern und den potenziellen Nutzern stattfindet. Unterstützend sollte ein Usability-Experte eingesetzt werden.

5.2 Performance-Messungen des Prototyps

Im Vergleich zur Usability kann die reine Leistung des Prototyps ohne Umwege ermittelt werden. Um Ergebnisse der Leistungsmessung einzuordnen, müssen jedoch zuerst die Leistungsanforderungen an eine Web-Schnittstelle definiert werden. Danach erfolgt die Aufsplittung von ***LibScens*** in kleinere Module, um zu ermitteln, welche Abschnitte besonders hohe Anforderungen an die Hardware stellen. Nachdem die besonders prozessorlastigen Anteile ermittelt wurden, können alternative Implementierungen bzw. Techniken zu diesem Bereich abgewägt werden.

5.2.1 Festlegung der Anforderungen an die Leistungsfähigkeit

Um überhaupt Aussagen über die zu messenden Zeitwerte von ***LibScens*** zu tätigen, müssen vorher Grenzwerte definiert werden. Ein möglicher Ansatz zur Festlegung der Grenzwerte basiert auf dem Fakt, dass es sich bei den Szenarios von ***LibScens*** um eine Internetanwendung handelt. Die Anforderungen an Web-Anwendungen unterscheiden sich grundlegend von nativen, betriebssystemnah implementierten Anwendungen, welche dadurch komplett auf dem Endrechner lauffähig sind. Die Reaktionszeiten beider Anwendungstypen differieren hierbei zwangsläufig voneinander. Dies ist der Tatsache geschuldet, dass zum einen durch die relativ langsamen Übertragungszeiten der Internetverbindungen und zum anderen durch die zusätzlich anfallenden Transformationsschritte bis zum konkreten Ausgabeformat deutlich Nachteile bei den Web-Applikationen auftreten.

Welche Response-Zeiten werden nun aber von Internetnutzern geduldet und ab welcher Zeitspanne sind die Antworten inakzeptabel? Ein ausgiebige Studie hierzu wird von [NIE00] geliefert. Seinen Untersuchungen zufolge dürfen die Antwortzeiten pro Seite die Zeitspanne von *10 Sekunden nicht überschreiten*. Diese Zeitangabe beziffert das Limit der menschlichen Fähigkeit, die Aufmerksamkeit auf eine bestimmte Aufgabe fokussiert zu halten, während sie auf die Reaktion warten. Die generellen Hinweise die Antwortzeiten betreffend haben sich seit des klassischen, wissenschaftlichen Artikels von Robert B. Miller aus dem Jahr 1968 [MIL68] nicht wesentlich geändert:

- Eine **Zehntelsekunde (0,1 s)** stellt das Limit dar, das dem Nutzer das Gefühl gibt, dass das System unmittelbar auf seine Eingaben reagiert. Damit ist dies das Antwortzeitenlimit für alle Applets, die dem Nutzer etwa die direkte Manipulation zusichern sollen. Damit sind Applikationen gemeint, die es dem Anwender erlauben, Objekte direkt zu bewegen, zu zoomen oder anderseitige Bildschirmelemente in Echtzeit zu manipulieren.

- **Eine Sekunde (1,0 s)** ist ungefähr das Limit eines Anwenders, in dem der Gedankenfluss ununterbrochen bleibt, obwohl der Nutzer die Verzögerung wahrnehmen wird. Für gewöhnlich ist für Antwortzeiten zwischen 0,1 s und 1,0 s kein explizites Feedback nötig, aber der Nutzer verliert das Gefühl, direkt auf den Daten zu operieren. Eine Seite innerhalb einer Sekunde zum Endanwender zu befördern heißt, dass sie mit unmerklicher Verzögerung ankommt.

- Mit **zehn Sekunden (10,0 s)** wird das Limit beziffert, das eingehalten werden muss, damit der Anwender auf seinen Dialog konzentriert bleiben kann. Für längere Verzögerungen wenden sich Nutzer erfahrungsgemäß anderen Aufgaben zu, während sie darauf warten, dass der Computer seinen Prozess beendet. Eine neue Internetseite innerhalb von 10,0 Sekunden zum Anwender zu transportieren bedeutet, dass der Nutzer zumindest auf die Navigation der Seite fokussiert bleiben kann.

Neben der reinen Geschwindigkeit ist auch eine geringe Fluktuation der Reaktionszeiten wichtig. Wenn der Nutzer ein und dieselbe Aktion mehrmals ausführt und das System dabei einmal schnell und beim nächsten Mal langsam reagiert, wissen die Anwender oftmals nicht, was sie zu erwarten haben. So können sie ihr Verhalten nicht adaptieren, um die Nutzung des Systems zu optimieren.

Zwei Ziele für ***LibScens*** können somit umgehend angegeben werden:

1. Die Antwortzeit pro Szenarioaufruf **muss** unter der Obergrenze von zehn Sekunden bleiben.
2. Die Antwortzeit für einen Szenarioaufruf **sollte** sich dem Wert von einer Sekunde annähern.

Werte um 0,1 s sind derzeit im Bereich von Web-Applikationen utopisch und können mit aktueller Hardware nicht erreicht werden.

5.2.2 Aufteilung und Messung der Abschnitte

Nachdem die Ziele im vorhergehenden Abschnitt klar festgelegt wurden, kann nun eine Messung realer Zeitwerte anhand eines ausgewählten Beispielszenarios erfolgen. Um die Leistungsfähigkeit zu testen, wurde ein Szenario gewählt, welches eine in der Größe variable Baumstruktur enthält. Bäume wurden gewählt, da sie im Gegensatz zu anderen Strukturen[4] sehr hohe Hardwareanforderungen an die Abarbeitung der XML-Elemente stellen und die XSL-Transformationen, bedingt durch ihre rekursive Natur, sehr komplex werden und den Prozessor damit extrem belasten. Durch die variable Größe der Baumelemente kann untersucht werden, inwiefern die reine Datenmenge die prozentuale Verteilung der unterschiedlichen Teilmodule von ***LibScens*** beeinflusst.

Alle folgenden Zeitmessungen wurden mit Hilfe von Log4J vorgenommen. Die Testumgebung basiert auf den Hardware- bzw. Softwarekomponenten[5], die in Abbildung 37 aufgeführt werden:

Komponente	Komponentendetails
Prozessor	Intel Pentium 4 mit 2400 MHz
Arbeitsspeicher	1024 MB PC3200 DDR-RAM CL3
Festplatte	Western Digital Caviar WD1600JB 160GB
Grafikkarte	Sapphire Atlantis Radeon 9800 Pro, 128MB DDR
Betriebssystem	Windows XP Professional inklusive ServicePack 1
Datenbanksystem	IBM DB2 Version 8.1 inklusive NetSearchExtender
Webserver	Apache Tomcat 4.1
Weitere Software	Java Developer Kit 1.4.2, DOM4J 1.4, SAXON 8.0, Apache Log4J 1.2.8

Abbildung 37: *Tabelle der eingesetzten Hard- und Software der Testumgebung*

[4] Wie beispielsweise Tabellen.

[5] Da die unterschiedlichen Versionen der Softwarekomponenten mitunter großen Einfluss auf die Performance haben, werden die getesteten Versionsnummern detailliert aufgeführt.

Um die gemessenen Werte der einzelnen Abschnitte in Relation zu setzen, wird vereinfacht der folgende Worst-Case-Fall angenommen: Die Anzahl der Elemente, die aus dem Datenbanksystem gelesen werden, sind identisch mit der Elementanzahl, welche in die komplexen Strukturen überführt werden und identisch mit der Elementmenge, die durch die XSL-Transformationen[6] abgearbeitet werden. In der Praxis wird die Elementmenge tendenziell jedoch eher von Transformationsschritt zu Transformationsschritt weniger werden, als konstant bleiben. Dadurch sollten praxisnahe Zeitmessungen noch Spielraum nach unten besitzen.

Da von vornherein einige Teilmodule aufgrund ihrer Implementierung als Performance-Engpässe ausgeschlossen werden können, erfolgt eine sinnvolle Festlegung der Gruppen für die initialen Messungen. Die Aufteilung wird in Bezug auf die vorgestellten Teilabschnitte von Kapitel 4.4 vorgenommen. Die Gruppen sind im Einzelnen[7]:

- Gruppe 1, `Initialisierung` beinhaltet:

  2. Laden der ***LibScens***-INI-Datei
  3. Cachen der XML-Szenariobausteine
  5. Überprüfung der Session-Objekte
  6. Extraktion des Szenarionamens
  7. Entfernen der temporären Eingabetabellen
  8. Befüllung der `GET`-Attribute
  9. Befüllung der `POST`-Attribute

- Gruppe 2, `Datenbankanfragen` enthält die Module:

  10. Laden der Datenbankkonnektoren
  11. Befüllung der DB-Anfrage-Eingabetabellen
  12. Einfügen der Tabellen ins XML-Dokument

- Gruppe 3, `Komplexe Strukturen` besteht aus:

  14. Konstruktion der komplexen Strukturen
  15. Einfügen der Strukturen ins XML-Dokument

- Gruppe 4, `XSL-Transformationen`:

  16. Anwendung der XSL-Transformationen

Die folgenden drei Abbildungen zeigen die prozentuale Verteilung der Gruppen. Dabei wurden jeweils zehn Messungen pro Diagramm vorgenommen, um etwaige temporäre Fluktuationen, wie beispielsweise durch das Einsetzen der Garbage Collection, zu eliminieren. Desweiteren wurden

[6] Berücksichtigt werden muss jedoch, dass die Implementierung der XSL-Templates erheblichen Einfluss auf die Performance-Messungen hat. Die Baumstruktur stellt aber auch hier wieder den aktuellen Worst-Case-Fall dar.

[7] Die Programmschritte 4 und 13 fallen aufgrund der fehlenden Implementierung des Authentifizierungsmoduls weg.

alle überflüssigen Logginginformationen deaktiviert, um optimale, unverfälschte Werte zu messen. Alle Messungen wurden dabei im laufenden Betrieb vorgenommen, wodurch die Gruppe `Initialisierung` praktisch keine Prozessorzeit beansprucht. Dies ist im Praxisbetrieb ebenfalls zu 95 % der Fall und kann daher auch als repräsentativ angesehen werden. In der Gesamtzeit sind außer den angegebenen Gruppen noch weitere kleinere Abschnitte enthalten, welche nicht explizit erwähnenswert sind, da sie, wie die Studien zeigen, nur einen sehr geringen Anteil der Prozessorzeit ausmachen.

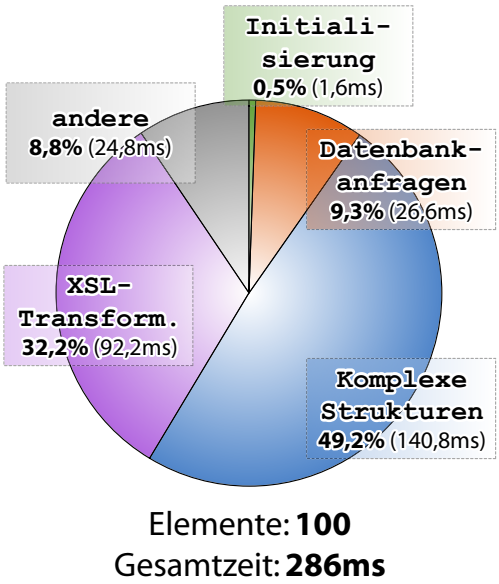

Abbildung 38: *Prozentuale Zeitaufteilung bei 100 gelesenen Elementen*

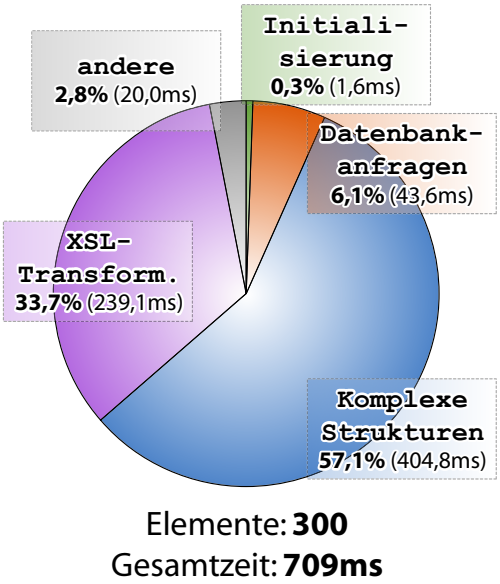

Abbildung 39: *Prozentuale Zeitaufteilung bei 300 gelesenen Elementen*

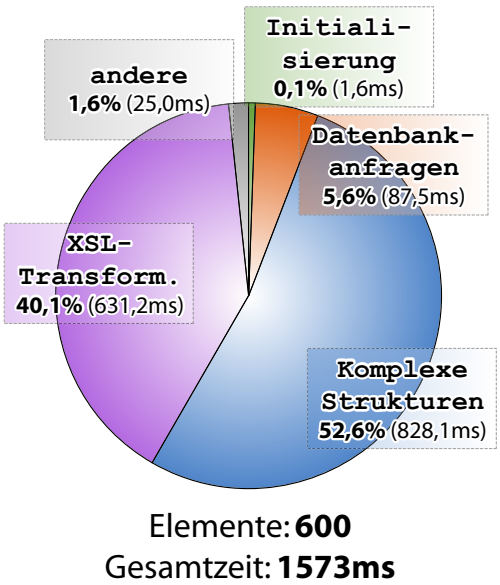

Abbildung 40: *Prozentuale Zeitaufteilung bei 600 gelesenen Elementen*

Wie zu erwarten war, stellen die beiden Gruppen `Komplexe Strukturen` und `XSL-Transformationen` den bei weitem zeititensivsten Anteil dar. Bei wenigen Eingabeelementen (100 Elemente) bewegt sich der Prozentanteil beider Gruppen zusammen um ca. 80 %. Mit steigender Elementanzahl nimmt ihre Bearbeitungszeit jedoch noch weiter zu und kann die 90 %-Marke deutlich übersteigen. Als Hauptgründe hierfür sind sicherlich einerseits die rekursiven Bearbeitungsalgorithmen von Bäumen und andererseits die eingesetzten Techniken wie das Document Object Model und XPath-Anfragen dieses Beispielszenarios anzusehen.

Die in der Gruppe `Initialisierung` zusammengefassten Abschnitte benötigen selbstverständlich nur einen minimalen Anteil des Gesamtaufwands, da diese nur beim Erstellen neuer Szenarios, Ändern von Szenarios, Ändern der INI-Datei oder neuen Session-Initialisierungen zum Tragen kommen. In dem Fall kommen hier einmalig zusätzliche Zeitaufwände im Bereich von Zehntelsekunden hinzu.

Überraschend gering fällt der Anteil der Gruppe `Datenbankanfragen` aus. Selbst bei extremen Datenmengen von 1000 Elementen beträgt das Zeitvolumen hier selten über 100 ms. Dabei muss beachtet werden, dass nicht nur das Abfragen der Datenbankanfragen in diese Gruppe fällt, sondern auch die Speicherung in den Java-Eingabetabellen. Die Zeitaufwände dieses Moduls sind schon zum Zeitpunkt des Prototyps komplett zufriedenstellend und benötigen keine weitere Verbesserung.

Allgemein lässt sich feststellen, dass der Prototyp die in Kapitel 5.2.1 gestellten Leistungsanforderungen teilweise bereits erfüllt, oder sich bei Grenzwerten immer noch in einem akzeptablen Bereich bewegt. Dabei muss berücksichtigt werden, dass die parallele Anforderung durch unterschiedliche Nutzer an dieser Stelle nicht explizit evaluiert wurde.

Da die Gruppen `Komplexe Strukturen` und `XSL-Transformationen` mit Abstand den größten Zeitfaktor ausmachen, werden im folgenden Abschnitt die eingesetzten Techniken ausgewertet und mögliche Alternativen dazu betrachet.

5.2.3 Techniken für komplexe Strukturen und XSL-Transformationen

Da die Überführung der Eingabetabellen in die komplexen Strukturen mit ca. 50 % einen erheblichen Anteil am gesamten Zeitaufwand eines Szenarioaufrufs ausmacht, stellt sich natürlich die Frage nach Alternativen, welche Performance-Vorteile bringen könnten. Die Anforderungen an den XML-Parser sind, dass er zum einen das Document Object Model beherrschen muss und zum anderen in der Lage sein muss, XPath-Ausdrücke adäquat auszuwerten. Alternative Java-Implementierungen von XML-Parsern beherrschen zwar DOM, wie beispielsweise Apache Xerces oder Apache Xerces2, sind jedoch nicht in der Lage, XPath-Anfragen zu bearbeiten. Das bedeutet, dass bei diesen Parsern zusätzlich eine XPath-API eingesetzt werden müsste. Hierfür käme beispielsweise die XPath-API von Apache Xalan in Frage. Nach den Untersuchungen von [BOE04] ist die Umsetzung von Apache Xerces/Xalan jedoch DOM4J in nahezu allen Bereichen deutlich unterlegen. Deshalb stellt DOM4J momentan die beste Umsetzung für die XPath-Bearbeitung von DOM-Bäumen dar. Sollten sich zukünftige Entwicklungen in diesem Bereich als leistungsfähiger herausstellen, kann durch die Nutzung der JAXP-Schnittstelle von Java jederzeit ein Austausch erfolgen.

Dieser Austausch ist ebenfalls für die XSLT-Prozessoren möglich, die für die Umsetzung des Layouts verantwortlich sind. Da es auch bei den XSLT-Prozessoren verschiedene Java-Implementierungen gibt, wurde die Leistungsmessung auf die beiden führenden Produkte beschränkt: Apache Xalan2 und SAXON. Auch in diesem Bereich erweist sich das eingesetzte Werkzeug SAXON leistungsfähiger als das Pendant von Apache. Abbildung 41 zeigt die Zeitmessungen beider Prozessoren beim Einsatz in ***LibScens***. Die Performance-Unterschiede beider Produkte sind dabei sehr signifikant. Auffällig ist, dass Apache Xalan bis zu einer Anzahl von 100 XML-Elementen noch genauso schnell bzw. sogar schneller als SAXON das Ausgabeformat erzeugt. Ab 200 Elementen ist jedoch SAXON wesentlich performanter. Dies steigert sich derart, dass Apache Xalan 2.6.0 bei einer Anzahl von 600 Elemente mehr als die fünffache Bearbeitungszeit in Anspruch nimmt[8]. Aufgrund dieser gravierenden Leistungsunterschiede wurde SAXON als Standard-XSLT-Prozessor für ***LibScens*** gewählt.

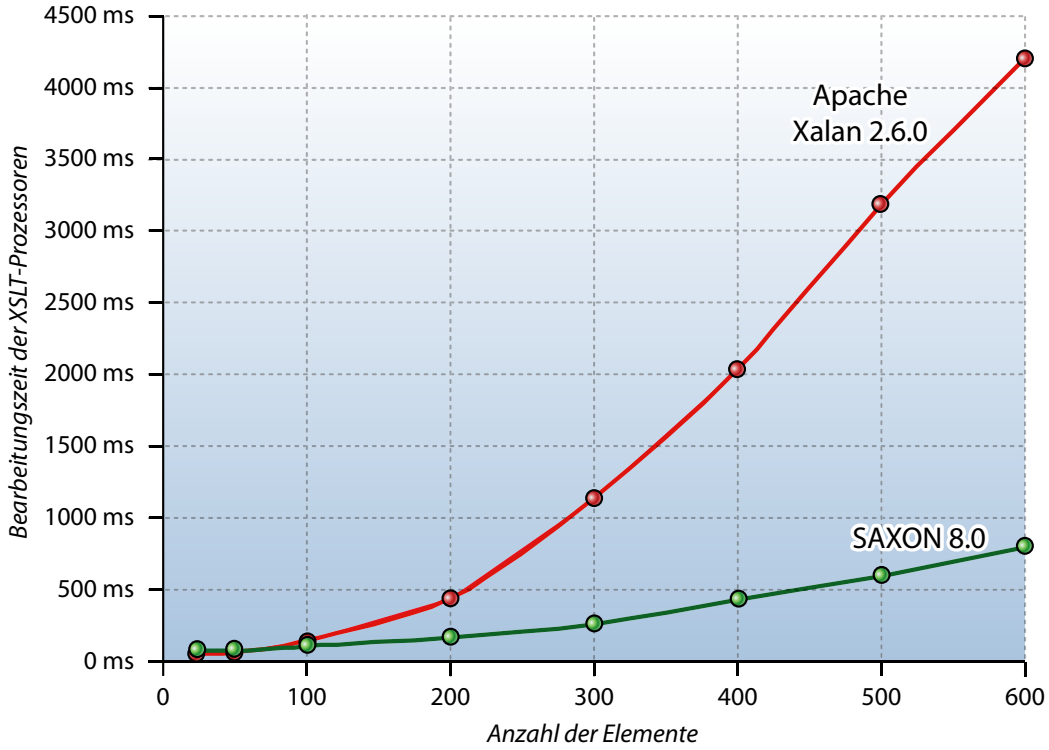

Abbildung 41: *Auswertung der Bearbeitungszeit der XSLT-Prozessoren*

[8] Als Vorlage für die Leistungsmessung wurde dasselbe Baumbeispiel wie in Kapitel 5.2.2 gewählt und dann jeweils die XSL-Anteile ausgewertet.

Kapitel 6

Zusammenfassung und Ausblick

Zusammenfassend lässt sich festellen, dass bezüglich der Aufgabenstellung alle Anforderungen an die *„Konzeption und Umsetzung von multimedialen Nutzerschnittstellen im eNoteHistory-Musikarchiv“* erfüllt wurden. Durch die Studien vorhandener Implementierungen digitaler Bibliotheken konnten die Ziele zur Umsetzung der Schnittstelle definiert werden (siehe Kapitel 2.4). Danach wurde eine dreischichtige Architektur entworfen (siehe Abbildung 13 auf Seite 35), die sich nahtlos in die Gesamtarchitektur des eNoteHistory-Musikarchivs einordnet. Dabei erfolgt eine Aufsplittung des gesamten Projekts in XML-Szenariobausteine (siehe Kapitel 3.3.1), welche sich in die Teilmodule Deskriptoren, Eingabetabellen, komplexe Datenstrukturen und Seitenlayout unterteilen lassen. Mittels des Konzeptes der Hilfsfunktionen bzw. -prozeduren wird ein flexibler Entwurf zur Steuerung des Datenflusses definiert. Die Integration einer an ***LibScens*** angepassten Nutzerauthentifizierung (siehe Kapitel 3.4) rundet die konzeptuelle Phase ab. Im praktischen Teil der Arbeit erfolgte die Implementierung durch einen dreistufigen Transformationsprozess (siehe Kapitel 4.2). Dabei wird das Ausgabeformat HTML mittels fortschrittlicher Techniken wie DOM-Interpretation durch XML-Parser, XPath-Anfragen und XSLT-Prozessoren in einer Java-2-Laufzeitumgebung erstellt. Nach Aufschlüsselung des ausführlichen Programmablaufplans (siehe Abbildung 28 auf Seite 65) wurden ausgewählte Implementierungsdetails vorgestellt. Schließlich konnte der Prototyp auf gewisse Performance-Merkmale hin untersucht werden, wobei er sich trotz des frühen Entwicklungsstadiums schon als leistungsfähig herausstellte.

Damit der Prototyp von ***LibScens*** zu einer finalen Version ausgebaut werden kann, werden nun im Ausblick die dafür notwendigen Anpassungen bzw. Erweiterungen vorgestellt.

Als unmittelbar nächster Schritt zur konsequenten Weiterentwicklung des Prototypen empfiehlt sich die Umsetzung der grafischen Nutzerschnittstelle. Eine GUI für ***LibScens*** sollte die Erstellung und Wartung der XML-Szenariobausteine, die Anpassung der Authentifizierungsdatei und die Änderung der Umgebungsvariablen in geeigneter Weise unterstützen. Für die Erstellung der Szenariobausteine wurde in Kapitel 5.1 eine beispielhafte Oberfläche vorgestellt. Dabei könnte auf der linken Seite die Auswahl eines Szenariobausteins erfolgen. In der Mitte wird dann jeweils ein Modul des Szenariobausteins in den Fokus gerückt (Deskriptoren, Eingabetabellen, komplexe Strukturen oder Seitenlayout) und auf der rechten Seite wäre Raum für die verfügbaren Hilfsfunktionen des aktuell ausgewählten Abschnitts. Durch Drag-&-Drop könnten so dem Szenariomodul Funktionen hinzugefügt werden.

Da ***LibScens*** durch die dreistufige Grundarchitektur eine Vielzahl an unterschiedlichen Datenbanksystemen unterstützt, besteht natürlich auch der Bedarf an verschiedene Anfragesprachen, wie z.B. SQL oder XQuery. Eine Möglichkeit ist die Integration aller speziellen Anfragesprachen in eigene Hilfsfunktionen, wie dies im Prototyp implementiert wurde. Dies bedeutet jedoch, dass der Nutzer auch jeweils alle Sprachen seiner angebundenen Datenbanksysteme beherrschen muss. Ein anderer Ansatz wäre die Definition einer globalen Anfragesprache für ***LibScens*** und der Einsatz von Wrappern, um diese auf die nativen Sprachen der speziellen Datenbanksysteme abzubilden. Der Vorteil hierbei wäre, dass der Adminstrator Kenntnisse von nur einer Sprache beherrschen muss. Außerdem wäre eine Implementierung eines grafischen Editors zur Definition von Sprachkonstrukten wünschenswert. Diese könnte als eigenständiges Projekt entwickelt werden und müsste dann als externes Modul in die GUI von ***LibScens*** eingebunden werden.

Die Integration einer Nutzerverwaltung wurde von der Konzeption bis hin zum Prototypen berücksichtigt. Für eine konkrete Implementierung hierfür ist die Methode `checkUserPrivileges()` der Klasse `ScenarioProcessor` vorgesehen. Dabei sind unterschiedliche Techniken für die Verwaltung einsetzbar. So kann beispielsweise eine globale Authentifizierungsdatenbank angebunden werden, wie beispielsweise LDAP-Server[1]. Oftmals sind jedoch globale Nutzerdaten nicht mit den lokalen Bedürfnissen der digitalen Bibliothek vereinbar oder eine Zusammenführung ist nicht erwünscht. In diesem Fall kann auch eine eigene Implementierung der Authentifizierungsdaten erfolgen, beispielsweise in Form einer XML-Datei. Ein Entwurf für ein entsprechendes XML-Schema wird in Abbildung 25 auf Seite 58 vorgestellt.

Obwohl der Prototyp bereits einen Grundsatz an Caching-Mechanismen beinhaltet, könnten sich zusätzliche Caching-Techniken als leistungssteigernd herausstellen. Alle Implementierungen gehen von einer flüchtigen Ablage der Daten im RAM aus, ein anderer Ansatz wäre die persistente Speicherung von Zwischenergebnissen auf der Festplatte. Dabei müssten alle relevanten Parameter eines konkreten Szenarioaufrufs ermittelt werden, da diese eine Befüllung des Szenarios eindeutig bestimmen. Aus der Kombination der relevanten Parameter könnte ein Hashwert gebildet werden, welcher als Dateiname für die spezielle Szenariobefüllung einzusetzen wäre. Erfolgt nun erneut ein Szenarioaufruf, wird abermals der Hashwert gebildet und daraufhin werden die in Frage kommenden Dateien ermittelt. Existieren eine oder mehrere Treffer, muss geprüft werden, ob die Datei tatsächlich die relevanten Parameter enthält[2]. Diese Caching-Methode ist performanter, wenn die Abarbeitungszeit dieses Algorithmus geringer als die Zeit des adäquaten dreistufigen Transformationsprozesses (siehe Kapitel 4.2) ausfällt.

Die Entwicklung vom Prototypen zur Vollversion könnte durch weitere Performance-Steigerungen abgerundet werden. So ist ein alternativer Algorithmus zur Verarbeitung der XML-Dateien denkbar, der statt einer DOM-Abarbeitung einen angepassten SAX-Parser[3] verwendet und eine eigens an die Bedürfnisse dieses Projekts adaptierte Baumstruktur implementiert. Auch gilt es zu überprüfen, ob durch XSLT 2.0 Leistungssteigerungen von ***LibScens*** erreichbar sind.

[1] Das Lightweight Directory Access Protocol (LDAP) ist in der Computertechnik ein Netzwerkprotokoll, das die Abfrage und die Modifikation von Informationen eines Verzeichnisdienstes erlaubt. Die aktuelle Version ist in RFC 2251 spezifiziert. [WIK04]

[2] Dieser Schritt ist notwendig, da perfekte Hashfunktionen für dynamische Inhalte praktisch nicht realisierbar sind. Das bedeutet, dass pro Hashwert verschiedene Attributkombinationen in Frage kommen. Für derartige Kollisionen gilt es Auflösungsstrategien zu entwickeln. Daher müssen alle gefundenen Treffer nochmals auf die gesuchte Attributkombination hin überprüft werden.

[3] SAX: Simple API for XML

Anhang A

Screenshots digitaler Bibliotheken

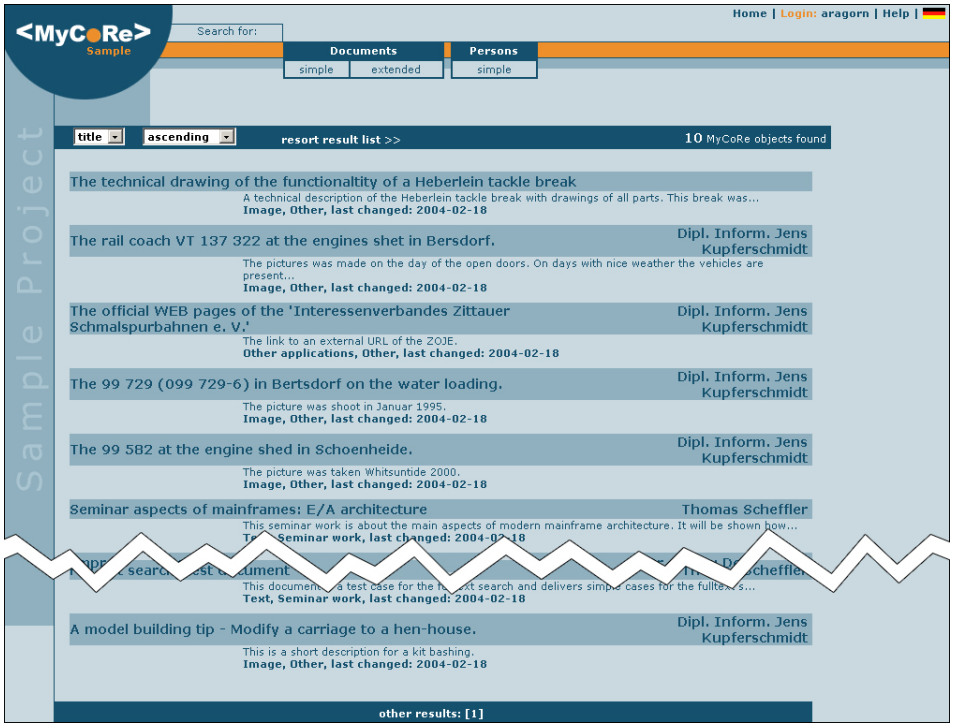

Abbildung 42: *Beispiel für eine grafische Nutzerschnittstelle von MyCoRe*

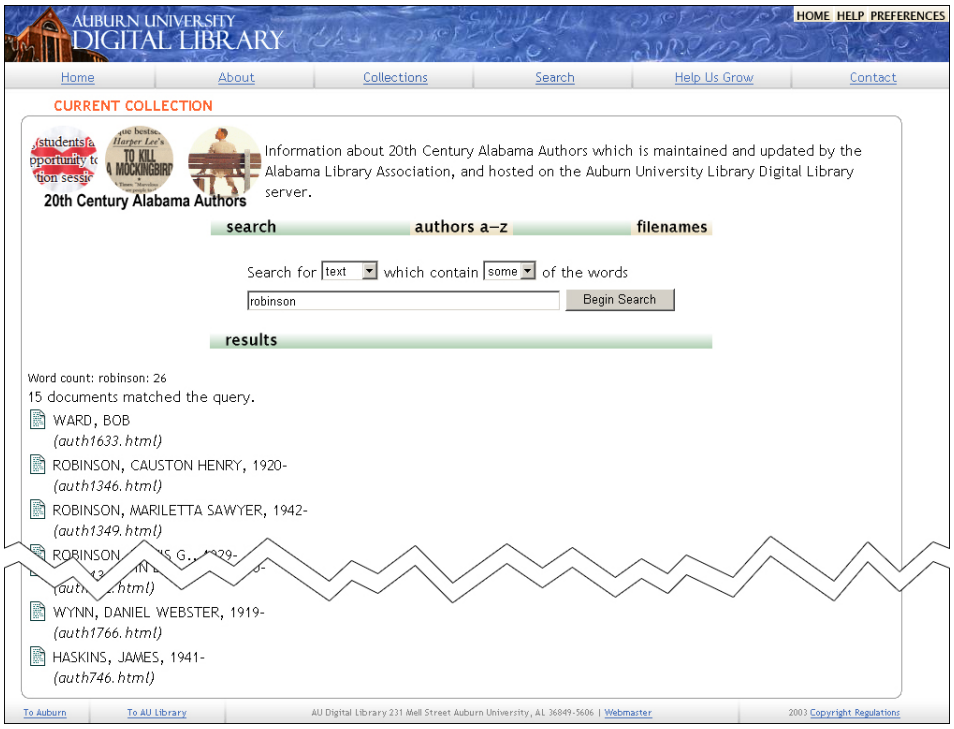

Abbildung 43: *Beispiel für eine grafische Nutzerschnittstelle von Greenstone*

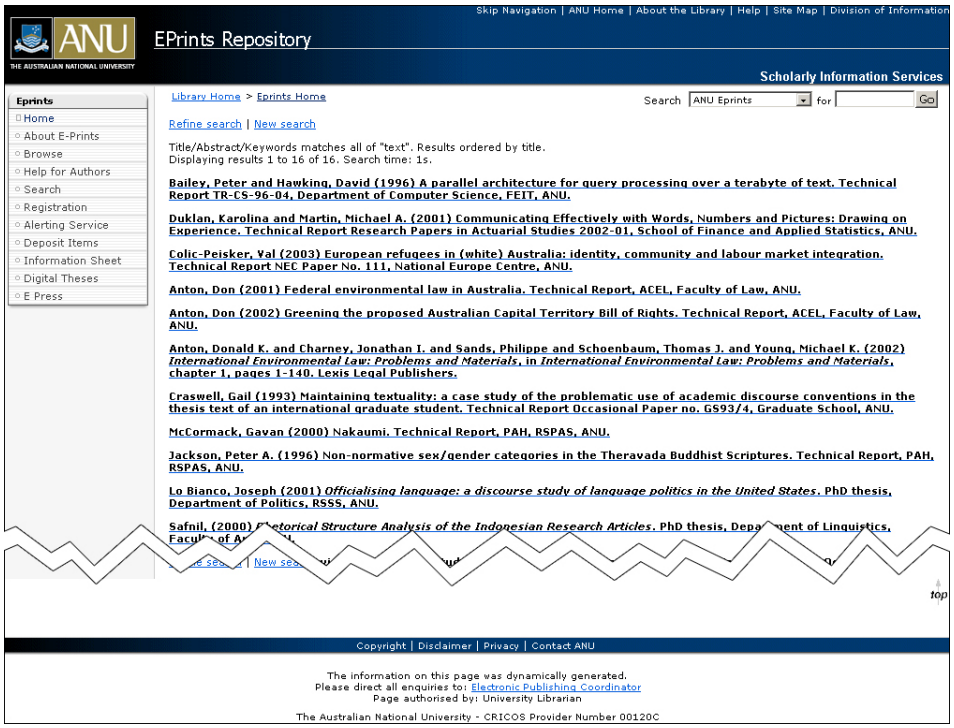

Abbildung 44: *Beispiel für eine grafische Nutzerschnittstelle von Eprints*

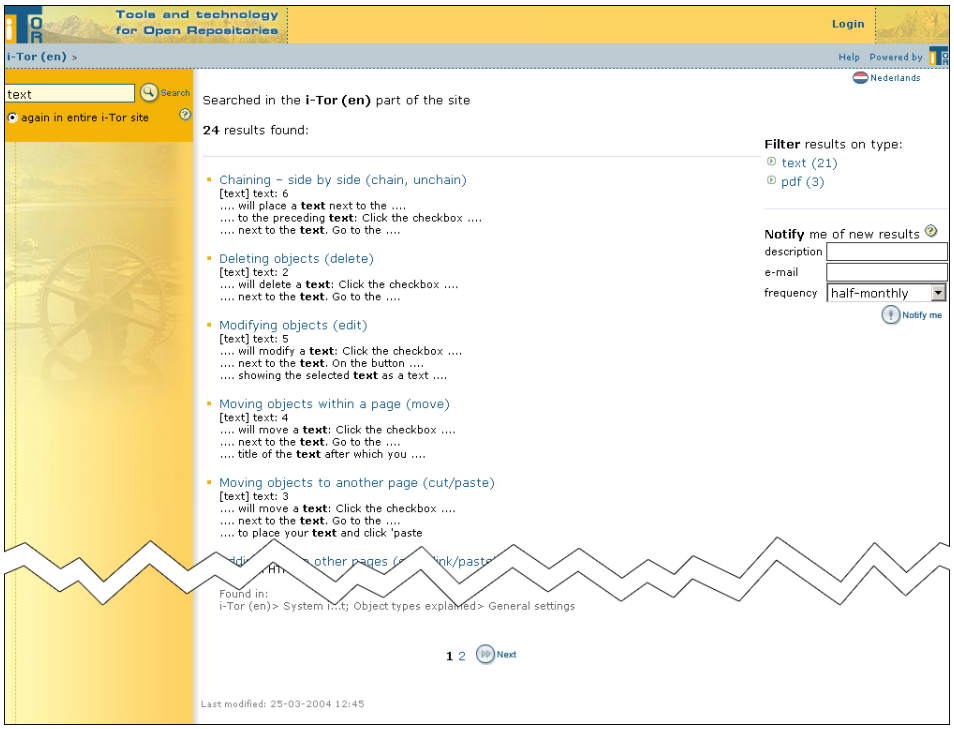

Abbildung 45: *Beispiel für eine grafische Nutzerschnittstelle von i-Tor*

Anhang B

Beispiele

B.1 Beispiel 1 — Entfernung von Knoten

Unter den vorher angesprochenen Hilfsfunktionen (Konzept siehe Kapitel 3.3.3) befinden sich auch Funktionen zur Bearbeitung von `LSString`-Datentypen. So liefert die Funktion `EvalStringConcat` bei Eingabe einer beliebigen Menge an Strings als Rückgabewert erneut einen `LSString`, wobei alle Eingabewerte konkateniert werden. Die Funktion `EvalCellConcat` liefert unter Angabe des Tabellennamens, des Spaltennamens und einer Zeilennummer ebenfalls als Rückgabewert einen `LSString`. Dies bedeutet, dass die Funktionen natürlich auch geschachtelt werden können, da der erwartete Eingabewert der einen Funktion gerade dem Ausgabwert der anderen entspricht. Abbildung 46 zeigt, wie die beiden Funktionen beispielsweise im XML-Szenariobaustein geschachtelt angeordnet sein könnten:

```
<!-- Der unbearbeitete Quelltext -->
<EvalStringConcat>
  <LSString>Dies <LSString>
  <EvalStringConcat>
    <LSString>ist ein <LSString>
    <EvalCellContent TableName="table_get_attr">
      <ColumnName>start</ColumnName>
      <RowNumber>0</RowNumber>
    </EvalCellContent>
  </EvalStringConcat>
</EvalStringConcat>
```

Abbildung 46: *Beispiel 1: Entfernung von Knoten — unbearbeitetes XML-Dokument*

Da alle Knoten vom Typ Bottom-Up sind, beginnt die Abarbeitung bei den Blättern der Knoten. Das erste zu bearbeitende Blatt ist demnach `<EvalCellContent>`, da die Elemente `<ColumnName>` und `<RowNumber>` lediglich Eingabewerte zur Hilfsfunktion `<EvalCellContent>` darstellen und für die rekursive Abarbeitung *nur* die Hilfsfunktionen von Interesse sind. Angenommen, die Funktion `EvalCellContent` liefert als Ergebnis der ersten Ersetzung den Inhalt `<LSString>Test.</LSString>` zurück, dann würde nach der Ersetzung des Knotens das folgende Dokument entstehen:

```
<!-- Das XML-Dokument nach der ersten Ersetzung -->
<EvalStringConcat>
  <LSString>Dies <LSString>
  <EvalStringConcat>
    <LSString>ist ein <LSString>
    <!-- Der ersetzte Knoten <EvalCellContent> -->
    <LSString>Test.</LSString>
  </EvalStringConcat>
</EvalStringConcat>
```

Abbildung 47: *Beispiel 1: Entfernung von Knoten — nach erster Knotenersetzung*

Das nächste Blatt stellt nun die innere `<EvalStringConcat>`-Hilfsfunktion dar. Sie konkateniert die beiden `LSString`s und es entsteht:

```
<!-- Das XML-Dokument nach der zweiten Ersetzung -->
<EvalStringConcat>
  <LSString>Dies <LSString>
  <!-- Der ersetzte Knoten <EvalStringConcat> -->
  <LSString>ist ein Test.<LSString>
</EvalStringConcat>
```

Abbildung 48: *Beispiel 1: Entfernung von Knoten — nach zweiter Knotenersetzung*

Wie erwartet wird nun auch die äußere `<EvalStringConcat>`-Funktion ersetzt werden und es bleibt folgender Ausdruck übrig:

```
<!-- Das XML-Dokument nach der letzten Ersetzung -->
<!-- Der ersetzte Knoten <EvalStringConcat> -->
<LSString>Dies ist ein Test.<LSString>
```

Abbildung 49: *Beispiel 1: Entfernung von Knoten — nach letzter Knotenersetzung*

B.2 Beispiel 2 — Hinzufügen von Knoten

Im Abschnitt (Kapitel 3.3.6) wird genauer auf die Befüllung der komplexen Strukturen eingegangen. Um das Hinzufügen von Knoten zu zeigen, wird an dieser Stelle eine Hilfsfunktion beschrieben, welche im Prinzip einen Iterator über einer Menge von Strukturelementen darstellt. Der Quellcode dafür folgt in Abbildung 50:

```
<!-- Der gekuerzte Quelltext -->
<StructArray ArrayName="buecher" CacheType="TEMP" LoopName="loop1">

  <ArrayElements>
    <LSIntegerArray>
      <LSInteger>3</LSInteger>
      <LSInteger>5</LSInteger>
      <LSInteger>7</LSInteger>
    </LSIntegerArray>
  </ArrayElements>

  <ArrayBody>
    <StructElem ElemName="buchname" ElemType="LSString">
      <Contents>
        <EvalCellContent TableName="table_db_buecher">
          <ColumnName>buch_name</ColumnName>
          <Row>
```

```
            <EvalLoopNumber LoopName="loop1"/>
          </Row>
        </EvalCellContent>
      </Contents>
    </StructElem>
  </ArrayBody>

</StructArray>
```

Abbildung 50: *Beispiel 2: Hinzufügen von Knoten — unbearbeitetes XML-Dokument*

Die Arbeitsweise der Hilfsfunktion `<StructArray>` ist wie folgt: Das XML-Element `<ArrayElements>` beinhaltet alle Ganzzahlwerte, über die iteriert wird. Für jeden `Integer`-Wert wird dann ein Klon des Inhalts von `<ArrayBody>` erstellt, indem alle Vorkommnisse von `<EvalLoopNumber>` durch den aktuellen Wert des Iterators ersetzt wird. Jeder Inhalt eines Iteratordurchlaufs wird dabei in das XML-Element `<ArrayRow>` gekapselt. Nach dem ersten Durchlauf entsteht folgender Code:

```
<!-- Der gekuerzte Quelltext -->
<StructArray ArrayName="buecher" CacheType="TEMP" LoopName="loop1">

  <ArrayElements>
    <LSIntegerArray>
      <LSInteger>5</LSInteger>
      <LSInteger>7</LSInteger>
    </LSIntegerArray>
  </ArrayElements>

  <ArrayBody>
    <ArrayRow Number="1">
      <StructElem ElemName="buchname" ElemType="LSString">
        <Contents>
          <EvalCellContent TableName="table_db_buecher">
            <ColumnName>buch_name</ColumnName>
            <Row>3</Row>
          </EvalCellContent>
        </Contents>
      </StructElem>
    </ArrayRow>
  </ArrayBody>

</StructArray>
```

Abbildung 51: *Beispiel 2: Hinzufügen von Knoten — nach dem ersten Durchlauf*

Nach dem zweiten Durchlauf wird dann das `<ArrayElement>` 5 abgearbeitet (Abbildung 52):

```
<!-- Der gekuerzte Quelltext -->
<StructArray ArrayName="buecher" CacheType="TEMP" LoopName="loop1">

  <ArrayElements>
    <LSIntegerArray>
      <LSInteger>7</LSInteger>
    </LSIntegerArray>
  </ArrayElements>

  <ArrayBody>
    <ArrayRow Number="1">
      <StructElem ElemName="buchname" ElemType="LSString">
        <Contents>
          <EvalCellContent TableName="table_db_buecher">
            <ColumnName>buch_name</ColumnName>
            <Row>3</Row>
```

```
          </EvalCellContent>
        </Contents>
      </StructElem>
    </ArrayRow>

    <ArrayRow Number="2">
      <StructElem ElemName="buchname" ElemType="LSString">
        <Contents>
          <EvalCellContent TableName="table_db_buecher">
            <ColumnName>buch_name</ColumnName>
            <Row>5</Row>
          </EvalCellContent>
        </Contents>
      </StructElem>
    </ArrayRow>
  </ArrayBody>

</StructArray>
```

Abbildung 52: *Beispiel 2: Hinzufügen von Knoten — nach dem zweiten Durchlauf*

Nachdem dann auch der letzte Knoten abgearbeitet wurde, können `<ArrayElements>` und `<ArrayBody>` entfernt werden, so dass nur noch der Inhalt übrig bleibt (Abbildung 53):

```
<!-- Der gekuerzte Quelltext -->
<StructArray ArrayName="buecher" CacheType="TEMP" LoopName="loop1">

  <ArrayRow Number="1">
    <StructElem ElemName="buchname" ElemType="LSString">
      <Contents>
        <EvalCellContent TableName="table_db_buecher">
          <ColumnName>buch_name</ColumnName>
          <Row>3</Row>
        </EvalCellContent>
      </Contents>
    </StructElem>
  </ArrayRow>

  <ArrayRow Number="2">
    <StructElem ElemName="buchname" ElemType="LSString">
      <Contents>
        <EvalCellContent TableName="table_db_buecher">
          <ColumnName>buch_name</ColumnName>
          <Row>5</Row>
        </EvalCellContent>
      </Contents>
    </StructElem>
  </ArrayRow>

  <ArrayRow Number="3">
    <StructElem ElemName="buchname" ElemType="LSString">
      <Contents>
        <EvalCellContent TableName="table_db_buecher">
          <ColumnName>buch_name</ColumnName>
          <Row>7</Row>
        </EvalCellContent>
      </Contents>
    </StructElem>
  </ArrayRow>

</StructArray>
```

Abbildung 53: *Beispiel 2: Hinzufügen von Knoten — nach dem letzten Durchlauf*

Da nun alle Elemente des Iterators abgearbeitet wurden und entsprechend vielfach geklont vorhanden sind, kann nun die Bottom-Up-Bearbeitung der einzelnen `<ArrayRow>`-Elemente erfolgen.

Anhang C

Glossar

DLO — Document Like Object (dokumentenähnliches Objekt)

Unter dokumentenähnlichen Objekten versteht man im Bereich digitaler Bibliotheken all diejenigen Objekte, die sich wie herkömmliche Dokumente behandeln lassen. Dazu gehört z.B., dass sich eine Volltextsuche auf ihnen durchführen lässt und dass sie nach bestimmten Kriterien katalogisiert werden können. Einen Kernsatz von Metadateninformationen, nach denen dokumentenähnliche Objekte eingeordnet werden können, wurde von der Dublin Core Metadata Initiative (DCMI) erarbeitet.

eNoteHistory (Schreiberidentifikation in historischen Notenhandschriften)

Hierbei handelt es sich um ein Projekt der Universität Rostock, welches eine digitale Bibliothek für die Archivierung von Notenhandschriften entwickelt. In Zusammenarbeit mit einer Bildanalysegruppe wird die Integration eines Notendokumenttyps mit assoziierten, inhaltsbasierten Retrieval-Funktionen erarbeitet. Insbesondere wird die Integration eines Notendokument-Datentyps in ein Content-Management-System vorgenommen. Geplant ist dabei die Einbindung dieses Typs in ein offenes, objektrelationales Datenbanksystem. [ENO02]

Stylesheet

Ein Stylesheet verwendet man, um die Form der Darstellung von den Inhalten eines strukturierten Dokuments (z.B. HTML oder XML) zu trennen. Stylesheets umfassen dabei alle Bereiche der Interpretation (bildliche, hörbare oder fühlbare Darstellung). Stylesheets erlauben es somit, Inhalte abhängig von dem Ausgabegerät (z.B. auch Braille-Lesegeräte für Blinde) zu interpretieren. Dabei muss der Inhalt nicht verändert werden. Dies erlaubt es, den Vorgang des Publizierens oder auch der Betreuung von Inhalten effizient zu gestalten — und ermöglicht auch Arbeitsteilung im höheren Maße, als das z.B. früher bei HTML und eingebetteten Formatierungsbefehlen möglich war. Beispiele für Stylesheet-Sprachen sind: DSSSL (sprich: dissl), XSL, CSS. [WIK04]

Usability (Benutzbarkeit, Bedienungsfreudigkeit, Ergonomie)

Nach ISO-Norm 9241 wird der Begriff Usability folgendermaßen definiert:
„Die Usability eines Produktes ist das Ausmaß, in dem es von einem bestimmten Benutzer verwendet werden kann, um bestimmte Ziele in einem bestimmten Kontext effektiv, effizient und zufriedenstellend zu erreichen."
Eine eindeutige Übersetzung von Usability ist nicht möglich. Am ehesten lässt es sich mit Begriffen wie „Benutzbarkeit", „Bedienungsfreundlichkeit" oder „Ergonomie" gleichsetzen.

Wrapper (Verpackung, Hülle)

Wrapper finden in vielen Bereichen der Informatik ihre Anwendung. Unter der „Hülle" versteht man dabei, dass sie nach außen erwartungskonforme Schnittstellen (Eingabe-, Ausgabeparameter, etc.) zur Verfügung stellen, deren interne Implementierung meistens weitaus komplexer realisiert wird. Sie stellen oft das Bindeglied zwischen unterschiedlichen Konzepten der Informatik dar, z.B. die Nutzung von Routinen einer Programmiersprache innerhalb einer völlig anderen Programmiersprache. So wird beispielsweise dem Java-Programmierer die bekannte Syntax von Java zur Verfügung gestellt, welche intern auf die Syntax von C++ übersetzt wird.

Literaturverzeichnis

[AMA04] Amazon - Online-Versandhandel:
`http://www.amazon.de`,
aufgerufen am 25.01.2004

[BAI03] David Bainbridge, Dana McKay, Ian H. Witten:
„Greenstone Digital Library Developer's Guide“
Department of Computer Science
University of Waikato, New Zealand,
März 2003

[BOE04] Martin Böhm, Jean-Jacques Dubray:
„Dom4J Performance versus Xerces/Xalan“
Eigner Precision Lifecycle Management,
`http://www.dom4j.org/benchmarks/xpath`,
aufgerufen am 19.06.2004

[BRO00] Brockhaus:
„Die Enzyklopädie in 24 Bänden“
Brockhaus, Mannheim,
2000

[BRO02] Brockhaus:
„Der Brockhaus multimedial 2002“
Brockhaus, Mannheim,
2002

[CRO04] Raym Crow:
„A Guide to Institutional Repository Software v2.0“
Open Society Institute,
Januar 2004

[DUB03] Dublin Core Metadata Initiative:
„Mission and Scope of DCMI“
`http://www.dublincore.org`,
aufgerufen am 27.05.2003

[EIC04] Armin Eichinger:
„Usability“
`http://pcptpp030.psychologie.uni-regensburg.de`
`/student2001/skripten/zimmer/usability.html`,
aufgerufen am 13.08.2004

[ENG03] Holm Engelbrecht:
„Gestaltung von Oberflächen - Die Quellen der Entwickler“
Vortrag im Rahmen des Usability-Seminars,
2003

[ENO02] Ilvio Bruder:
„Antrag auf Gewährung einer Sachbeihilfe im Rahmen des DFG-Förderprogramms“
Universität Rostock,
2002

[EPR04] EPrints - Self-Archiving and Open Archives:
`http://www.eprints.org`,
aufgerufen am 14.04.2004

[GAL97] Wilbert O. Galitz:
„The Essential Guide to User Interface Design“
Wiley Computer Publishing,
1997

[GDB99] GDBM - GNU Database Manager:
`http://www.gnu.org/software/gdbm/gdbm.html`,
aufgerufen am 09.08.2004

[GRE04] Greenstone Digital Library Software:
`http://www.greenstone.org`,
aufgerufen am 22.02.2004

[GUL02] Jürgen Gulbins, Markus Seyfried, Hans Strack-Zimmermann:
„Dokumenten Management“
Axel Springer Verlag: Berlin - Heidelberg - New York,
2002

[HEG03] Marcus Hegner:
„Methoden zur Evaluation von Software“
Informationszentrum Sozialwissenschaften, Bonn,
2003

[ITO04] i-Tor - Tools and Technology for Open Repositories:
`http://www.i-tor.org/en`,
aufgerufen am 17.08.2004

[JAV04] SUN Microsystems:
„Java Technology“
`http://java.sun.com`,
aufgerufen am 14.05.2004

[KLE02] Meike Klettke:
„Vorlesung: XML und Datenbanken“
Universität Rostock,
2002

[KOE04] Yves Köth:
„User Interface für ein generisches Modellierungswerkzeug“
Technische Universität Dresden,
2001

[LIN94] Gitte Lindgaard:
„Usability and System Evaluation“
Chapman & Hall, London,
1994

[LUE04] Frank Lützenkirchen, Jens Kupferschmidt, Detlev Degenhardt:
„MyCoRe Starting Guide“
Universität Essen,
Feburar 2004

[MCR04] MyCoRe - Digitale Bibliotheken und Archivlösungen:
`http://www.MyCoRe.de`,
aufgerufen am 25.03.2004

[MAY03] Philip Mayr:
„Dublin Core: Die Renaissance der Bibliothekare“
`http://www.informatik.hu-berlin.de/~mayr/d_c.htm`,
aufgerufen am 17.08.2004

[MIL56] George A. Miller:
„The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information“
`http://www.well.com/user/smalin/miller.html`
März 1956, aufgerufen am 17.08.2004

[MIL68] Robert B. Miller:
„Response Time in Man- Computer Conversational Transactions“
Fall Joint Computer Conference, AFIPS Press
1968

[NEL03] Manja Nelius:
„Mousemap-basierte Erkennung der Problemfelder von Anwendern bei der Bedienung von Software“
Universität Rostock,
2003

[NIE93] Jakob Nielsen:
„Usability Engineering“
Academic Press,
1993

[NIE95] Jakob Nielsen:
„Technology Transfer of Heuristic Evaluation and Usability Inspection“
IFIP INTERACT'95 International Conference on Human-Computer Interaction,
1995

[NIE00] Jakob Nielsen:
„Designing Web Usability: The Practice of Simplicity“
New Riders Publishing,
2000

[NSE04] International Business Machines - IBM:
„DB2 Net Search Extender“
`http://www.ibm.com/software/data/db2/extenders/netsearch`,
aufgerufen am 17.08.2004

[PHP04] PHP - Hypertext Preprocessor:
“PHP Introduction Tutorial„
`http://www.php.net`,
aufgerufen am 17.08.2004

[PER04] O'Reilly:
„The Source for Perl“
`http://www.perl.com`,
aufgerufen am 17.08.2004

[SHN83] Ben Shneiderman:
„Direct Manipulation: A Step beyond programming Languages“
IEEE Computer,
August 1983

[SHN92] Ben Shneiderman:
„Designing the User Interface“
Addison-Wesley Publishing Company,
1992

[SER04] SUN Microsystems:
„Java Servlet Technology“
`http://java.sun.com/products/servlet`,
aufgerufen am 17.08.2004

[SPR04] Springer Online:
„SpringerLink - Online-Volltextarchiv“
Axel Springer Verlag: Berlin - Heidelberg - New York,
`http://www.springerlink.com`,
aufgerufen am 17.08.2004

[UPA03] Usability Professionals' Association:
„What is User-Centered Design?“
`http://www.upassoc.org`,
aufgerufen am 17.08.2004

[W3C04] World Wide Web Consortium (W3C):
`http://www.w3.org`,
aufgerufen am 17.08.2004

[WAT98] Donald J. Waters:
„Digital Library: What Are Digital Libraries?“
CLIR Issues,
1998

[WEB02] Prof. Dr. Michael Weber:
„Vorlesung: User Centered Design“
Universität Ulm, 2002

[WIK04] Wikipedia - Die freie Enzyklopädie:
`http://www.wikipedia.org`,
aufgerufen am 17.08.2004

[WIT99] Ian H. Witten, Alistair Moffat und Timothy C. Bell:
Managing Gigabytes: Compressing and Indexing Documents and Images
Morgan Kaufmann Publishers, Inc.,
`http://www.cs.mu.oz.au/mg`,
1999, aufgerufen am 17.08.2004

[XML04] World Wide Web Consortium:
„Extensible Markup Language (XML)“
`http://www.w3.org/XML`,
aufgerufen am 17.08.2004

[XPA04] World Wide Web Consortium:
„XML Path Language (XPath)“
`http://www.w3.org/TR/xpath`,
aufgerufen am 17.08.2004

[XQU04] World Wide Web Consortium:
„An XML Query Language (XQuery)“
`http://www.w3.org/TR/xquery`,
aufgerufen am 17.08.2004

[XSL04] World Wide Web Consortium:
„Extensible Stylesheet Language Transformations (XSLT)“
`http://www.w3.org/TR/xslt`,
aufgerufen am 17.08.2004

Abbildungsverzeichnis

1 Entwicklungszyklus der benutzerorientierten Gestaltung 12
2 Übersicht der 18 generellen Designprinzipien 13
3 Prioritätentabelle für Standardnutzer 16
4 Prioritätentabelle für Expertennutzer 17
5 Prioritätentabelle für Administratoren 18
6 Komponentenmodell von MyCoRe 20
7 Grobes Schichtenmodell von MyCoRe 21
8 Greenstone-System in Verbindung mit dem „Null-Protokoll“ 24
9 Das Greenstone-Runtime-System 25
10 Greenstone-Archivformat: Document Type Definition (DTD). 25
11 Greenstone-Archivformat: XML-Datei für einen Beispieldatensatz. 26

12 Einordnung von ***LibScens*** in die eNoteHistory-Architektur 33
13 Schnittstellenarchitektur von ***LibScens*** 35
14 Modularer Aufbau eines XML-Szenariobausteins 37
15 Integration der Eingabedaten in die ***LibScens***-Laufzeitumgebung 42
16 Auszug der Strukturobjekte aus dem XML-Schema 46
17 XML-Interface für die komplexe Struktur „LSTable“ 47
18 XML-Interface für die komplexe Struktur „LSTree“ 50
19 Standardrepräsentation einer Tabelle 52
20 Repräsentation einer Tabelle in Listenform 53
21 Repräsentation eines Baums als flache Tabelle 54
22 Repräsentation eines Baums als Microsoft-Explorer-Struktur 54
23 Repräsentation eines Baums fokussiert auf den aktuellen Knoten 55
24 Hierarchische Anordnung der vierschichtigen Nutzerauthentifizierung 57
25 XML-Schema einer vierschichtigen Nutzerauthentifizierung 58

26 Dreistufiger Transformationsprozess 62
27 Package DE.LibScens 63
28 Detaillierter Programmablaufplan von ***LibScens*** 65
29 Schnittstellenbeschreibung für eine Hilfsfunktion bzw. -prozedur 67
30 Dynamisches Einbinden von Klassen durch die Reflection-API 68
31 Rekursive Abarbeitung der DOM-Knoten des XML-Szenariobausteins 68
32 Beispiel für ein grafisches Front-End für Log4J (Chainsaw V2) 70
33 Listing des XSL-Templates LibScensMainTemplate.xsl 72

34 Standardrepräsentation des Basisdatentyps LSText 73

35 Entwurf einer grafischen Nutzerschnittstelle für ***LibScens*** 76
36 Bezug zw. Häufigkeit und eingeschätztem Nutzen der Methoden [NIE95] 80
37 Tabelle der eingesetzten Hard- und Software der Testumgebung 83
38 Prozentuale Zeitaufteilung bei 100 gelesenen Elementen 85
39 Prozentuale Zeitaufteilung bei 300 gelesenen Elementen 85
40 Prozentuale Zeitaufteilung bei 600 gelesenen Elementen 85
41 Auswertung der Bearbeitungszeit der XSLT-Prozessoren 87

42 Beispiel für eine grafische Nutzerschnittstelle von MyCoRe 90
43 Beispiel für eine grafische Nutzerschnittstelle von Greenstone 91
44 Beispiel für eine grafische Nutzerschnittstelle von Eprints 91
45 Beispiel für eine grafische Nutzerschnittstelle von i-Tor 92

46 Beispiel 1: Entfernung von Knoten — unbearbeitetes XML-Dokument 93
47 Beispiel 1: Entfernung von Knoten — nach erster Knotenersetzung 94
48 Beispiel 1: Entfernung von Knoten — nach zweiter Knotenersetzung 94
49 Beispiel 1: Entfernung von Knoten — nach letzter Knotenersetzung 94
50 Beispiel 2: Hinzufügen von Knoten — unbearbeitetes XML-Dokument 94
51 Beispiel 2: Hinzufügen von Knoten — nach dem ersten Durchlauf 95
52 Beispiel 2: Hinzufügen von Knoten — nach dem zweiten Durchlauf 95
53 Beispiel 2: Hinzufügen von Knoten — nach dem letzten Durchlauf 96

Thesen

1. Es ist möglich, eine komplette Szenariobeschreibung in *einem* Dokument zu speichern. Dabei kann es durch vier separate Module beschrieben werden, welche logisch aufeinander aufbauen (Deskriptor, Eingabetabellen, komplexe Strukturen, Seitenlayout).

2. Durch eine Modularisierung der XML-Szenariobausteine können die unterschiedlichen Abschnitte komfortabel serialisiert werden und erleichtern somit die Erstellung grafischer Oberflächen.

3. Die in dieser Arbeit vorgestellte dreistufige Transformation (Datenbank — Eingabetabellen, Eingabetabellen — komplexe Strukturen, komplexe Strukturen — Ausgabeformat) hat sich aus Sicht einer einheitlichen Standardisierung und Normalisierung von divergierenden Datenbanksystemen bewährt.

4. Aus Performance-Sicht hat sich die praktische Umsetzung von XML-Transformationen durch XSLT-Techniken als realisierbar herausgestellt. Die Umsetzung der Szenariobausteine mittels XML hat sich bewährt und wird durch zukünftige Weiterentwicklungen noch leistungsfähiger werden.

5. Durch die dynamischen, zur Laufzeit erstellbaren Implementierungen bzw. Interpretationen der XML-Elemente ist jederzeit eine einfache Wartung und Erweiterung der Hilfsfunktionen bzw. -prozeduren durch den Administrator möglich.

6. Die klassischen Usability-Betrachtungen aus den 50er und 60er Jahren, beispielsweise von George A. Miller und Robert B. Miller, sind derzeit auch noch für den Entwurf und die Umsetzung von digitalen Bibliotheken gültig.

7. Durch den Einsatz einer geschachtelten, vierschichtigen Nutzerauthentifizierung (Gruppe, Nutzer, Szenario, Eingabetabelle) können Sicherheitsbarrieren auf unterschiedlichem Niveau geschaffen werden, welche eine exakte Anpassung an individuelle Bedürfnisse ermöglicht.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 15. September 2004