Diplomarbeit im Studiengang Medieninformatik

XSLT: Anwendung und Einbindung in Applikationen am Beispiel einer Webanwendung zur Darstellung von Sendeplänen.

1. Prüfer: Prof. Dr. Martin Goik
2. Prüfer: Dipl.-Ing. (FH) Wolf-Rüdiger Krenglowski

Vorgelegt von Sebastian Platz

am 28.05.2004

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angeführten Hilfsmittel in Anspruch genommen habe.

_____ _____

Datum Unterschrift

Inhaltsverzeichnis

Zusammenfassung

Diese Arbeit befasst sich mit der eXtensible Stylesheet Language (XSL) bzw. XSL Transformationen (XSLT). Dabei werden die grundsätzlichen Konzepte von XSLT untersucht. Darauf aufbauend werden Konzepte zur effizienten Verwendung von XSLT aufgezeigt.

Der Einsatz von XSLT wird am Beispiel einer Webanwendung vorgestellt, die bei der ProSieben Information Service GmbH in Ismaning / München erstellt wurde und zur Darstellung von Sendeplänen eingesetzt wird. Anhand dieser Anwendung werden unterschiedliche Möglichkeiten untersucht, XSLT mit anderen Technologien zu verbinden.

1. Einleitung

1.1. Überblick

XML hat sich als geräte- und plattformunabhängiges Format zum Austausch von Daten über Unternehmens- und Systemgrenzen hinweg etabliert. Das liegt vor allem daran, dass es dem W3 Konsortium, das für die Spezifikation von XML verantwortlich ist, gelungen ist, ein einfaches Konzept vorzustellen, welches die Trennung von Inhalt und Struktur bei Dokumenten ermöglicht. Dass XML in der Tat weite Verbreitung findet, lässt sich zum einen an den ebenso zahlreichen wie unterschiedlichen Verwendungsformen belegen und zum anderen an den vielen Spezifikationen, die um XML herum entstanden sind und ebenso wie XML selbst einem beständigen Prozess der Weiterentwicklung unterworfen sind. Als aktuelles Beispiel, das die Möglichkeiten von XML basiertem Datenaustausch zeigt, seien hier Web Services genannt, deren Protokoll-Mechanismen auf XML beruhen und die einen automatisierten Austausch von Informationen unterschiedlicher Systeme ermöglichen.

Mit der ausschließlichen Beschreibung logischer Strukturen innerhalb von XML Dokumenten stellt sich die Frage, wie diese Strukturen in einer Form dargestellt werden können, die nicht nur eine maschinelle Lesbarkeit erleichtert, sondern auch für das menschliche Auge leicht zu erfassen ist. Zwar ist XML aufgrund der syntaktischen Eigenschaften prinzipiell gut verständlich, aber vor allem bei XML Dokumenten, die größere Datenmengen enthalten, nimmt die Übersichtlichkeit wieder ab. Diese Problematik ist nicht neu und bereits 1997 wurde vom W3 Konsortium der erste Vorschlag für die eXtensible Stylesheet Language (XSL) veröffentlicht.[1] Stellt XML eine Weiterentwicklung und Vereinfachung von Standard Genralized Markup Language (SGML) dar, so handelt es sich bei XSL um eine ähnliche Vereinfachung der auf SGML aufsetzenden Document Style Semantics and Specification Language (DSSSL), die eine einheitliche Beschreibung der Ausgabe von SGML Dokumenten ermöglichen sollte. SGML und DSSSL waren bereits in den 1980er Jahren entstanden, erwiesen sich allerdings als zu komplex, um in der Praxis tatsächlich umgesetzt zu werden. Im Laufe der letzten Jahre stand die Entwicklung von XSL nicht still, und die Funktionen der Sprache wurden weiter verfeinert. Momentan werden im Bereich der Darstellung von XML Dokumenten Begriffe wie XSL, XSL-FO und XSLT verwendet, welche der Darstellung von XML Dokumenten dienen, jedoch unterschiedliche Teilgebiete abdecken. Hinzu kommt, dass sich diese Formatierungssprachen in vielen Bereichen der Sprache XPath[2] bedienen. XPath ermöglicht die Navigation in XML Dokumenten und wird in einer eigenständigen Spezifikation definiert. Diese Unterscheidungen haben nicht unbedingt dazu beigetragen, die Übersichtlichkeit von XSL und den damit verbundenen Konzepten zu erleichtern. So lässt sich sagen, dass fünf Jahre nach der Veröffentlichung der offiziellen Empfehlung von XSLT 1.0 durch das W3C die Verbreitung von XSLT keineswegs so weit fortgeschritten ist wie die von XML. Ein weiterer Grund für die zögerliche Verbreitung von XSL lässt sich in der Verwendung von Cascading Stylesheets (CSS) im Bereich der Webentwicklung bzw. des Webdesigns finden. CSS ermöglicht ebenfalls eine umfassende Formatierung von XML und HTML Dokumenten und erschließt sich auf den ersten Blick schneller als XSL.

[1] [XSL]
[2] [XPath10]

Dabei bieten sich mit XSLT eine Vielzahl von Möglichkeiten, auch sehr komplexe Transformationen von XML Dokumenten vorzunehmen und damit nicht nur unterschiedliche Ausgabeformate, sondern auch eine umfassende Neustrukturierung der zugrunde liegenden XML Daten zu erzeugen.

1.2. Die Problemstellung

1.2.1. Die ProSieben Information Service GmbH

Die ProSieben Information Service GmbH, welche den thematischen Rahmen für die vorliegende Arbeit gegeben hat, verwendet XML zum Datenaustausch zwischen unterschiedlichen Applikationen und Systemen. Auf das Unternehmen und die konkrete Aufgabestellung wird im Folgenden näher eingegangen.

Die ProSieben Information Service GmbH (PSI) ist ein Tochterunternehmen der ProSiebenSat.1 Media AG zu 100%. Als ganzheitlicher IT-Dienstleister hat sich die PSI auf die Entwicklung von Software für und die Integration von Systemen in die Prozesswelt der Medienbranche spezialisiert. Die entwickelte Software unterstützt den gesamten Workflow innerhalb der Unternehmensgruppe, und umfasst Lösungen für die Fernsehproduktion, den Online-Bereich sowie begleitende Unter–nehmensaktivitäten. Dabei steht vor allem die Umsetzung von Projekten im Bereich Media Asset Management, Vermarktungs- und Planungsprozesse (Medienforschung, Produktionsverfolgung und Erlösplanung) im Vordergrund sowie deren Integration in die bestehende Infrastruktur der ProSiebenSat.1 Media AG. Als zugrunde liegende technologische Plattform kommen Java und die J2EE zum Einsatz; dabei muss jedoch auch die Anbindung an bestehende Altsysteme bei der Entwicklung berücksichtigt werden. Die PSI wurde 1996 gegründet und hat inzwischen einen Teil der Entwicklungsarbeit nach Minsk, Republik Belarus, ausgelagert.

Vorstellung von ProSeco

Die Anforderungen an Programm- und Sendeplanungssysteme innerhalb der ProSiebenSat.1 Media AG haben sich im Laufe der letzten Jahre geändert. Eine stark veränderte Senderlandschaft mit Fenstersendern, Live Sendern und immer individuelleren Programmabläufen verursacht einen erhöhten Planungsaufwand; zudem müssen Planungssysteme flexibler auf kurzfristige Änderungen in der Programm- und Werbeplanung reagieren können. Die derzeit eingesetzten Systeme erfüllen viele dieser Anforderungen nicht, so dass sich im Planungsablauf Schwachstellen ergeben. Dies gilt vor allem bezüglich der Planung von Werbeblöcken, die für die Finanzierung der gesamten Sendergruppe wichtig ist. Aus diesem Grund wurde das Projekt ProSeco ins Leben gerufen, das den Entwurf und die Umsetzung neuer Planungssysteme als Ziel hat, welche zum einem den neuen funktionalen Anforderungen innerhalb der Sendergruppe gerecht werden und zum anderen neu verfügbare Technologien berücksichtigen, so dass ein reibungsloser Planungsablauf auch längerfristig gewährleistet werden kann. Die im Rahmen von ProSeco entwickelten Systeme bilden die Plattform für die Datenhaltung und alle anfallenden Planungsaufgaben im Broadcastbereich - von der strategischen Programmplanung, über die Sendeplanung bis hin zur Sendeabwicklung.

Dabei muss jedoch stets bedacht werden, dass der produktive Betrieb mehrerer Fernsehsender äußerst zeitkritisch ist und so nur eine schrittweise Umstellung auf neue Systeme erfolgen kann. Konkret bedeutet dies, dass bei der Entwicklung neuer technologischer Ansätze stets die Anbindung an bereits bestehende Altsysteme realisiert werden muss.

Vor diesem Hintergrund entstand als Teilprojekt von ProSeco die Komponente SchedEx, deren Funktionalität im Import und Export von Planungsdaten innerhalb von ProSeco besteht. Dazu werden Sendepläne aus unterschiedlichen Systemen in ein einheitliches Format konvertiert, mit dem andere Komponenten und Systeme arbeiten können. Aus technologischer Sicht bietet sich, wie auch schon zuvor beschrieben, XML als geräteunabhängiges Format an.

1.2.2. Aufgabenstellung

Als Bestandteil der SchedEx Komponente soll nun eine Anwendung - der SchedEx Viewer - entwickelt werden, die es unterschiedlichen Benutzergruppen ermöglicht, Sendepläne in unterschiedlichen Versionen (bezüglich der endgültigen Fassung, die die Ausstrahlung der Programmelemente bestimmt) und unterschiedlichen Zeitintervallen[3] zu betrachten und anhand dessen evtl. nötige Korrekturen und Änderungen vorzunehmen.

1.3. Das Ziel der Diplomarbeit

In der vorliegenden Arbeit sollen die grundlegenden Konzepte von XSLT näher untersucht werden und im Anschluss daran nach praktischen Ansätzen zur Verwendung von XSLT speziell im Rahmen der unter 1.2.2 vorgestellten Aufgabestellung gesucht werden. Dabei geht es um die Evaluierung der Möglichkeiten, die sich mit der Verwendung von XSLT bieten sowie deren optimale Anbindung und Nutzung in der bestehenden Infrastruktur, die durch das Gesamtprojekt gestellt wird. Für ein besseres Verständnis des später beschriebenen Lösungsansatzes sowie dessen Umsetzung sollen im anschließenden Kapitel zunächst die Technologien erläutert werden, die den Kern der Anwendung bilden, nämlich XML und XSLT. Im Anschluss daran wird das System analysiert, innerhalb dessen die Anwendung integriert werden muss sowie Anforderungen genannt, die an die Anwendung gestellt werden. Aus den Grundlagen, der Analyse und den Anforderungen wird dann ein Entwurf für das System bzw. die Anwendung abgeleitet, dessen Umsetzung schließlich in Kapitel 5 beschrieben wird. Den Schluss dieser Arbeit bildet das Fazit, in dem nochmals zusammenfassend erläutert werden soll, inwieweit die gestellten Anforderungen an das System konkret umgesetzt werden konnten bzw. welche Schlüsse für die Zukunft der Anwendung und die generelle Verwendung von XSLT gezogen werden können.

1.4. Der Rational Unified Process (RUP)

Innerhalb der PSI kommt zur Optimierung aller Entwicklungsabläufe der Rational Unified Process (RUP) zum Einsatz. In der Terminologie des RUP wird ein Projekt in Workflows mit unterschiedlichen Phasen unterteilt, die wiederum in mehreren Schritten (Iterationen) durchlaufen werden.[4] Diese Arbeit orientiert sich in ihrer Gliederung an den Workflows des RUP; dabei werden allerdings nur diejenigen Workflows berücksichtigt, die bei der Entstehung dieser Arbeit eingesetzt wurden. In jedem Kapitel wird kurz auf die formale Beschreibung der entsprechenden Phase und deren Umsetzung in der PSI eingegangen.

[3] Ein übliches Zeitintervall ist dabei ein Sendetag. Dieser enthält alle Programmelemente eines Tages; dabei ist die Dauer eines Tages allerdings nicht fest auf 24 Stunden beschränkt.
[4] [RUP]

2. Grundlagen

2.1. Überblick

In diesem Kapitel werden die Technologien erläutert, die die Grundlage dieser Diplomarbeit bilden. Zunächst werden die wesentlichen Konzepte der eXtensible Markup Language (XML)[5] erläutert. Diese bildet das Ausgangsformat, in dem die Sendepläne innerhalb der ProSiebenSat1.Media AG vorliegen und ermöglicht so die Anwendung von Transformationen mit Hilfe der eXtensible Stylesheet Language Transformations (XSLT)[6].

Danach werden die Hauptmerkmale von XSLT erklärt, die nötig sind, um die späteren Vorgehensweisen in den Kapiteln 4 und 5 zu verstehen.

2.2. XML

Es soll hier keine detaillierte Beschreibung von XML folgen; dies ist nicht das Thema der Arbeit und ist zudem in entsprechender Fachliteratur nachzulesen. Vielmehr sollen gezielt die Eigenschaften hervorgehoben werden, die für das Verständnis von XSLT wichtig sind, oder die bei der Verarbeitung mittels XSLT besonders zu berücksichtigen sind.

Für die normative und vollständige Spezifikation von XML sei hier auf die entsprechende Vorlage des W3 Konsortiums unter [XML10] verwiesen.

Im Einzelnen werden in diesem Kapitel die folgenden Punkte behandelt:

1. Entstehung
2. Wesentliche Merkmale
3. DTDs und Schemas[7]
4. Namensräume

2.2.1. Entstehung

In den achtziger Jahren des vergangen Jahrhunderts wurde erstmals die Definition einer Standard General Markup Language (SGML) vorgestellt. Das wesentliche Konzept von SGML war, die Trennung von Inhalt und Darstellung in Dokumenten zu erreichen; zudem sollte ein einheitlicher Standard geschaffen werden, der einen konsistenten Einsatz dieses Konzepts über Unternehmensgrenzen, Anwendungen und Plattformen hinweg ermöglicht. Die Verbreitung und die praktische Umsetzung dieses Ansatzes scheiterten allerdings aufgrund der hohen Komplexität von SGML. Die eXtensible Markup Language (XML) entstand als wesentlich vereinfachte Teilmenge von SGML und stellt leicht verständliche Konzepte zur Verfügung, die logische Struktur von Dokumenten zu beschreiben.

[5] [XML10]
[6] [XSLT10]
[7] Diese Arbeit schließt sich der allgemeinen gebräuchlichen Fachterminologie von XML an und verwendet die englische Pluralform "Schemas" anstatt der deutschen, grammatikalisch richtigen, Form "Schemata".

2.2.2. Die wesentlichen Konzepte von XML

XML Dokumente

Im Sprachgebrauch von XML ist ein Dokument ein Datenobjekt, das dem Anspruch der Wohlgeformtheit (siehe dazu 2.2.2) genügen muss. Zudem kann ein XML Dokument noch gültig sein, wenn es bestimmten Einschränkungen entspricht (siehe dazu 2.2.3).

In einem Dokument sind die folgenden Elemente und Konstrukte zu finden:

1. Prolog und Dokumenttyp-Deklaration
2. Markup
3. Zeichendaten
4. Kommentare
5. CDATA-Abschnitte[8]

Ein Dokument besitzt sowohl eine logische als auch eine physische Struktur. In der logischen Struktur wird ein Dokument in Form von Elementen dargestellt. Dabei gibt es unterschiedliche Typen von Elementen, die alle besonders gekennzeichnet sind (durch den Markup) und sich dadurch von den eigentlichen Textdaten des Dokuments unterscheiden.
Der Markup von XML wird definiert als Menge von Start- und End-Tags bzw. leeren Element-Tag sowie Entity-Referenzen, Zeichenreferenzen, Kommentaren, Begrenzungen für CDATA-Abschnitte, Dokumenttyp-Deklarationen, Verarbeitungsanweisungen, XML-Deklarationen, Text-Deklarationen, und jeglichem Leerraum auf oberster Ebene des Dokument-Entity. Text, der nicht unter diese Definition fällt, bildet die Zeichendaten des Dokuments. Der Name eines Elements stellt zugleich den Typ eines Elements dar; zudem können in einem Element auch Attribute, bestehend aus Namen und zugewiesenem Wert, enthalten sein. Auf die Bezeichnung von Elementen wird im Abschnitt 2.2.3. Namensräume nochmals näher eingegangen.

In der physischen Struktur ist ein Dokument in Speichereinheiten unterteilt, die als Entities bezeichnet werden. Ein Entity kann als Mechanismus verstanden werden, der es ermöglicht, eine definierte Menge von Daten innerhalb eines Dokuments unter einem zuvor zugewiesenem Namen zu referenzieren. Ein Dokument kann ein oder mehrere Entities enthalten; allen Entities ist dabei gemein, dass sie einen Inhalt haben und mit Ausnahme des Dokument-Enitity durch einen Namen identifiziert werden. Prinzipiell kann zwischen zwei Arten von Entities unterschieden werden: analysierten (parsed) Entities und nicht-analysierten (unparsed) Entities. Der Inhalt eines analysierten Entity wird als Ersetzungstext bezeichnet und gilt als integraler Bestandteil des Dokuments. Ein nicht-analysiertes Entity ist eine Ressource, deren Inhalt nicht nur Text sein kann (beispielsweise Bilder). Handelt es sich bei dem Inhalt um Text, so muss dieser nicht der XML Spezifikation entsprechen.

[8] CDATA-Abschnitte dürfen überall dort stehen, wo auch Zeichendaten erlaubt sind. Sie dienen dazu, ganze Textblöcke zu schützen, die Zeichen enthalten, die normalerweise als Markup interpretiert würden. CDATA-Abschnitte beginnen mit der Zeichenkette »<![CDATA[« und enden mit »]]>«.

Wohlgeformtheit von XML Dokumenten

Um wohlgeformt zu sein, muss ein Dokument die folgenden Bedingungen erfüllen:

1. Es muss in seiner Gesamtheit betrachtet zu der *document* Produktion[9] passen[10]. Das bedeutet konkret: Ein Dokument enthält ein oder mehrere Elemente und zudem genau ein Wurzel-Element, von dem kein Teil in einem anderen Element enthalten ist. Alle anderen Elemente werden durch Start- und End-Tag begrenzt und müssen korrekt ineinander verschachtelt sein. D.h. befindet sich das Start-Tag eines Elements im Kontext eines anderen Elements, so muss sich auch das End-Tag des ersten Elements im gleichen Kontext befinden.
   Zur Verschachtelung und der Beziehung von Elementen untereinander sei folgende Definition zitiert: *„[...] für jedes Element k, das nicht die Wurzel ist, [existiert] ein anderes Element v [...], so dass sich k im Inhalt von v befindet, aber nicht im Inhalt eines anderes Elements, das sich [ebenfalls] im Inhalt von v befindet. V heißt Vater von k, und k heißt Kind von v.“*[11]
2. Alle Wohlgeformtheitsbeschränkungen der XML Spezifikation müssen eingehalten werden.[12]
3. Ebenso müssen alle analysierten Entities, die entweder direkt im Dokument enthalten sind oder indirekt referenziert werden, wohlgeformt sein.

Gültigkeit von Dokumenten: DTD und Schemas

Mit der Deklaration eines Dokumenttyps mittels einer DTD (Dokumenttyp-Definition) stellt XML einen Mechanismus zur Verfügung, um Beschränkungen der logischen Struktur eines Dokuments festzulegen und die Verwendung von vordefinierten Einheiten innerhalb eines Dokuments zu ermöglichen. Ein Dokument ist dann gültig, wenn es eine solche Typendefinition besitzt und zudem die darin formulierten Beschränkungen einhält. Eine DTD stellt somit eine Grammatik für eine Klasse von Dokumenten dar.

Während der zunehmenden Verbreitung von XML zur Strukturierung von Dokumenten stellten sich zwei wesentliche Nachteile der bisher verwendeten DTDs heraus. Zum einem sind DTDs in einer eigenen Syntax abgefasst, die sich stark von der XML Syntax unterscheidet und die Auswertung von XML Dokumenten erschwert. Zum anderen zeigten sich bei der Prüfung von Dokumenten (sowohl im Sinne eines Text als auch als Transportmittel von Daten) bezüglich zulässiger Wertemengen von bestimmten Elementen recht schnell die Grenzen der DTD: Mehr als eine grobe Definition von Datentypen und deren Überprüfung ist mit einer DTD nicht umzusetzen.

Hier setzt die XML Schema Spezifikation an und bietet wesentliche Verbesserungen. Zunächst sind Schemas in der XML Syntax gehalten und unterstützen vollständig das Namensraum Konzept, das mit zunehmendem Maß an Bedeutung gewinnt und das im anschließenden Abschnitt erläutert wird. Dadurch dass ein XML Vokabular wiederum in XML beschrieben werden kann, wird die Verwendung

[9] Produktion: Dieser Begriff entstammt der Theoretischen Informatik und wird zur Beschreibung der Syntax einer Programmiersprache verwendet. Eine Produktion ist eine der möglichen Kombinationen aus einer gegebenen Grundmenge von Worten bzw. Zeichenketten(*terminal Symbols*) aufgrund der Syntaxregeln.
[10] [XML10]
[11] ebd.
[12] ebd.

von XML und die Strukturbeschreibung von Dokumenten vereinheitlicht und erleichtert. Bezüglich der Definition von Datentypen beseitigt ein Schema die Schwachstellen der DTD dadurch, dass

1. zusätzliche primitive Datentypen (`Byte`, `Date`, `Integer` u. ä.) eingeführt werden
2. Anforderungen unterschieden werden können, die entweder die lexikalischen Repräsentation von Daten betreffen oder ein zugrunde liegendes Information Set verwalten
3. zudem ist Verwendung von benutzerdefinierten Datentypen möglich, beispielsweise solche, die sich von den primitiven Datentypen ableiten und deren Gebrauch weiter einschränken.

Zum Zeitpunkt der Entstehung dieser Arbeit werden in der Praxis sowohl DTDs und als auch Schemas verwendet. Schemas bieten vor allem in den Bereichen, in denen XML für den Transport von Daten verwendet wird, erhebliche Vorteile und gewinnen dort zunehmend an Bedeutung.

2.2.3. XML Namensräume

In Dokumenten, die mehr als ein Vokabular verwenden, können Probleme bezüglich der Erkennung von Elementnamen und damit das Problem der Namenskollision auftreten: Der Namen eines Elements kann mehrfach in unterschiedlichen Vokabularen definiert sein; somit ist bei der gleichzeitigen Verwendung der namentlich identischen Elemente nicht erkennbar, welchem Vokabular das jeweilige Element entstammt und welchem Zweck dieses Element nun wirklich dient.

Namensräume stellen einen einfachen Mechanismus dar, um Element- und Attributnamen in XML Dokumenten eindeutig zu bestimmen. Elemente und Attribute können eine Namensraum-Deklaration enthalten, die ihre Eindeutigkeit durch die Verwendung von IRI (Internationalized Resource Identifier) erhält. Die Verwendung von IRIs stellt eine Neuerung im Namensraum Konzept dar und wird in Version 1.1 spezifiziert.[13] In Version 1.0 sind noch URIs (Uniform Resource Identifier) angegeben[14]. IRIs sollen weltweit eindeutig sein, somit wird durch Verwendung einer weltweit eindeutigen Zuordnung eines Namens an einen Namensraum eine Namenkollision vermieden.

Das Namensraum-Konzept erlaubt die Verwendung von *erweiterten Namen*. Ein erweiterter Name besteht aus einem Namensraum-Namen und einem lokalen Namen. Innerhalb einer IRI Referenz können Zeichen auftreten, die nicht in Namen innerhalb eines XML Dokuments verwendet werden dürfen; zudem kann eine solche IRI Adressangabe sehr lang sein, so dass erweiterte Namen in der Regel nicht direkt verwendet werden, sondern statt dessen ein *qualifizierter Name* oder auch ein so genannten *Präfix* eingesetzt wird. Hat ein Element kein solches Präfix, so wird es automatisch dem Standard Namensraum des Dokuments zugeordnet.

Ein Namensraum wird deklariert durch die Verwendung der reservierten Attribute `xmlns` oder `xmlns:`. Der Wert dieses Attributs muss entweder ein Verweis in Form eines IRI sein oder ein leerer String. Um tatsächliche eindeutige Verweise zu erhalten, wird die Verwendung von URLs (Uniform

[13] [XMLName11]

[14] Ein IRI stellt eine Erweiterung des URI Konzepts dar. Allgemein wird mittels IRI und URI auf eine (nicht notwendigerweise real existierende) Resource in Form einer standardisierten Adressangabe verwiesen. Wurde bei URIs noch eine bestimmte Teilmenge des ASCII Zeichensatzes verwendet, so sollen Adressen in der IRI Form nun mit Hilfe des Unicode Zeichensatzes ausgedrückt werden. Dies ermöglicht die Verwendung von wesentlich mehr Zeichen und soll dem Gedanken eines *world* wide web gerecht werden. Ein IRI kann wiederum zu einem URI konvertiert werden. Momentan ist der Gebrauch von IRIs noch kein offizieller Standard.

Resource Locators) empfohlen. Wird ein Namensraum in einem Element deklariert, so werden alle weiteren enthaltenen Elemente und Attribute diesem Namensraum zugeordnet, es sein denn, sie deklarieren erneut einen (anderen) Namensraum. Dies gilt dann, wenn einem Namensraum kein Präfix zugeordnet wird; bei Verwendung eines Präfixes, werden nur die Elemente dem Namenraum zugewiesen, in deren Namen das Präfix angegeben wird.

Ein Beispiel:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Diese Deklaration findet sich in XSL Stylesheets und deklariert mittels des Verweises auf eine Internetadresse den XSL Namensraum mit dem Präfix xsl. Dies entspricht dem XSL Standard. Unter der angegeben Adresse muss nicht tatsächlich etwas zu finden sein, vielmehr geht es um die eindeutige Zuordnung von Präfix und Namensraum.

2.3. XSLT

Wie in der Einleitung in Kapitel 1 erwähnt, bildet XSLT den wesentlichen Teil der zu erstellenden Anwendung und damit den Schwerpunkt dieser Diplomarbeit. In den folgenden Abschnitten sollen werden zum besseren Verständnis die wesentlichen Konzepte von XSLT kurz beschrieben. Von einer ausführlichen Einführung wird dabei abgesehen; vielmehr soll hier die Sprache XSLT auf die Aspekte untersucht werden, die einen entscheidenden Einfluss auf das Design und die Umsetzung der Anwendung haben. Für weiter reichende Informationen sei hier auf die vollständige Spezifikation des W3 Konsortiums unter [XSLT10] verwiesen.

2.3.1. Begriffsklärung

XSL, XSL-FO, XSLT und XPath

In ähnlicher Weise wie XML einen vereinfachten Ansatz der Konzepte von SGML darstellt, ist XSL als Weiterentwicklung der auf SGML basierenden Document Style Semantics and Specification Language (DSSSL) zu verstehen und verfolgt wie XML als Ziel die Vereinfachung der vorhergehenden Spezifikation. Zunächst wurde aus DSSSL die Extensible Stylesheet Language XSL. Diese war und ist dazu konzipiert, eine Sprache zur Beschreibung von Formatierung und Präsentation von XML Dokumenten am Bildschirm, als Druckausgabe oder als gesprochenes Wort zu schaffen. Mit fortschreitender Entwicklung von XSL stellte sich heraus, dass sich dabei um einen zweistufigen Prozess handelt: zunächst findet die strukturelle Transformation statt, in der Elemente ausgewählt und neu geordnet werden; im Anschluss daran erfolgt der Formatierungsprozess, in dessen Verlauf die Elemente für die Ausgabe auf Papier oder als Pixel auf dem Bildschirm angeordnet werden. Da diese beiden Schritte weitgehend unabhängig voneinander sind, wurde beschlossen, XSL dementsprechend in zwei Bereiche aufzutrennen: XSLT für die Definition der Transformation und XSL bzw. XSL-FO für die Formatierung.

XSL Formatting Objects (XSL-FO) ist wie XSLT in der XML Syntax definiert und behandelt die Teilbereiche einer druckbaren Seite als Formatierungsobjekte mit entsprechend manipulierbaren Eigenschaften, wie etwa der Positionierung innerhalb einer Seite oder Schriftart und -größe. Die hauptsächliche Verwendung von XSL-FO liegt in der Erzeugung von Dokumenten im PDF Format. XSL-FO verwendet einen eigenen Namensraum, dem standardgemäß das Namensraumpräfix `fo` zugewiesen wird. XSL-FO ist im Vergleich zu XSLT wesentlich umfangreicher, da die Formatierung

von Dokumenten für die Druckausgabe sehr komplex ist. Im Rahmen dieser Arbeit wird es dafür verwendet werden, um PDF Dokumente zu erzeugen und wird nochmals näher in Kapitel 5 erläutert.

Innerhalb von XSLT lassen sich nochmals zwei Unterscheidungen feststellen: Zur gezielten Navigation von einzelnen Teilbereichen in einem XML Dokument bedient sich XSLT der Sprache XPath, für die eine eigene Spezifikation existiert und die unabhängig ist von XSLT. Da jedoch XSLT in seiner Funktionalität stark auf den Konzepten von XPath basiert und XPath im Gegensatz beispielsweise zur Zuweisung von Variablen (dieses Konzept ist in XPath nicht vorgesehen) wiederum auf Funktionen von XSLT angewiesen ist, wird in allen folgenden Erläuterungen dieser Arbeit keine weitere Unterscheidung zwischen XSLT und XPath getroffen werden. Beide Technologien werden unter dem Begriff XSLT referenziert.

XSL-FO und XSLT werden in den meisten Fällen und auch in dieser Arbeit kombiniert verwendet, um ein FO-Dokument zu erzeugen, aus dem wiederum die Druckausgabe generiert wird. Im Abschnitt XSLT Prozessoren wird nochmals näher darauf eingegangen werden.

Stylesheets

In dieser Arbeit wird wiederholt der Begriff *Stylesheet* verwendet werden. Im Bereich der Entwicklung von Webanwendungen können damit zwei unterschiedliche Technologien gemeint werden: Zum einen Cascading Stylesheets (CSS) und zum anderen XSLT Stylesheets. Da beide Techniken in den gleichen Bereichen eingesetzt werden, wird hier nochmals kurz auf die Unterschiede eingegangen. CSS und damit inbegriffen CSS2 wird vornehmlich dazu verwendet, um HTML Dokumenten aber auch XML Dokumenten eine endgültige Formatierung zuzuweisen. In CSS kann angegeben werden, wie einzelne Teile eines Dokuments bezüglich gestalterischer Aspekte wie Schriftart, -größe oder –farbe aussehen sollen. Vielmehr kann mit einem CSS allerdings nicht getan werden: es kann weder die eine Änderung der Reihenfolge von Elementen des Quelldokuments vorgenommen werden noch irgendwelche Berechnungen durchgeführt werden. Ebenso kann mittels CSS nicht entschieden werden, welche Teile eines Dokuments angezeigt werden sollen und welche nicht. Die Verwendung von CSS ist meist erst dann sinnvoll, wenn die endgültige Struktur des Dokuments feststeht. Cascading Stylesheets sind einfach zu erstellen und sehr sparsam im Umgang mit den Ressourcen des Rechners, auf dem sie ausgeführt werden. Im Gegensatz dazu bietet XSLT weitere Möglichkeiten. Das macht XSLT komplexer in der Handhabung und vergrößert wesentlich den Bedarf an Rechner-Ressourcen. In der Praxis und auch im Rahmen dieser Arbeit erweist sich die Kombination beider Technologien als sehr sinnvoll. XSLT wird eingesetzt, um die endgültige Struktur eines Dokuments zu erzeugen; im Anschluss können die Eigenschaften des Layouts in CSS festgehalten werden. Dadurch kann innerhalb der Darstellung eine weitere Trennung erreicht werden, die spätere Änderungen und Anpassungen sowohl an der Struktur als auch am Layout eines Dokuments erleichtern. Im weiteren Verlauf dieser Arbeit wird mit der alleinigen Verwendung des Begriffs *Stylesheet* ein XSLT Stylesheet bezeichnet.

2.3.2. XSLT im Zusammenhang

XML Dokumente haben im Wesentlichen zwei Vorteile: Zum einen lassen sich Inhalt und Formatierung unabhängig voneinander darstellen. Zum anderen lässt sich mit Hilfe von XML Dokumenten ein Datenaustausch zwischen unterschiedlichen Systemen durchführen, bei der lediglich das gemeinsame Format, beispielsweise mit Hilfe einer DTD, definiert werden muss. Um

für den Anwender Informationen, die im XML Format vorliegen, anschaulich darzustellen, empfiehlt sich die Transformation dieser Daten in ein anwenderfreundliches Format, wie etwa HTML oder PDF. Damit ist die Hauptaufgabe von XSLT genau diese Transformation.

Die Transformation mittels XSLT gliedert sich in zwei Bereiche:

1. Die strukturelle Transformation. Hier wird XML Datenstruktur in die Struktur umgewandelt, die für Ausgabe gewünscht wird.
2. Der zweite Bereich umfasst die Formatierung, bei der die neu entstanden Struktur im gewünschten Format (beispielsweise HTML oder PDF) ausgegeben wird.

Vor der Entstehung von XSLT wurde eine solche Transformation von eigens dazu entwickelten Anwendungen übernommen werden. Diese Anwendungen griffen dabei in der Regel auf bereits bestehende Application Programming Interfaces (APIs) zu, um Zugriff auf die im XML Dokument enthaltenen Informationen zu bekommen. Dabei existieren zwei wesentliche APIs: die *Simple API for XML* (SAX) und das *Document Object Model* (DOM). Da diese beiden APIs neben XSLT auch weiterhin von Bedeutung sind bzw. die Grundlage der XSLT Prozessoren (siehe Abschnitt 2.3.5) bilden, soll im Folgenden auf die Grundzüge dieser beiden APIs eingegangen werden.

- Der SAX API liegt eine Ereignis-basierte Verarbeitung von XML Dokumenten zugrunde. Dabei feuert der Parser bei jedem Element innerhalb der XML Struktur ein Ereignis (*Event*), das die Anwendung weiter verwerten kann. Dafür werden Handler definiert, in denen die Zuordnung von empfangenem Ereignis und auszuführender Handlung festgelegt wird.
- Bei der Verwendung der DOM API untersucht der Parser das Dokument und baut eine Baumstruktur des Dokuments im Speicher der Anwendung. Die Anwendung kann anschließend auf diese Struktur zugreifen.

Anwendungen, die eine der beiden APIs verwenden, haben in der Regel den Nachteil, dass sie in einer bestimmten Reihenfolge auf das XML Dokument bzw. die vom Parser produzierte Struktur zugreifen müssen, um an die Informationen zu gelangen. Das bedeutet: ändert sich die Struktur des XML Dokuments, so müssen große Teile der Anwendung neu geschrieben werden, die die neue Struktur berücksichtigten. Die Anwendung ist also sehr eng an die Struktur der XML Dokumente gekoppelt.

Der Ansatz von XSLT ist nun der, die Schritte der Transformation, die allen Anwendungen gemeinsam sind, in einem Satz von Regeln zu definieren und diese Regeln in einer abstrakten, deklarativen Sprache zugänglich zu machen. Diese Regeln basieren auf der Definition, welche Ausgabe entstehen soll, wenn ein bestimmtes Muster im XML Dokument vorkommt. Mit XSLT wird die gewünschte Transformation beschrieben, die effiziente Umsetzung derselben ist abhängig vom XSLT Prozessor. Bei einem XSLT Prozessor handelt sich um eine Anwendung, die oftmals auf die SAX oder DOM API zugreift, um aus dem XML Dokument eine Baumstruktur zu generieren. Eine ausführlichere Betrachtung von XSLT Prozessoren ist im Abschnitt 2.3.5 zu finden. XSLT greift nun auf diese Baumstruktur zu und nicht auf das eigentliche XML Dokument. Jeder Bestandteil des XML Dokuments (Elemente, Attribute, Verarbeitungsanweisungen, …) wird als Knoten innerhalb der Baumstruktur dargestellt und kann separat angesprochen werden. Damit kann der Bereich der Neustrukturierung und Formatierung von XML Dokumenten von der darauf zugreifenden

Anwendung teilweise entkoppelt werden, was eine deutlich erhöhte Flexibilität im Bezug auf Änderungen der XML Struktur bedeutet.

2.3.3. Wesentliche Merkmale von XSLT

Elemente

Das primäre syntaktische Merkmal von XSLT ist die Verwendung der XML Syntax. Es handelt sich also um eine deklararive Sprache, die in Form von Elementen ausgedrückt wird. Standardgemäß werden dabei erweiterte Namen bei Elementen verwendet, die dem xsl Namensraum zugeordnet sind. Die Verwendung der XML Syntax hat den Vorteil, dass damit die Formulierung von Formatierungsregeln in XSLT erleichtert wird. Da mit XSLT jedoch auch komplexere Berechnungen angestellt werden können, kann dies allerdings dann als Nachteil angesehen werden, wenn XSLT im Sinne einer Programmiersprache verwendet werden soll. Eine einfache Zuordnung, etwa `y = f(x)`, sieht in der XML Syntax von XSLT in etwa so aus:

```
01: <xsl:variable name="y">
02:   <xsl:call-template name="f">
03:     <xsl:with-param name="x"/>
04:   </xsl:call-template>
05: </xsl:variable>
```

Listing 2-1

Als Begründung für die Verwendung der XML Syntax lassen die folgenden Argumente anführen:

- Da in Browsern bereits ein XML Parser eingebaut ist, kann dieser für XSL wieder verwendet werden.
- Die syntaktische Inkonsistenz zwischen HTML, XML und CSS sollte vermieden werden.
- Die an Lisp angelehnte Syntax von DSSSL wurde als eine der Hauptursachen für das Scheitern des Standards angesehen. Die XML Syntax sollte der HTML gewöhnten Web Community entgegen kommen.
- In vielen populären Template-Sprachen wie etwa ASP oder JSP werden Instruktionen innerhalb von Elementen eingebettet. Das Konzept konnte als bekannt vorausgesetzt werden.
- Da das Ergebnis einer Transformation durchaus ein neues XSL Stylesheet sein kann, ist es erforderlich, dass ein Stylesheet auf ein anderes zugreifen kann. Das wird durch die Verwendung der XML Syntax gewährleistet.
- Die relative umständliche Syntax, allen voran der Gebrauch von spitzen Klammern, lässt sich erheblich erleichtern durch die Verwendung von WYSIWYG Entwicklungswerkzeugen.

Ausdrücke

Bei den in XSLT verwendeten Ausdrücken (*Expressions*) zur Navigation innerhalb der Struktur eines Dokuments handelt es sich um XPath Ausrücke. Mit diesen lassen sich über so genanten Lokalisierungspfade Angaben machen, welche Elemente an welcher Stelle des XML Dokuments betrachtet werden sollen. Ein Lokalisierungspfad kann in mehrere Lokalisierungsschritte unterteilt sein, die durch das Zeichen "/" voneinander getrennt werden. Damit ähneln sie Pfadangaben, wie sie in UNIX Dateisystemen oder URLs verwendet werden. Stellt man sich ein XML Dokument als Baumstruktur vor, so lassen sich die Pfadangaben leicht nachvollziehen. Ein Lokalisierungsschritt

liefert als Ergebnis eine Menge von *Knoten*[15] des XML Dokuments. Diese Menge kann durch die Verwendung von Prädikaten weiter gefiltert werden. Prädikate liefern Aussagen über die Eigenschaften von Objekten, die entweder wahr oder falsch sein können, und werden in XSLT in eckige Klammern gesetzt. Als Beispiel sei die Verwendung eines Lokalisierungspfades im Element `<xsl:apply-templates>` betrachtet:

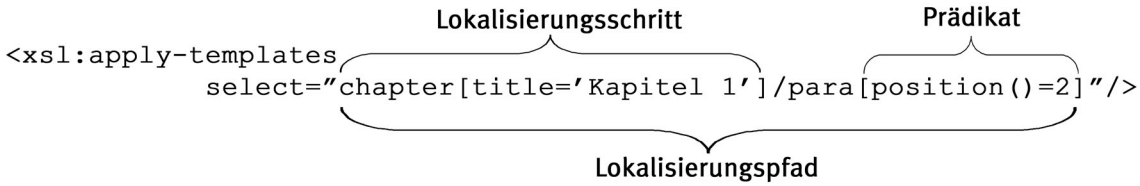

Abbildung 2-1 XSLT Ausdruck

Mit dieser Anweisung wird zunächst das `chapter` Element ausgewählt, dessen Kind-Element `title` den Textknoten mit dem Wert "`Kapitel 1`" besitzt. Ist ein solches Element vorhanden, wird dessen `para` Kind-Element ausgewählt, das sich an zweiter Stelle bezüglich der Reihenfolge im XML Dokument befindet.

Muster

Muster dienen in XSLT dazu, eine Bedingung zu formulieren, der ein Knoten in der Datenstruktur des Ausgangsdokuments genügen muss, um vom XSLT Prozessor zur Weiterverarbeitung und zur Anwendung einer Stylesheet-Regel ausgewählt zu werden. Am häufigsten wird dieser Mechanismus im Element `<xsl:template>` mit dem Attribute `match` verwendet; hier gibt das Muster an, für welchen Knoten die Regel angewandt werden soll. Zudem werden Muster in den Elementen `<xsl:key>` (mittels `match`) und `<xsl:count>` (mittels `count` und `from`) verwendet. Muster und Ausdrücke sind in äußeren Form sehr ähnlich, und in der Tat ist jedes Muster ein gültiger Ausdruck in XPath; umgekehrt hingegen ist nicht jeder Ausdruck ein gültiges Muster. In [Kay01] wird ein Muster formal wie folgt definiert: "*Der Knoten N stimmt mit einem Muster P überein, wenn ein Knoten A existiert, der entweder N vorausgeht oder selbst N ist, so dass die Auswertung von P als Ausdruck - mit A als Kontextknoten - eine Menge von Knoten liefert, die N enthält.*"[16]

Funktionen

In XSLT gibt es vordefinierte Funktionen, die unterschiedliche Bereiche abdecken und beispielsweise Funktionalitäten zur Manipulation von Strings oder einfache Berechnungen zur Verfügung stellen. Ein Großteil davon entstammt der XPath Bibliothek. Allen diesen Funktionen ist gemein, dass sie im Default Namensraum verwendet werden, d.h. sie benötigen kein besonderes Präfix zur Kennzeichnung.

Um die Fähigkeiten von XSLT Stylesheets zu erweitern, besteht in XSLT die Möglichkeit, eigene Funktionen in einer anderen Programmiersprache zu definieren und diese im Stylesheet zur

[15] Knoten: Diese Bezeichnung setzt die Annahme voraus, dass während einer Transformation ein XML Dokument in Form einer Baumstruktur vorliegt, in der die Elemente in Form von Knoten repräsentiert werden. Siehe dazu Abschnitt 2.3.4
[16] vgl [KAY02], S. 431

Verfügung zu stellen, so dass der Prozessor darauf zugreifen kann. So ist es beispielsweise möglich, auf Funktionen der Standard Java Bibliotheken innerhalb eines Stylesheets zuzugreifen. Dies bietet sich an, wenn der verwendete Prozessor ohnehin schon in Java geschrieben wurde. Die Einbindung dieser Funktionen wird allerdings erst in der Spezifikation von XSLT 1.1 definiert; diese Spezifikation hat momentan noch den Status eines Entwurfs, so dass bis zu endgültigen Version durchaus noch Änderungen denkbar sind. Zudem ist die Unterstützung von erweiterten Funktionen immer noch abhängig von der Implementierung des verwendeten XSLT Prozessors. Aus diesen Gründen soll im Rahmen dieser Arbeit nicht weiter auf dieses Thema eingegangen werden.

Regelbasiert

Die Abarbeitung eines XSLT Stylesheets erfolgt vornehmlich über im Stylesheet definierte Regeln; diese werden in Form von Templates (Vorlage) festgehalten. Innerhalb dieser Templates werden die Schritte definiert, die der Prozessor ausführen soll, wenn er auf ein entsprechendes Element im XML Dokument stößt. Dabei ist die Reihenfolge, in der diese Regeln abgearbeitet werden prinzipiell bedeutungslos (in der praktischen Umsetzung und bei der Neustrukturierung von XML Dokumenten kann die Reihenfolge allerdings sehr wohl von Bedeutung sein.) Diese Eigenschaft macht XSLT zu einer deklarativen Sprache im Vergleich zu prozeduralen Sprachen, in denen die sequentielle Abarbeitung einzelner Rechenschritte ausschlaggebend ist.

Prinzipiell wird eine Regel in einem Stylesheet mit dem Element `<xsl:template>` und dem Attribut `match` dargestellt. Der Wert des Attributs `match` ist ein Muster, das angibt, für welche Elemente innerhalb des XML Dokuments diese Regel angewandt werden soll. Wird in einem Stylesheet beispielsweise die Regel `<xsl:template match="chapter">` definiert, so betrifft diese Regel alle `<chapter>` Elemente in der XML Quelle.

Der Inhalt einer solchen Regel besteht aus weiteren Elementen und/oder Textknoten. Kommentare und Verarbeitungsanweisungen im XSL Stylesheet werden ignoriert. Je nach Namensraum der enthaltenen Elemente handelt es sich um Anweisungen oder um Daten. Textknoten sind stets Daten. Bei der Instantiierung einer Template Regel werden die Anweisungen ausgeführt und die Daten in den Ergebnisbaum kopiert.

Keine Nebeneffekte

Unter einem Nebeneffekt versteht man in Zusammenhang mit Programmiersprachen "*das Verhalten eines Ausdrucks, auch ohne explizite Zuweisung die Inhalte von Variablen zu verändern. Meist sind diese Nebeneffekte erwünscht, wie etwa bei der Verwendung der Inkrement- und Dekrement–operatoren.*"[17]

Bereits in den ersten Entwürfen von XSL wird erwähnt, dass XSL eine deklarative Sprache ohne Nebeneffekte sein soll[18]. Das liegt vor allem daran, dass eine Sprache mit Nebeneffekten die Festlegung einer Reihenfolge erfordert, in der die einzelnen Anweisungen abgearbeitet werden. Nebeneffekte stellen in heutigen (objektorientierten) Programmiersprachen wie Java ein wesentliches und häufig verwendetes Merkmal. Die Programmierung in XSLT kann in diesem Zusammenhang mit dem Schreiben funktionaler Programme verglichen werden.

[17] [KRU02], S. 107
[18] [KAY02], S. 37

Die Idee hinter einem funktionalen Programm ist, die Ausgabe als Funktion der Eingabe zu definieren. Bei XSLT handelt sich um eine Sprache zur Beschreibung von Transformationen; es soll also ein Eingabedokument in ein Ausgabedokument transformieren. Somit lässt sich ein Stylesheet als Funktion betrachten, dass diese Transformation beschreibt. Formal ließe sich dies wie folgt ausdrücken:

$A = S(E)$ mit E für Eingabe, A für Ausgabe und S für Stylesheet.

Die Eigenschaft einer solchen reinen Funktion ist, dass sie beliebig oft in beliebiger Reihenfolge ausgeführt werden kann und immer dieselben Werte liefert, also keinerlei Nebeneffekte hat. Im Gegensatz ist die Reihenfolge und Häufigkeit, in der einer Variablen ein Wert zugewiesen wird von essentieller Bedeutung. Der wesentliche Grund für den Entwurf von XSLT in Anlehnung an eine funktionale Programmiersprache ist der, dass Transformation inkrementell ausgeführt werden können: Wenn sich ein Teil eines großes Datenbestandes ändert, so soll bei der Darstellung dieser Daten im Idealfall nur dieser Bereich neu dargestellt werden, ohne dass die Transformation aller Daten erneut erfolgen muss. Theoretisch ließe sich auf diese Weise die Transformation zur Darstellung von Daten auch schon dann starten, wenn nur ein Teil dieser Daten vorliegt (etwa beim Herunterladen eines großen XML Dokuments über ein Netzwerk). Dies ist allerdings bisher keinem verfügbaren XSLT Prozessor möglich; hier wird stets eine vollständige Datenquelle vorausgesetzt. Jedoch war dies eine wesentliche Anforderung an XSLT bei der Konzeption der Sprache, und die technologische Umsetzung ist in Zukunft durchaus möglich.

Als direkte Konsequenz ergibt sich, dass innerhalb von XSLT keine mehrfache Wertzuweisung von Variablen möglich ist. In Abschnitt 2.3.6 Entwurfsmuster wird noch weiter auf dieses Thema eingegangen und Ansätze bzw. eine Art von Best Practises aufgezeigt, mittels derer in XSLT auch komplexere Funktionen programmiert werden können.

2.3.4. Das Datenmodell von XSLT

Betrachtet man ein einfaches XML Dokument, so lässt sich die Darstellung desselben in einer Baumstruktur leicht nachvollziehen: Jedes Textelement kann als einzelnes Blatt angesehen werden, während jedes Element einen weiteren Knoten innerhalb des Baumes darstellt. Das oberste Element in dieser hierarchischen Struktur wird als Wurzel bezeichnet. Im Bezug zum zugrunde liegenden XML Dokument hat es keine inhaltliche Bedeutung; es stellt vielmehr das Dokument als Ganzes dar. Analog dazu wird die Struktur von XML Dokumenten in XSLT als Baummodell dargestellt, mittels dessen jedes wohlgeformte XML Dokument dargestellt werden kann. In einem wohlgeformten XML Dokument enthält das einzige Wurzelelement alle andere Textknoten sowie alle anderen Elemente; diesem Element dürfen lediglich Verarbeitungsanweisungen und Kommentare vorausgehen, nicht jedoch andere Elemente oder Textknoten. In dem in XSLT verwendeten Datenmodell kann das Wurzelelement jedoch alle Kindknoten enthalten, die jedes andere Element auch enthalten kann. Demnach ist das XSLT Datenmodell nicht wohlgeformt; [Kayo1] verwendet hier stattdessen den Begriff well balanced, da lediglich die Anforderung besteht, dass ein Start-Tag eines Elements auch ein entsprechendes End-Tag aufweisen muss, also korrekt ineinander verschachtelt sein müssen. Damit lassen sich nicht nur vollständige XML Dokumente mit XSLT bearbeiten, sondern auch Fragmente davon.

Ein Knoten im XSLT Datenmodell besitzt drei Eigenschaften:

1. Einen Typ oder die Art des Knotens, die die Zusammenfassung von Knoten mit ähnlichen Eigenschaften ermöglicht,
2. einen Namen und
3. den jeweiligen Wert, den ein Knoten enthält.

Innerhalb dieses Modells können sieben verschiedene Arten von Knoten unterschieden werden. Diese entsprechen mehr oder weniger direkt den Strukturen eines XML Dokuments.

1. Wurzel
   Jedes Dokument hat genau einen Wurzelknoten; dieser ist sozusagen die Repräsentation des gesamten Dokuments.
2. Element
   Als Element wird ein Teil eines Dokuments bezeichnet, der von Start- und End-Tag umgeben ist, bzw. ein in sich geschlossenes Element.
3. Text
   Ein Textknoten ist die Aufeinanderfolge von Buchstaben innerhalb des Teils eines Elements, der in XML als PCDATA deklariert ist. Mehrere aufeinander folgende Textknoten werden in XSLT zu einem Knoten zusammengefasst.
4. Attribut
   Attributknoten umfassen sowohl den Namen als auch den zugewiesenen Wert des Attributs und stehen entweder im Start-Tag eines Elements oder in einem leeren Element-Tag. Attribute, die in der DTD mit einem Standardwert belegt sind, jedoch in einem einzelnen Element nicht vorkommen, werden ebenfalls jedem Element zugeordnet. Das Namensraum-Attribut `xmlns` wird nicht durch einen Attributknoten repräsentiert.
5. Kommentar
   Ein Kommentarknoten enthält den Text, der im XML Dokument zwischen den Zeichen `<!--` und `-->` steht.
6. Verarbeitungsanweisung
   Verarbeitungsanweisungen stehen zwischen den Zeichen `<?` und `?>`. Das Ziel der Anweisung, das im XML Quelldokument angeben wird, wird als Namen des Knotens verwendet, der restliche Inhalt als zugewiesener Wert.
7. Namensraum
   Ein solcher Knoten repräsentiert die Deklaration eines Namensraumes. Für jeden Namensraum, der innerhalb des Bereichs eines Elements deklariert wird, wird diesem Element ein entsprechender Namensraum-Knoten zugewiesen.

Es gibt nun mehrere Möglichkeiten, diese Knoten zu klassifizieren, beispielsweise, ob sie Kindknoten enthalten, Elternknoten besitzen oder Text enthalten. Um aber eine flache Hierarchie im Datenmodell von XSLT zu erhalten, werden einem Knoten verschieden Eigenschaften zugeordnet. Besitzt ein Knoten für eine Eigenschaft keinen konkreten Wert, so wird dieser ein leerer Wert zugewiesen. Dabei ist zu beachten, dass in XSLT keine `null` Werte wie etwa in Java vorkommen. Ein leerer Wert besteht entweder aus einem leeren String mit der Länge `0` oder im Fall einer Knotenmenge aus einer leeren Menge.

Bei einer XSLT Transformation wird also nicht direkt auf ein XML Dokument zugegriffen, sondern vielmehr auf eine neue Datenstruktur. Bei der Erzeugung dieser Struktur findet bereits eine Bewertung der Informationen bezüglich deren Bedeutung für eine Transformation statt, da nicht alle Daten, die im XML Dokument ursprünglich enthalten waren, in die Baumstruktur übernommen werden. Da diese Bewertung nicht unwesentlich ist für die allgemeine Verwendung von XSLT, sollen im Folgenden einige Bemerkungen dazu gemacht werden.

Die Auswertung von Informationen in XML Dokumenten

Der große Vorteil von XML ist, dass damit ein de-facto Standard[19] geschaffen wurde, der dem einzelnen Anwender immer noch genügend Freiheiten lässt, Daten und Dokumente seinen Bedürfnissen entsprechend zu definieren. Bei der Auswertung von Informationen, die unterschiedlichen Dokumenten entstammen können, stellt sich recht schnell die Frage, ob dabei alle Informationen die gleiche Gewichtung hinsichtlich ihrer Bedeutung erfahren sollen, oder ob bestimmte Informationen vernachlässigt werden können. Prinzipiell können die Eigenschaften eines XML Dokuments in drei Kategorien unterteilt werden: definitiv signifikant, definitiv unbedeutend und nicht fest definierbar. Beispielsweise ist die Reihenfolge der Elemente definitiv von Bedeutung, die Reihenfolge der Attribute in einem Element eindeutig unbedeutend, wohingegen die Bedeutung von Kommentaren durchaus unterschiedlich gewichtet werden kann.

Der XML Standard definiert diese Unterscheidungen nicht besonders deutlich. Es wird festgelegt, dass bestimmte Eigenschaften eines XML Dokuments einer Anwendung mitgeteilt werden müssen; diese können als bedeutend angesehen werden. Andere Eigenschaften, etwa die Reihenfolge der Elemente bleiben gänzlich unerwähnt. Aufgrund der inzwischen zahlreichen Standards und Spezifikationen, die alle auf XML basieren, und der fehlenden Definition eines eindeutigen Datenmodells bestehen nun verschiedene Auffassungen davon, welche Informationen innerhalb eines XML Dokuments von Bedeutung sind und welche vernachlässigt werden können.

Um diese Fragen in einem einheitlichen Ansatz zu lösen, wurde vom W3C eine weitere Activity Group ins Leben gerufen, die mit der Spezifikation des XML Information Set ein einheitliches Modell der in einem XML Dokument enthaltenen Informationen vorstellen soll. Diese Spezifikation ist einem steten Wandel unterzogen. In der zum Zeitpunkt der Entstehung dieser Arbeit aktuellen Version behandelt die Spezifikation elf Typen von Informationen, die in der Menge der Informationen enthalten sein können, und nennt einige andere Informationen, die nicht in dieser Menge enthalten sind. Die Schwierigkeit, einen einheitlichen Standard festzulegen, lässt sich an der Tatsache erkennen, dass die Spezifikation lediglich Empfehlungen für eine einheitliche Behandlung von Informationen gibt. Unter dem Absatz Konformität wird allerdings darauf hingewiesen, dass andere Spezifikationen, die die Informationsmenge eines XML Dokuments anders definieren, die Unterschiede dementsprechend kenntlich machen müssen.[20]

Ein weiterer Ansatz, eine einheitliche Definition der wesentlichen Informationen zu schaffen, ist die Spezifikation des Canonical XML unter [XMLCan]. Hier wird versucht, Regeln zu definieren, um zu

[19] Ein Standard kann nach internationalen Übereinkünften nur von Körperschaften verabschiedet werden, die von einer Regierung anerkannt wird. Das Dokument mit der höchsten normativen Bedeutung, das vom W3 Konsortium veröffentlicht werden kann, ist die *Empfehlung (Recommendation)*. Durch den normativen Charakter einer Empfehlung und die inzwischen weit verbreitete Verwendung von XML kann es als Standard betrachtet werden.
[20] [XMLInfo10]

entscheiden, wann zwei lexikalisch unterschiedlichen Dokumente den identischen Informationsgehalt haben. Dafür wird eine Transformation definiert, die auf jedes XML Dokument angewandt werden kann, um es in einheitliche (kanonische) Form zu bringen. Ist die die kanonische Form zweier Dokumente gleich, so wird auch der Informationsgehalt als gleich angesehen.

Für die Verarbeitung von XML Dokument mit Hilfe von XSLT bedeutet dies:

- Es gibt mehrere Klassifikationen von Informationen bezüglich ihrer Bedeutung.
- Es gibt wiederum mindestens zwei Ansätze, diese Klassifikationen zu definieren.
- In XSLT wird wiederum einer Ansatz definiert, in dem bestimmt wird, welche Informationen im Datenmodell enthalten sein müssen und welche nicht. Im vorhergehenden Abschnitt sind die sieben Knotentypen benannt, die im Datenmodell enthalten sind. Diese entsprechen nicht vollständig den Strukturen eines XML Dokuments.

Somit kann nicht davon ausgegangen werden, dass auf alle Informationen eines XML Dokuments während einer Transformation zugegriffen werden kann. Dies ist unter anderem abhängig davon, welchem Ansatz der verwendete XSLT Prozessor folgt und welche Daten er somit für die Weiterverarbeitung zur Verfügung stellt. In den meisten Fällen, in den XSLT zum Einsatz kommt, fallen diese Unterschiede allerdings nicht weiter ins Gewicht, da sie nicht den wesentlichen Inhalt eines XML Dokuments betreffen.

2.3.5. XSLT Prozessoren

Die Aufgabe eines XSLT Prozessors ist, eine Transformation[21], die in einem oder mehreren XSLT Stylesheets beschrieben ist, auf XML Daten anzuwenden und als Ergebnis neue XML Datenstrukturen[22] zu generieren. Allgemeinen Regeln der Konformität zufolge arbeitet ein Prozessor nicht direkt mit XML Dokumenten, sondern mit einer Baumstruktur dieser Daten. Somit muss ein XSLT Prozessor nicht notwendiger Weise in der Lage sein, XML Dokumente in eine solche Baumstruktur umzusetzen (was als *Parsen* bezeichnet wird) oder umgekehrt aus einer solchen Baumstruktur erneut ein XML Dokument zu erzeugen (allgemein als *Serialisierung* bezeichnet). In der Praxis sind diese Funktionalitäten allerdings in nahezu allen Implementationen inbegriffen.

Bei der Baumstruktur handelt es sich um einen abstrakten Datentyp; für ihn existiert keine einheitliche Definition, etwa in Form einer API. Allerdings lässt sich sagen, dass diese Baumstruktur im Allgemeinen jener ähnelt, die von W3C in der DOM API festgelegt ist. Manche implementierten XSLT Prozessoren verwenden auch tatsächlich die DOM API; andere hingegen verwenden eine Datenstruktur, die sich an der des XPath Datenmodels orientiert oder aber gänzlich eigene Datenstrukturen.

Der Ablauf einer Transformation lässt sich wie folgt darstellen:

[21] Man spricht bei XSLT von einer Transformation, da der Typ des Input-Dokuments identisch ist mit dem Typ des Ergebnis-Dokuments. Als Folge dessen kann eine komplexe Transformation in eine Abfolge mehrerer einfacher Transformationen zerlegt werden.
[22] In XSLT 1.1 kann das Ergebnis einer Transformation kann über mehrere Kanäle ausgegeben werden. Allerdings lassen sich mehrere Ausgabedokumente auch mit einigen XSLT 1.0 Produkten umsetzen. Vgl. dazu [KAY01], S. 55.

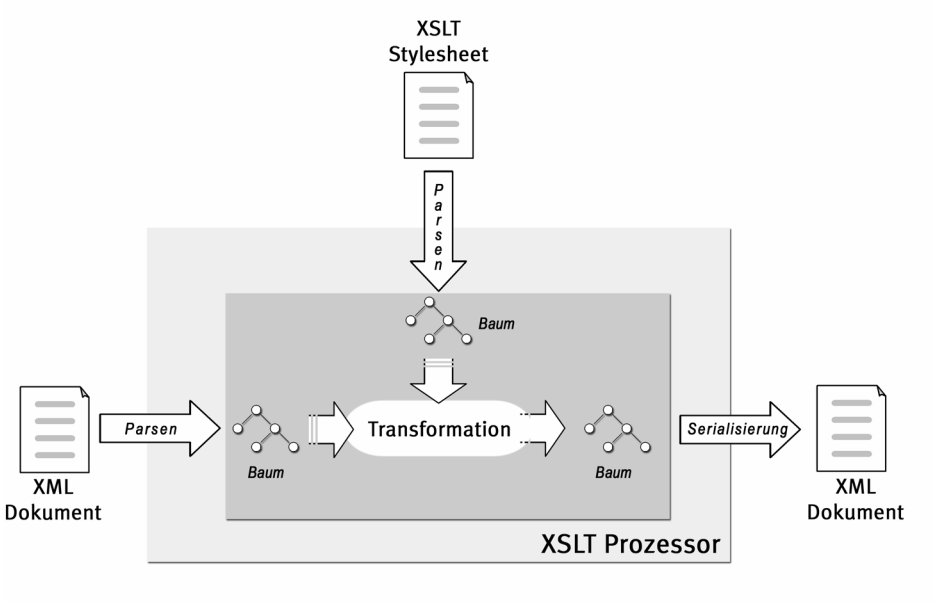

Abbildung 2-2 Funktionsweise von XSLT Prozessoren

Der hellgrau hinterlegte Bereich in Abbildung 2-2 umfasst die Funktionalitäten (das Parsen und die Serialisierung) eines Prozessors, die nicht notwendigerweise implementiert sein müssen.

Mit XSLT lassen sich mit Hilfe des `<xsl:output>` Elements nähere Angaben zu dem Format des Ausgabe-Dokuments machen. Innerhalb dieses Elements können zwischen drei Formaten bzw. Methoden unterschieden werden:

1. XML: Ausgabe Dokument ist ein XML Dokument; dabei kann es sich auch nur um einen Teil eines Dokuments handeln.
2. HTML: Ausgabe ist ein HTML Dokument oder ein Teil davon.
3. Text: Hiermit sind alle textbasierten Formate gemeint. Als häufig verwendete Formate seien hier das Rich Text Format (RTF) von Microsoft oder das Portable Definition Format (PDF) von Adobe genannt.

Die Angabe des Ausgabeformats ist nicht zwingend erforderlich; beim Fehlen dieses Elements versucht ein XSLT Prozessor zu erkennen, um welches Format es sich handeln könnte und wählt im Zweifelsfall XML als Ausgabeformat.

Verfügbare XSLT Prozessoren

Im Folgenden sollen zwei wesentlichen Implementationen vorgestellt werden, die unter der Open Source Linzenz zur freien Verwendung verfügbar sind und im Rahmen dieser Arbeit zum Einsatz kommen.

- Xalan
  Xalan ist ein Prozessor, der von der Apache Organisation produziert wird und unter [XALAN] verfügbar ist. Dieser Prozessor ist als Implementation sowohl in Java als auch in C++ erhältlich. Als Parser wird die Apache eigene Implementation Xerces verwendet, allerdings kann auch jeder andere JAXP konforme Parser eingebunden werden. Xalan findet allgemein häufige Verwendung und unterstütz nach Angaben unter [XALAN] vollständig XSLT 1.0 und XPath 1.0

- Saxon
  Saxon ist ein XSLT Prozessor, der von Michael Kay produziert wird und unter [SAXON] unter der Mozilla Public License verfügbar ist. Saxon ist in Java implementiert und ist als vollständige Version inklusive SourceCode oder als Instant Version, die über die Kommandozeile gestartet wird, erhältlich. Neben der Unterstützung von XSLT 1.0 und XPath 1.0 sind zudem Teile der XSLT 1.1 Spezifikation umgesetzt.

FO Prozessoren

XSL-FO Dokumente werden in erster Linie dazu verwendet, PDF Dokumente zu erzeugen. Dies geschieht mit Hilfe von FO Prozessoren, die von unterschiedlichen Herstellern angeboten werden und teilweise frei verfügbar sind. Generell lässt sich feststellen, dass aufgrund der hohen Komplexität der XSL-FO Spezifikation alle nicht darin enthaltenen Elemente und Eigenschaften in einem FO Prozessor umgesetzt sind. Im Folgenden soll der auch in dieser Arbeit eingesetzte Prozessor fop[23] kurz vorgestellt werden.

- fop
  fop wird wie Xalan von der Apache Gruppe unter [FOP] angeboten; Xalan wird auch als Standard XSLT Prozessor von fop genutzt. Das hat den Vorteil, dass mit fop direkt XSLT Stylesheets, die entsprechende XSL-FO Anweisungen enthalten, verarbeitet werden können. Dabei funktioniert fop ähnlich wie in Abbildung 2-2 gezeigt, nur dass in einem weiteren Zwischenschritt implizit aus einem XSLT Dokument ein FO-Dokument bzw. eine entsprechende Datenstruktur generiert wird, aus welcher schließlich ein PDF erzeugt werden kann. Auch in fop sind noch nicht alle Merkmale von XSL-FO umgesetzt, so dass sich bei der Verwendung meist Einschränkungen ergeben. Der wesentliche Vorteil von fop ist damit die Tatsache, dass er frei verfügbar ist.

Die Einbindung in Java

Allen Prozessoren ist gemein, dass sie sich über die Java API for XML Processing (JAXP)[24] in Java einbinden lassen. JAXP stellt zum einen Mechanismen zur Verfügung, um die Standards SAX und DOM zum Parsen von XML Dokumenten zu nutzen und unterstützt zum anderen die Verwendung von XSLT. Dazu bietet JAXP mehrere APIs, die in unterschiedliche Packages unterteilt sind und die im Folgenden kurz erläutert werden sollen:

`javax.xml.transform`	In diesem Package werden die Klassen `TransformerFactory` und `Transformer` definiert, die die Einbindung von XSLT Prozessoren ermöglichen. Über `TransformerFactory` wird ein `Transformer` Objekt instantiiert, mit dessen `transform()` Methode ein Input (als `Source`) in einen Output (`Result`) umgewandelt werden kann. Input und Output können SAX und DOM Instanzen sein sowie Streams (die beispielsweise durch Auslesen von Dateien erzeugt werden). Für den einheitlichen Zugriff auf Input und Output werden die Interfaces `Source` und `Result` bereitgestellt, die Objekte der gerade genannten Typen enthalten können.

[23] [FOP]
[24] [JAXP]

`javax.xml.transform.dom`	Hier befinden sich Klassen zur Erzeugung von Input bzw. Output aus einem Data Object Model (DOM)
`javax.xml.transform.sax`	Analog zum `dom` Package können hier Input und Output aus einem SAX Handler erzeugt werden.
`javax.xml.transform.stream`	In diesem Package schließlich wird die Erzeugung von Input und Output aus I/O Streams ermöglicht.

Die Verwendung von JAXP Klassen lässt sich ähnlich wie in Abbildung 2-2 darstellen:

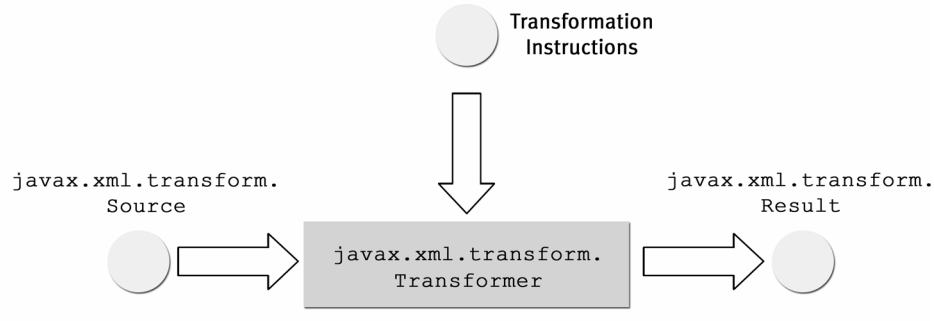

Abbildung 2-3 Wesentliche JAXP Klassen

Bei der Einbindung von fop in Java Applikationen müssen noch einige Besonderheiten berücksichtigt werden; Näheres dazu ist in Abschnitt 5.6.4 zu finden.

2.3.6. Entwurfsmuster

Vorbemerkung

Die hier beschriebenen Muster wurden im Laufe dieser Arbeit zunächst eingeständig aufgrund selbst angestellter Überlegung entworfen bzw. entwickelten sich während der Anwendung und dem Test der Funktionen von XSLT. Bei Recherchen in der erst später entdeckten Quelle [KAY01] stellte sich heraus, dass die dort beschriebenen Muster sich größtenteils mit denen dieser Arbeit deckten. Um dieser Arbeit einen allgemeingültigeren Charakter zu verleihen, werden bei der nun folgenden Beschreibung dieser Muster die Bezeichnungen aus [KAY01] übernommen.

Entwurfsmuster stellen den Versuch dar, Verfahren, die sich im praktischen Einsatz als sinnvoll erwiesen haben, auf ihre Gemeinsamkeiten hin zu untersuchen und dabei die Schlüsselmerkmale so zusammenzufassen, dass dabei ein allgemeiner Ansatz erkennbar wird, der einen wiederholten Einsatz eines Musters ermöglicht.

Fill-In-The-Blanks[25]

Basiert auf dem Prinzip von Vorlagen (Templates), die in vielen Template-Sprachen eingesetzt werden, beispielsweise in JSP oder ASP. Dabei existiert eine statische Vorlage, die das Layout eines

[25] vgl. [KAY01], S. 60off.

Dokuments bereits bestimmt, in die an definierten Stellen dynamische Inhalte eingefügt werden können.

```
01: <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
02:   <body>
03:     <table>
04:       <tr>
05:         <td>
06:           <xsl:value-of select="para"/>
07:         </td>
08:       </tr>
09:     </table>
10:   </body>
11: </html>
```

Listing 2-2

Dieses Muster verwendet in sehr eingeschränktem Maß Funktionalitäten von XSLT und zielt in erster Linie auf Anwender, die wenige Kenntnisse von XSLT haben bzw. sich diese nicht aneignen müssen (etwa Webdesigner). Voraussetzung hierfür ist, dass die Quelldaten im XML Format vorliegen und nicht, wie bei vielen Template-Sprachen, direkt von einer Datenbank abgefragt werden.[26]

Navigational Stylesheets[27]

Diese werden als eigenes Stylesheet definiert und nicht mehr in andere Dokumente eingebunden. Auch hier ist das Layout ein großer Bestandteil des Dokuments mit dem Unterschied, dass mit dem Element `<xsl:template name="">` sozusagen Methoden definiert werden können, die bereits komplexere Berechnungen durchführen können:

```
01: <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
03:   <body>
04:     <table>
05:       <tr>
06:         <td>
07:           <xsl:call-template name="doSomething"/>
08:         </td>
09:       </tr>
09:     </table>
10:   </body>
11: </html>
12:
13: <xsl:template name="doSomething">
14:   <!-- Ausführen von XSLT Funktionen wie etwa sum() o.ä. -->
15: </xsl:template>
```

Listing 2-3

Wesentliches Merkmal dieses Musters ist dabei, dass – den Konzepten von XSLT zufolge – nicht die Reihenfolge bei der Abarbeitung eines Templates von Bedeutung ist (was bei prozeduralen Sprachen sehr wohl wichtig ist), sondern viel mehr die genaue Angabe, welche Elemente an welcher Stelle des Dokuments bearbeitet werden sollen (also die gezielte Navigation in der Datenstruktur).

[26] Wie in Abschnitt 2.3.5 erwähnt, arbeiten XSLT Prozessoren primär nicht mit XML Dokumenten, sondern mit Baumstrukturen. Viele Prozessoren bieten diesbezüglich die Möglichkeit der Erweiterung, so dass die benötigte Baumstruktur auch aus anderen Datenquellen - etwa durch einen Datenbankzugriff über JDBC - erzeugt werden kann.

[27] Vgl. [KAY01], S. 602ff.

Rule-based Stylesheets[28]

Dieses Muster kann als Umsetzung des wesentlichen Konzepts von XSLT angesehen werden: Dabei geht es um Definition von Regeln mittels `<xsl:template>`, die beschreiben, was der XSLT Prozessor zu tun hat, wenn er auf ein Element stößt, das dem Muster im Attribut `match` entspricht. Dieses Muster soll auch deshalb hier nochmals explizit erwähnt werden, da beispielsweise im ersten Prototyp der später beschriebenen Anwendung, in dem ebenfalls XSLT zum Einsatz kam, keineswegs mit diesem Mechanismus gearbeitet wurde. Es sei deswegen betont, dass dies in keiner Weise zwingend ist, auch wenn im Laufe dieser Arbeit nahezu ausschließlich mit diesem Muster gearbeitet wird.

Computional Stylesheets[29]

Der Einsatz von rekursiven Template-Aufrufen stellt ein wesentliches Entwurfsmuster dar, dass vor allem dann Verwendung findet, wenn XSLT nicht nur zur Formatierung von XML Dokumenten verwendet wird, sondern für komplexere Berechnung, etwa für die Aggregation von Daten, die nicht in der gewünschten Form im Ausgangsdokument enthalten sind. [Kay02] spricht hier von einem „computional pattern“, da hier das Ausführen von Berechnungen im Vordergrund steht. Prinzipiell ist für Berechnungen die Verwendung von Variablen möglich; es gibt in XSLT dazu das Element `<xsl:variable>` mit den Attributen `name` und `select`. Diese fungieren allerdings im Kontext von XSLT als Konstanten, da ihnen nur einmalig ein Wert zugewiesen werden kann.

Um also Berechnungen vorzunehmen, bei denen etwa der Wert einer bestimmten Eigenschaft mit der Anzahl bestimmter Elemente geändert werden soll, muss in XSLT mit den Mitteln der *konditionalen Zuweisung* und der *Rekursion* gearbeitet werden.

- *Konditionale Zuweisung*
  Soll eine Variable einen Wert annehmen, der abhängig ist von anderen Bedingungen, so kann dies in XSLT wie folgt ausgedrückt werden:

```
01: <xsl:variable name="backgroundColor">
02:   <xsl:choose>
03:     <xsl:when test="DISPLAY_CATEGORY/@ID=12">
04:       #FFFFFF
05:     </xsl:when>
06:     <xsl:otherwise>
07:       #CCCCCC
08:     </xsl:otherwise>
09:   </xsl:choose>
10: </xsl:variable>
```

Listing 2-4

Hier wird der Variablen `backgroundColor` ein Wert in Abhängigkeit von der Eigenschaft des Elements `DISPLAY_CATEGORY` zugewiesen.

- *Rekursion*
  Mit dem Element `<xsl:template name="">` lassen sich in XSLT Methoden ausdrücken, die sich nicht auf ein Element im XML Quelldokument beziehen müssen, sondern explizit mittels der Anweisung `<xsl:call-template name="">` aufgerufen werden können. Damit lässt

[28] vgl. [KAY01], S. 604ff.
[29] vgl. ebd., S. 608ff.

beispielsweise ein Wert in Abhängigkeit bestimmter Elementeigenschaften inkrementieren und einer Variablen zuweisen, indem die Methode zur Bestimmung des Wertes rekursiv aufgerufen wird. Dies soll in folgendem Beispiel verdeutlicht werden:

```
01: <xsl:variable name="theValue">
02:   <xsl:call-template name="doIncrement">
03:     <xsl:with-param name="start" select="number(1)"/>
04:   </xsl:call-template>
05: <xsl:variable>
06:
07: <xsl:template name="doIncrement">
08:   <xsl:param name="start">
09:   <xsl:choose>
10:     <xsl:when test="$start&lt;10">
11:       <xsl:call-template name="doIncrement">
12:         <xsl:with-param name="start" select="$start + 1"/>
13:       <xsl:call-template>
14:     <xsl:when>
15:     <xsl:otherwise>
16:       <xsl:value-of select="$start"/>
17:     </xsl:otherwise>
18:   </xsl:choose>
19:</xsl:template>
```

Listing 2-5

Innerhalb des Anweisungsblocks der Variablen `theValue` wird das Template `doIncrement` aufgerufen. Dieses ruft sich selbst auf und erhöht dabei den Wert des Parameters `start` jeweils um 1. Nach dem zehnten Aufruf wird der Wert des Parameters zurückgegeben und der Variablen zugewiesen. Auf diese Art und Weise lassen sich die Ergebnisse komplexer Berechnungen auch in XSLT Variablen speichern. Allerdings muss dabei berücksichtigt werden, dass die Ausführung von einer steigenden Anzahl von Rekursionsschritten zu Lasten der Performance des Programms geht, in dem die rekursive Berechnung ausgeführt wird. Eine Verwendung dieser Muster wird nochmals in Abschnitt 5.6.5 bei der Berechungen der Sendezeiten beschrieben.

Prinzipiell sei dazu angemerkt, dass beim Gesamtentwurf einer Anwendung, also vom Datenmodell bis hin zur Frontend, zur Abfrage dieser Daten nach Möglichkeit so vorgegangen werden sollte, dass solche Berechnungen nicht mehr bei der *Darstellung* der Daten stattfinden müssen. Wie sich aber auch im Verlauf dieser Arbeit herausgestellt hat, gibt es durchaus Situationen, in den erst zum Zeitpunkt der Darstellung der Daten Berechnungen stattfinden können, deren Umsetzung durchaus mit XSLT vorgenommen werden kann.

Allgemein lässt sich sagen, dass mit zunehmender Anzahl der Rekursionsschritte die Ge–samtperformance einer Berechnung negativ beeinflusst wird. Da außerdem der rekursive Aufruf von Methoden die Gefahr von versteckten Endlosschleifen mit sich bringt, sollte der Einsatz nur dann erfolgen, wenn andere Möglichkeiten zur Berechnung fehlen; für die meisten Berechnungen lassen sich die Funktionen der XSLT Bibliothek nutzen.

3. Analyse und Anforderungen

3.1. Analyse

Mit der Analyse sollen die Eigenschaften des bestehenden Systems aufgezeigt werden, innerhalb dessen der SchedEx Viewer eingebunden werden muss. Dazu sollen die wesentlichen Zusammenhänge und Schnittstellen aufgezeigt werden, die der SchedEx Viewer benötigt, um mit anderen Komponenten innerhalb des Systems zu kommunizieren. Die Analyse bildet zusammen mit den in Abschnitt 3.2 erfassten Anforderungen die Grundlage für den Entwurf der Webanwendung.

3.1.1. Die Architektur von ProSeco

Nachdem der Gesamtzusammenhang sowie die Absicht von ProSeco bereits in der Einleitung in Kapitel 1 erläutert wurden, sollen hier die technologischen Aspekte von ProSeco näher betrachtet werden.

In der technischen Infrastruktur ist ProSeco als 3-Tier Architektur realisiert. Diese besteht aus:

1. der User Interface Schicht (Client), umgesetzt in Web-Frontends wie Browsern oder Java-Client-Applikationen,
2. der Businesslogik Schicht, die über J2EE auf einem IBM WebSphere Applikation Server basiert sowie
3. der Datenhaltungsschicht in Form einer Informix Datenbank.

Der funktionale Schwerpunkt des SchedEx Viewers liegt auf der Repräsentation von Sendeplandaten auf der Seite des Clients; zudem ist die Integration innerhalb der Middleware von zentraler Bedeutung. Ein direkter Zugriff auf die Persistenzschicht muss nicht erfolgen; vielmehr wird indirekt über die Komponente SchedEx, die im Folgenden analysiert wird, darauf zugegriffen.

3.1.2. Vorstellung von SchedEx

Wie bereits erwähnt ist SchedEx ein Teilprojekt von ProSeco und dient als standardisierte Schnittstelle für den Zugriff auf Sendeplandaten sowohl innerhalb von ProSeco als auch für Systeme, die von extern auf entsprechende Daten zugreifen müssen. Zu diesem Zweck werden die Daten aus der Datenbank des bisherigen Sendeplanungstools SePla in XML konvertiert, um damit eine standardisierte Weiterverarbeitung zu erleichtern.

Das Datenmodell

Das Datenmodell in Form einer DTD ist die Grundlage für das XML Format, in dem die Sendepläne vorliegen und somit auch der Ausgangspunkt und das Kommunikationsmittel für alle Darstellungsformen, die mit Hilfe von XSLT Transformationen erzeugt werden sollen. In Abbildung 3-1 wird ein Auszug aus dem Datenmodell gezeigt. Darin sind die Elemente eines Sendeplans und deren Beziehung zueinander, wie sie in der DTD definiert werden, dargestellt. Das Modell wird im Anschluss daran weiter erläutert.

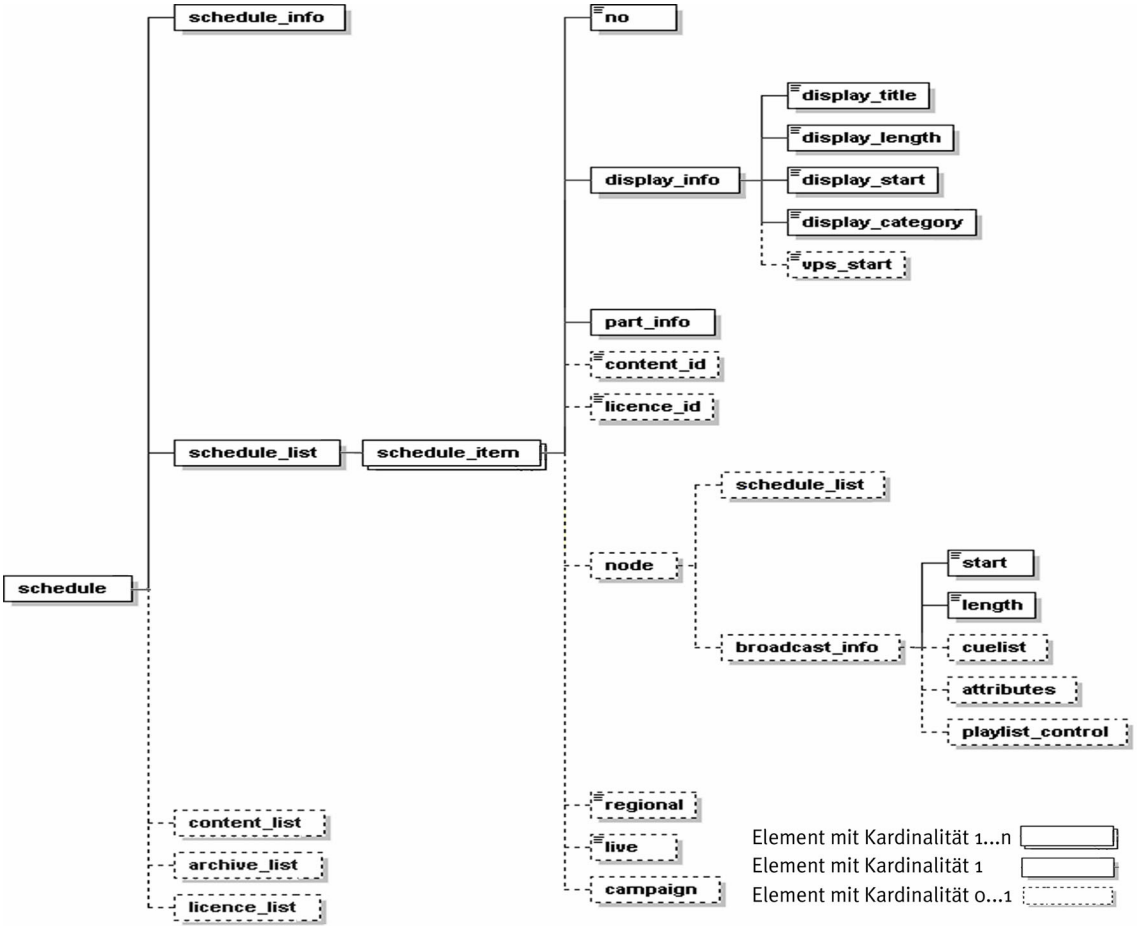

Abbildung 3-1: Auszug aus Datenmodell von SchedEx

In diesem Modell besteht ein Sendeplan (`schedule`) zunächst aus einem allgemeinen Teil, der Informationen (`schedule_info`) enthält, die den gesamten Sendeplan betreffen (etwa das Datum der Ausstrahlung und der jeweilige Sender) und eine Liste von Sendeplanelementen (`schedule_item`), die alle Elemente enthält, die für diesen Zeitraum und für diesen Sender zur Ausstrahlung bestimmt sind. Jedes Sendeplanelement erhält eine fortlaufende Nummerierung und enthält nähere Informationen zur Ausstrahlung wie Titel der Ausstrahlung, Startzeit, Länge und Kategorie. Zudem kann mittels unterschiedlicher Elemente wie `content_id` oder `licence_id` auf weitere Informationen bezüglich des eigentlichen Inhalts, der Lizenzen oder der Archivierung verwiesen werden. Diese Informationen werden in externen Listen gehalten, die ebenfalls in einem Sendeplan enthalten sein können.

In der Konzeption von ProSeco kann ein Sendeplanelement auch als eine Art übergeordnetes Element (ein *Container*) verstanden werden, das in sich weitere Elemente enthalten kann. Beispielsweise kann ein Werbeblock als ein solcher Container gesehen werden, der die betreffenden Werbespots enthält. Dieser Ansatz ist auch im Datenmodell verwirklicht worden. So kann ein Sendeplanelement wiederum eine Liste enthalten, in der weitere Elemente aufgeführt sind. Dieser Ansatz lässt sich mit dem bekannten Composite Pattern[30] vergleichen. Ein Sendeplanelement enthält in diesem Entwurf des Modells stets eine Längenangabe im Format HH:MM:SS, die die Länge auf die Sekunde angibt. Optional kann ein Element zusätzlich eine Länge enthalten, die in

[30] Vgl. [GoF], S. 163

Millisekunden angeben wird. Diese Eigenschaft spielt eine bedeutende Rolle bei der Berechnung der Sendezeiten, auf die im Kapitel 5 weiter eingegangen werden wird.

Die Architektur von SchedEx

SchedEx ist in der zuvor beschrieben Gesamtarchitektur von ProSeco integriert. Hier sollen nun kurz die Mechanismen beschrieben werden, mit denen letztendlich der XML Stream erzeugt wird, der anderen Anwendungen und Systemen zur Verfügung steht. Im Wesentlichen besteht SchedEx aus einer Reihe von Servlets und Enterprise JavaBeans (EJBs), die im Web Container bzw. im EJB Container des Applikationsservers liegen. Über EJB wird der Zugriff auf die zugrunde liegende Datenbank von SePla bzw. unterschiedliche andere Datenbanken, die bestimmte Teilinformationen für einen Sendeplan enthalten, realisiert; zudem wird mittels EJB ein Cache Mechanismus implementiert, der einmal aus der Datenbank geladene Sendeplandaten zwischenspeichert, um so Zugriffszeiten für häufiger verwendete Daten zu optimieren. Extern sind die Sendeplandaten über ein Servlet abrufbar, das über einen URL aufgerufen wird und das als Parameter u. a. das gewünschte Sendedatum und die `ID` des Senders übergeben bekommt. Innerhalb von SchedEx sind mehrere Servlets vorhanden, die die Sendeplandaten in unterschiedlichen Ausgabeformaten zurückliefern können, etwa als XML Stream, als DOM `Document` oder bereits als HTML.

Die folgende Abbildung zeigt einen Überblick über die einzelnen Komponenten, deren Abhängigkeiten sowie die Schnittstellen, über welche von extern aus auf die Sendeplandaten zugegriffen werden kann.

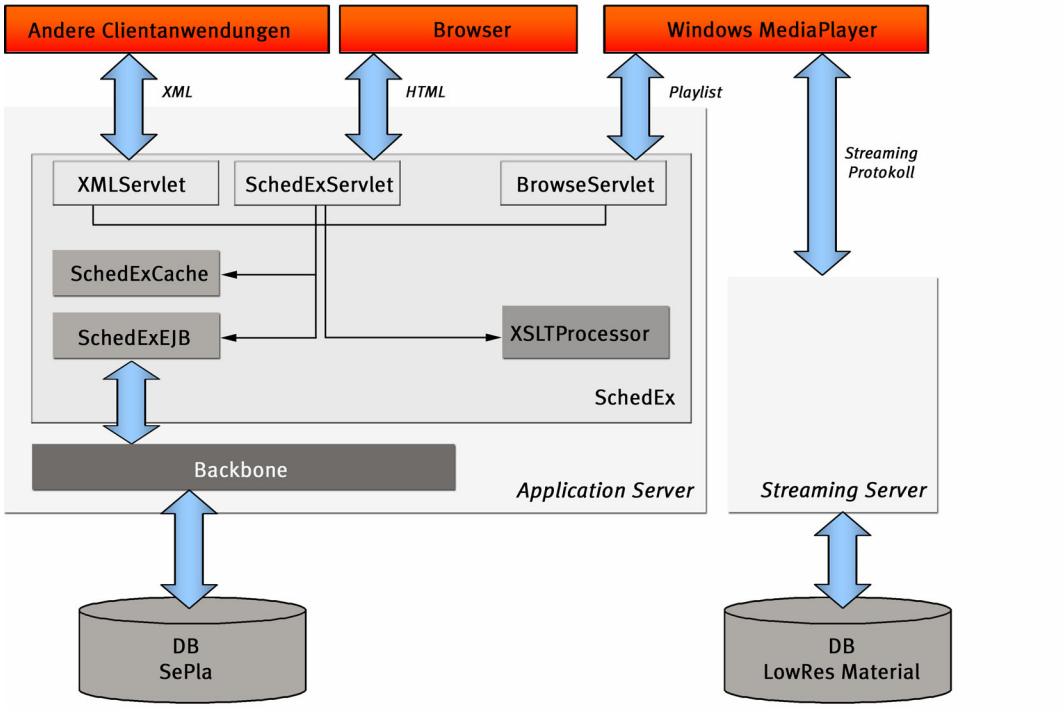

Abbildung 3-2 Die Architektur von SchedEx

Dabei soll der Zugriffsmechanismus nochmals verdeutlicht werden: Die Servlets können als Business Delegate[31] verstanden werden, die die Schnittstelle für externe Zugriffe darstellen. Dabei kann innerhalb der Servlets entweder auf zwischengespeicherte Sendeplandaten zugegriffen

[31] vgl. [BIE02], S. 72

werden oder direkt eine Anfrage an die Persistenzschicht gestartet werden. Je nach Bedarf der zugreifenden Anwendung können unterschiedliche Ausgabeformate zurückgegeben werden. Im rechten Teil der Abbildung wird der Streaming-Mechanismus gezeigt: Aus den Daten eines Sendeplans kann über eine XSLT Transformation eine Playlist (im ASX Format) generiert werden, die im Browser über einen Verweis bereitgestellt wird. Klickt der Anwender auf den Verweis, wird das Servlet angesprochen, das die Transformation der Playlist startet und diese an der Browser zurückliefert. Dort wird die ASX Datei (bzw. der Stream) erkannt und, falls vorhanden, das entsprechende PlugIn des WindowsMedia Players gestartet. Über das Streaming haben die Anwender die Möglichkeit, sich die Programminhalte eines Sendetags anzusehen.

Die Stabilität der Schnittstellen

Die Entwicklung der Komponente SchedEx ist noch nicht abgeschlossen, da bestimmte Aspekte bezüglich der Anbindung an Altsysteme und bezüglich der Sicherheit noch nicht umgesetzt wurden. Das bedeutet, dass der Zugriffsmechanismus auf Sendeplandaten vom Viewer aus fest steht, aber nicht immer mit der Verfügbarkeit von SchedEx gerechnet werden kann. Aus diesem Grund muss als Alternative ein Weg gefunden werden, der diese Funktionalität zumindest ansatzweise simuliert. Beim Datenmodell sind evtl. noch einige Änderungen zu erwarten, so dass der spätere Entwurf auch hier Änderungen berücksichtigen muss und flexibel sein sollte.

3.2. Anforderungen

Für die Erfassung und die Kommunikation von Anforderungen wird im RUP ein eigener Prozess definiert, das Requirement Engineering. Ziel des Requirement Engineering Workflows ist die Beschreibung, was das zu entwickelnde System leisten muss. Dabei dient die einheitliche Erfassung von Anforderungen als Kommunikationsmittel für Entwickler und Kunden, um Missverständnissen auf beiden Seiten vorzubeugen.

Anforderungen bestehen an den SchedEx Viewer seitens des Kunden (vorwiegend funktional), der die Anwendung später produktiv einsetzen soll und auch seitens der PSI (vorwiegend nicht-funktional), da die Anwendung mit anderen Systemen der PSI kommunizieren muss. Anforderungen werden dabei durch verschiedene Methoden (beispielsweise durch Anwenderinterviews) aufgenommen und in einer Datenbank gehalten. Zur einheitlichen Erfassung von Anforderungen werden sprachliche Templates verwendet; damit soll objektive eine Beschreibung der Anforderungen ermöglicht werden, die nicht (unbewusst) durch die subjektive Wahrnehmung des Entwicklers oder des Kunden beeinflusst wird.

Der Aufbau eines solchen Templates lässt sich wie in Abbildung 3-3 darstellen:

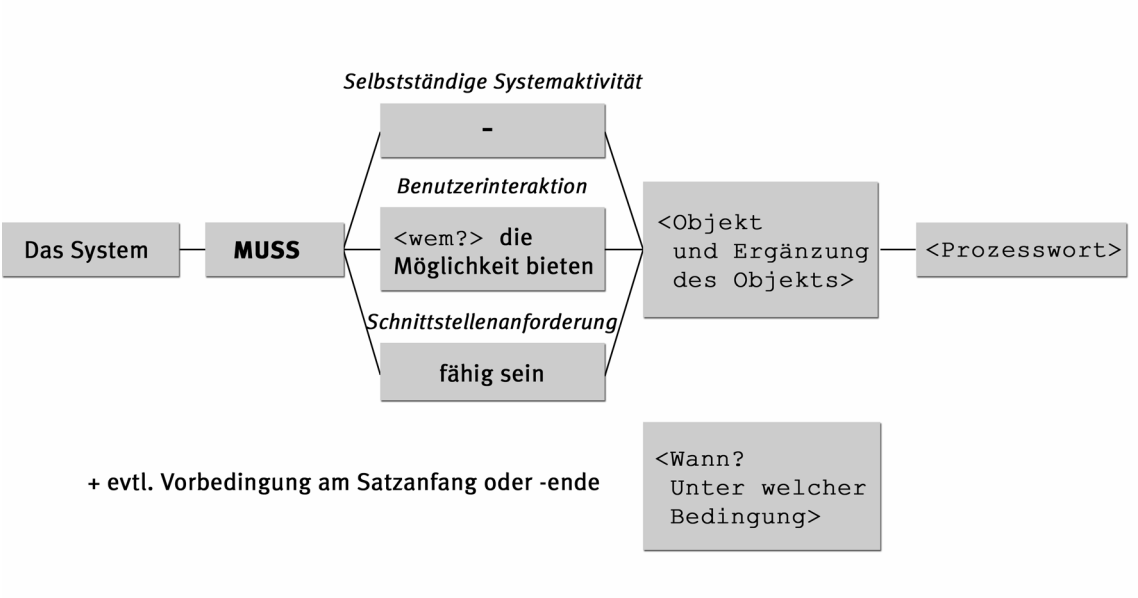

Abbildung 3-3 Sprachliches Template zur Erfassung von Anforderungen

In den folgenden Übersichten werden die Anforderungen nicht alle einzeln genannt, sondern der Übersichtlichkeit wegen zusammengefasst dargestellt.

3.2.1. Funktionale Anforderungen

Bei diesen handelt es sich um Anforderungen, die seitens der Anwender gestellt werden, und die die Hauptfunktionalität der Anwendung betreffen. Diese Anforderungen wurden in Interviews mit Anwendern aus dem Bereich Sendeabwicklung spezifiziert. Im Folgenden sollen nun die wesentlichen funktionalen Anforderungen genannt und kurz erläutert werden.

FA 01 Für die Darstellung eines Sendeplans gibt es eine Standard-Ansicht sowie individuelle Ansichten für unterschiedliche Rollen. Für die Form der Darstellung gibt es konkrete Vorgaben von den Anwendern, zusätzlich können auch alternative Formen der Darstellung erarbeitet werden.

FA 02 Die Ansicht eines Sendeplans soll über einen URL Abruf in einem Browser möglich sein. Da der SchedEx Viewer zunächst nur für den Einsatz der ProSiebenSat.1 Gruppe gedacht ist, kann als Browser zunächst der Internet Explorer von Microsoft vorausgesetzt werden.

FA 03 Bei den Anwendern wird zwischen den Rollen Sendeleiter (SL), Sendegrafik (SG) und Sendeabwicklung (SAW) unterschieden. Den Rollen werden entsprechende Sendeplan-Ansichten zugeordnet, die Informationen enthalten, die

1. der Benutzerrolle entsprechend dargestellt werden und
2. nur angezeigt werden, wenn die mit der Rolle verbundenen Rechten dies zulassen.

FA 04 Der Standard-Ansicht wird eine fest definierte Menge an Elementen zugeordnet, die angezeigt werden müssen (Datum, Sender, Status, Version, Kommentar, Autor, Optimale Minute, Gesamtauslastung, Gründeliste, Anzeige der Werbestundengrenze, Anzeige der Auslastung pro Werbestunde)

FA 05 Die Sendeelemente der Standard-Ansicht haben die folgenden Inhalte: Startzeit, Nummerierung, Titel, Kategorie, Länge, Cuelist, Schnittfassung, Contentinformation, Archivinformation, Lizenzinformation. Aus diesen Elementen werden für die Rollen SL (Sendeleiter), SG (Sendegrafik) und SAW (Sendeabwicklung) die jeweils notwendigen Elemente ausgewählt und in entsprechend gewünschter Art und Weise dargestellt.

FA 06 Es werden zwei Ausgabeformate unterschieden:

1. Darstellung für den Bildschirm (HTML) und
2. für den Druck (PDF)

FA 07 Die Auswahl der Darstellungsformen muss dem Anwender über ein Benutzermenü zugängig gemacht werden. Die angezeigten Menüoptionen sind abhängig von der Rolle und damit von den Rechten des Anwenders.

FA 08 Für die Darstellung der Druckausgabe werden von der zuständigen Kontaktperson auf Kundenseite Vorlagen geliefert, die als Richtlinien zur Umsetzung dienen. Für die Druckausgabe wird weiterhin festgelegt:

1. Für unterschiedliche Benutzerrollen werden entsprechende Druckoptionen angeboten.
2. Im Fall, dass ein Anwender mehreren Rollen zugeordnet ist, muss das System eine entsprechende Auswahl der Rolle zur Verfügung stellen.
3. Für die Druckausgabe wird das Querformat festgelegt, sowie eine farbliche Kennzeichnung der Elemente, die sowohl als Grautonabstufung als auch in Farbe deutlich unterscheidbar ist.
4. In Abhängigkeit von der Kategorie eines Sendeelements, wird der Hintergrund entsprechend farbig hinterlegt. Die Vorgaben hierfür werden vom Kunden gestellt.

FA 09 Die Sendezeiten eines Sendetags müssen evtl. neu berechnet werden. Diese Anforderung ergibt sich aus der folgenden Tatsache: Der Sendeplan, der im SchedEx Viewer betrachtet wird, muss nicht die endgültige Fassung darstellen und kann zeitliche Überlappungen und Lücken zwischen mehreren Sendeelementen enthalten. Diese so genannten Fixzeit-Verletzungen müssen dem Anwender angezeigt werden.

FA 10 Bei der Darstellung des Sendeplans, der dem aktuellen Sendetag entspricht, soll die dem Anwender die Möglichkeit geboten werden, Informationen über das aktuell ausgestrahlte Sendeelement abzurufen. Diese Informationen (Titel, Länge und verbleibende Dauer) sollen in Abhängigkeit der Uhrzeit aktualisiert werden.

FA 11 Der SchedEx Viewer soll auch die Möglichkeit bieten, Anwendern die Ansicht von Videomaterial eines bestimmten Sendetages zu bieten. Dabei kann auf einen Streaming Server zugegriffen werden, auf dem das Material in LowRes Qualität abgelegt wird.

Die wesentlichen funktionalen Anforderungen lassen sich in dem folgenden Use Case Diagramm festhalten:

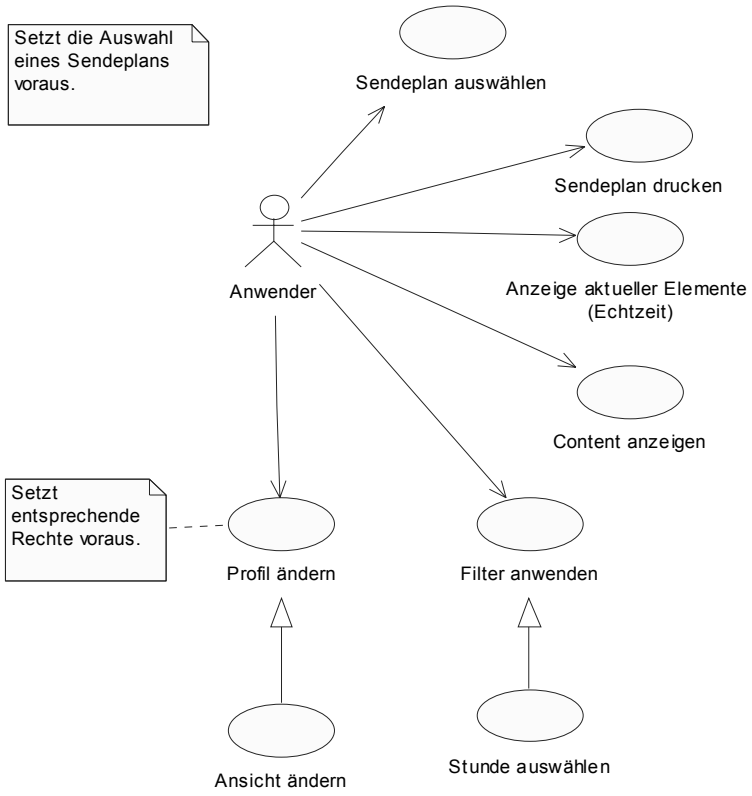

Abbildung 3-4 Use Case Diagramm

In diesem Use Case sind alle essentiellen Anforderungen bezüglich der Funktionalitäten des SchedEx Viewers zusammengefasst.

3.2.2. Nicht-funktionale Anforderungen

Diese Anforderungen betreffen die technische Umsetzung des SchedEx Viewers und legen Merkmale der Anwendung wie etwa Performance oder Sicherheit fest. In der PSI werden nicht-funktionale Anforderungen in einer zentralen Datenbank gehalten, um diese Anforderung projektübergreifend verfügbar zu machen. Im Folgenden sollen die wesentlichen Merkmale genannt werden, deren Umsetzung bei der Entwicklung der Anwendung konkret eine Rolle spielen. Für andere Merkmale werden in anderen Bereichen des Gesamtprojekts Konzepte entworfen, die zum Zeitpunkt der Analyse und des Designs des Viewers noch nicht konkret berücksichtigt werden können: so muss beispielsweise für die Benutzerauthentifizierung ein einheitliches Gesamtkonzept bestehen, dass bei der Entwicklung des SchedEx Viewers mit eingebunden werden sollte.

Die Anforderungen im Überblick:

NFA 01 Modifizierbarkeit

Für eine verbesserte Wartbarkeit wird die Anwendung in einzelne Module unterteilt. Zudem werden standardisierte Technologien (Java, J2EE) verwendet.

NFA 02 Performance

Zur Darstellung eines Sendeplans wird ein mehrstufiger Prozess durchlaufen, der unter anderem mindestens eine Datenbankabfrage sowie mindestens eine Transformation mittels

XSLT beinhaltet. Dieser Prozess sollte die maximale Dauer von 50 Sekunden nicht überschreiten. Diese Zeitangabe orientiert sich an den Ladezeiten im bisherigen Planungs-Tool SePla und gilt für Sendepläne, welche für einen Sendetag gelten.

NFA 03 Analysierbarkeit

Die Unterteilung des Systems in Module, um Fehlerbereiche eingrenzen zu können.

NFA 04 Stabilität

Der SchedEx Viewer ist zur Erzeugung einer Darstellung eines Sendeplans auf andere Komponenten angewiesen: die Komponente SchedEx stellt die Sendeplandaten zur Verfügung; das zugrunde liegende Datenmodell des Sendeplans dient als Grundlage für die Transformation. Änderungen an diesen beiden Punkten müssen so bedacht werden, dass sie möglichst einfach und zentral vorgenommen werden können, ohne die Stabilität der gesamten Anwendung zu gefährden.

NFA 05 Zuverlässigkeit

Unter diesem Punkt werden verschiedene Aspekte der Anwendung festgehalten, u. a. Performance des Clients, einer Datenbankabfrage, die Verfügbarkeit des Systems, die Stabilität, die Wiederherstellbarkeit, Fehlertoleranz, Robustheit, Prüfung der eingegebenen Benutzerdaten, die Ausgabe von Fehlermeldung (Logging).

NFA 06 Übertragbarkeit

Die Austauschbarkeit einzelner Komponenten und somit die Wiederverwendbarkeit der Anwendung sowie deren Anpassung an neue Anforderungen und Umgebungen.

NFA 07 Benutzbarkeit

Unter diesem Punkt werden Anforderungen festgehalten, die zum einem die Benutzbarkeit der Anwendung durch eine grafische Oberfläche spezifizieren als auch die Dokumentation der wesentlichen Merkmale der Anwendung. Für den SchedEx Viewer ergibt sich konkret die Forderung nach der Darstellung des Fortschritts einer Transformation. Wie schon unter dem Punkt Performance (NFA 02) bemerkt, kann der Prozess zur Darstellung durchaus einige Sekunden dauern. Deshalb soll der Anwender zumindest über den Fortschritt der Transformation informiert werden.

3.3. Zusammenfassung

Aus Analyse und Anforderungen ergibt sich, dass ein Teil der Funktionalität des SchedEx Viewers bereits in der SchedEx Komponente zumindest konzeptionell umgesetzt wurde. Das bedeutet, dass diese Konzepte in der Entwicklung des SchedEx Viewers übernommen und weiterentwickelt bzw. modifiziert werden können.

Der Großteil der funktionalen Anforderungen kann mit XSLT abgedeckt werden. Das betrifft die gezielte Filterung und die übersichtliche Darstellung der Sendeplandaten. Im Bereich der nicht-funktionalen Anforderung liegt der Focus verstärkt auf anderen Technologien, die einen effizienten Betrieb des Viewers in der Gesamtumgebung garantieren.

Die Komponente SchedEx ist selbst noch in der Entwicklung. Es kann davon ausgegangen werden, dass in Zukunft Änderungen vorgenommen werden zum einen im Bereich der angebundenen Datenbanken, aus denen Informationen zur Erstellung des kompletten Sendeplans abgefragt werden, und zum anderen bei der Authentifizierung der Anwender sowie einer damit verbundenen Zuordnung zwischen einzelnen Anwendern und Rollenprofilen. Was die Anbindung weiterer Datenbanken betrifft, so muss dies zunächst im Entwurf des Viewers nicht weiter berücksichtigt werden; hier kann damit gerechnet werden, dass nicht allen Informationen, die angezeigt werden sollen, auch wirklich im Sendeplan enthalten sind. Als Grundlage für den Entwurf von Mechanismen zur Darstellung kann das Datenmodell, also die DTD, genutzt werden. Deutlich stärker zu gewichten ist das Fehlen eines konkreten Rollenkonzepts. Da eine Unterscheidung zwischen verschiedenen Rollen funktionaler Bestandteil des Viewers ist, muss beim Entwurf versucht werden, Wege zu finden, die eine spätere Anbindung relativ leicht bewerkstelligen lassen.

4. Der Entwurf

4.1. Überblick

Nachdem im vorhergehenden Kapitel die Architektur analysiert und die Anforderung an die Funktionalitäten der Anwendung spezifiziert wurden, soll nun überlegt werden, wie die gewonnenen Erkenntnisse zu einem Systementwurf zusammengefasst werden können, der die Anforderungen berücksichtigt und die Anwendung möglichst optimal in die bestehende Infrastruktur einpasst.

Beim Design der Anwendung lassen sich zwei Kategorien unterscheiden, die zunächst getrennt voneinander betrachtet werden sollen: Der Entwurf der Webanwendung im Zusammenhang mit der umgebenden Projektarchitektur und damit die Erfüllung nicht-funktionaler Anforderungen. Zum anderen die Umsetzung der funktionalen Anforderungen, die vor allem die Darstellung der Sendepläne und damit die XSLT Technologie betrifft. Da das Projekt ProSeco J2EE als Plattform verwendet, werden zur Anbindung an die Architektur von ProSeco Technologien aus diesem Bereich verwendet. Die Erzeugung der Darstellung eines Sendeplans ist davon zunächst unabhängig; da aber innerhalb der J2EE mit JAXP eine eigene API dafür zur Verfügung gestellt wird, lässt sich eine gute Verbindung des funktionalen Bereichs an den nicht-funktionalen Bereich der Anwendung schaffen.

Zunächst soll ein grober Entwurf präsentiert werden, der vor allem die Anbindung an das Gesamtsystem berücksichtigt. Anschließend muss überlegt werden, mit Hilfe welcher Technologien die Funktionalitäten der Anwendung umgesetzt werden können und über welche Schnittstellen die einzelnen Komponenten der Anwendung miteinander kommunizieren können. Zum Schluss sollen die Feinheiten einzelner Bestandteile entworfen werden, die die zuvor präsentierten Schnittstellen nutzen. Im Einzelnen müssen die folgenden Punkte bedacht werden:

1. Anbindung des Viewers an die Architektur von SchedEx bzw. ProSeco
2. Entwurf einzelner Komponenten des Viewers
   - 2.1. Die Abfrage der Sendeplandaten
   - 2.2. Die Benutzeroberfläche
   - 2.3. Die Erzeugung der Sendeplan-Darstellung
3. Die Kommunikation der Komponenten und Überlegungen zum Ablauf

Es werden zunächst Teile des Gesamtentwurfs beschrieben, die nicht direkt mit XSLT zu tun haben, indirekt aber Einfluss nehmen auf die Gestaltung der Stylesheet Struktur und die Verarbeitung der Stylesheets. Ein Überblick über den Gesamtentwurf und die Abhängigkeiten der einzelnen Komponenten ist unter Abschnitt 4.5. auf Seite 51 zu finden.

4.2. Die Anbindung an die Infrastruktur

Durch die Analyse ist bekannt, dass über verschiedene Servlets auf die Sendeplandaten zugegriffen werden kann und diese in unterschiedlichen Formaten angefordert werden können. Zudem sollen der SchedEx Viewer und die SchedEx Komponente zunächst relativ getrennt voneinander entwickelt werden können. Um beide Komponenten lose miteinander koppeln zu können, bietet sich die Kommunikation über das Hypertext Transport Protokoll (HTTP) durch den Aufruf von URLs an. Durch

die Verwendung dieses Protokolls bleibt der SchedEx Viewer relativ unabhängig von anderen architektonischen Eigenschaften des Systems (etwa Firewalls), da damit ein weit verbreiteter Standard verwendet wird. Von der SchedEx Komponente muss damit lediglich ein URL bekannt sein, unter dem sie angesprochen werden kann und den angeforderten XML Stream liefert. Wird auf der Seite des SchedEx Viewers ebenfalls ein Servlet verwendet, so ließe sich der URL des SchedEx Servlets etwa als Parameter zur Laufzeit übergeben; damit wird eine flexible Anpassung an zukünftige Änderungen gewährleistet. Damit kann auch berücksichtigt werden, dass SchedEx noch nicht produktiv eingesetzt wird und nicht immer zur Abfrage von Sendeplänen zur Verfügung steht. Durch von Verwendung von Parametern kann also auch eine vorläufige Test-Komponente angesprochen werden, die beispielsweise auf lokal abgespeicherte Sendeplandaten zugreift und diese in gleicher Form wie SchedEx an den Viewer senden kann.

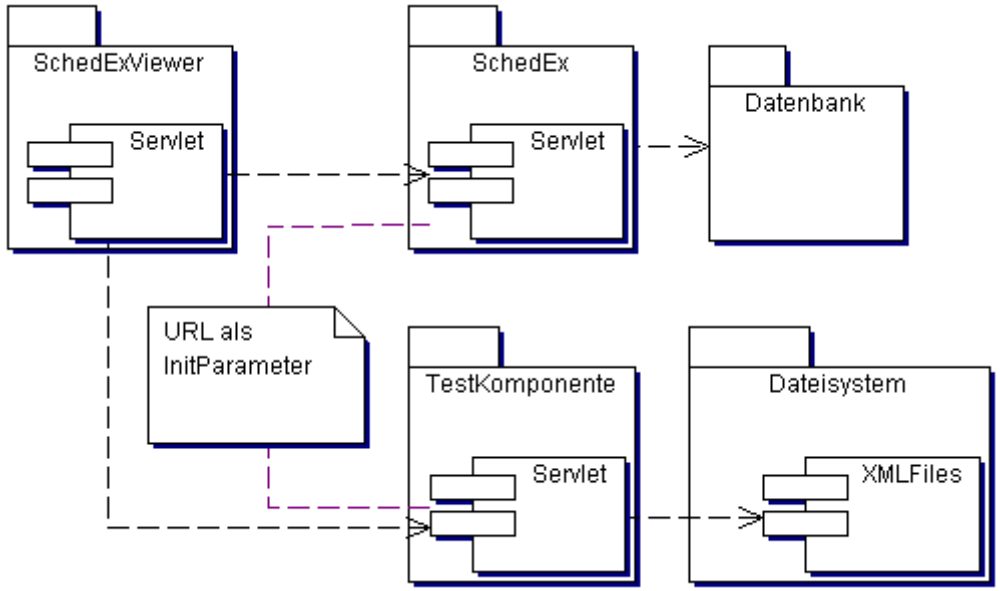

Abbildung 4-1 Die Anbindung an SchedEx

4.3. Die Komponenten der Anwendung

Der SchedEx Viewer lässt sich gemäß seiner geforderten Funktionen nochmals in einzelne Bereiche bzw. Komponenten unterteilen. Ein modularer Aufbau der Anwendung hat dabei den Vorteil, dass die einzelnen Module getrennt voneinander entwickelt werden können und somit die Wartbarkeit sowie die Anpassung der gesamten Anwendung an zukünftige Änderungen verbessert werden. Im Einzelnen lassen sich die folgenden Bereiche unterscheiden:

1. Die Abfrage der Sendeplandaten bei der SchedEx Komponente.
2. Die Benutzeroberfläche, über die die Sendeplan-Anforderungen seitens der Anwender eingegeben werden können und in der die Ansichten eines Sendeplans angezeigt werden. Dabei kann nochmals zwischen der statischen Menüführung und zusätzlichen dynamischen Bereichen, die echtzeitabhänige Informationen anzeigen, unterschieden werden.
3. Die Transformation des Sendeplans, die eine Darstellung gemäß den Eingaben des Anwenders erzeugt.

Die Unterteilung der Bereiche sowie deren Abhängigkeiten untereinander lassen sich schematisch wie folgt darstellen:

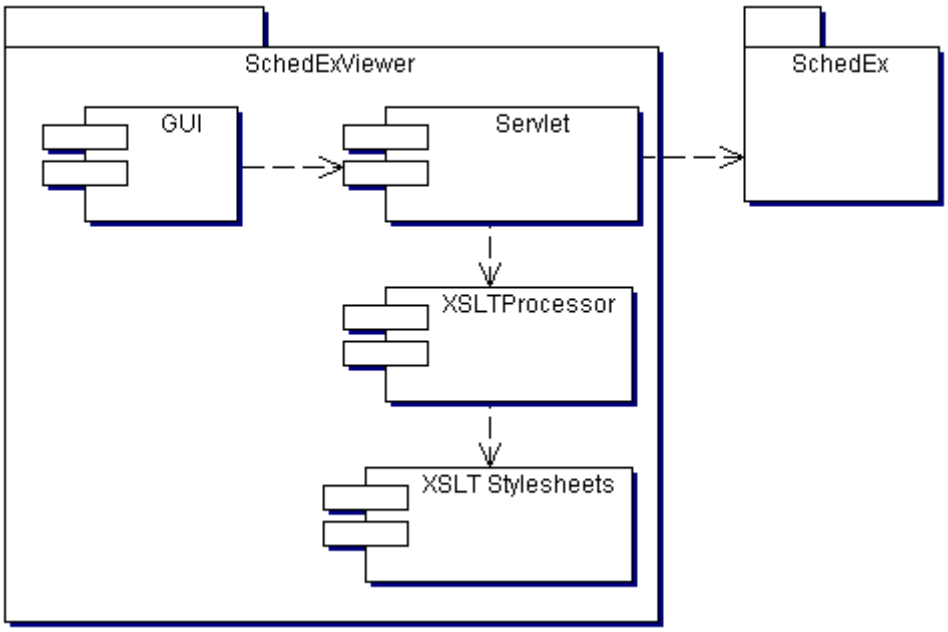

Abbildung 4-2 Die Komponenten des SchedEx Viewers

Beim Entwurf der beiden erst genannten Punkte lassen sich Technologien und Konzepte aus der J2EE anwenden. Das J2EE Framework hat sich inzwischen für die Umsetzung komplexer Webanwendungen fest etabliert. Damit haben sich eine große Anzahl von Best Practices und Entwurfsmustern gebildet, die beim Entwurf einer Webanwendung genutzt werden können. Zusätzlich gibt es in vielen Teilbereichen der J2EE Technologie bereits Komponenten, die diesen Bereich sehr gut abdecken und oftmals als Open Source zur freien Verfügung und Einbindung in andere Technologien stehen. Auf die Webanwendung bezogen, lassen sich zum einem das Struts[32] Framework für die Erzeugung einer Benutzeroberfläche und zum anderen das Cocoon[33] Framework als so genanntes Publishing Framework zur Erzeugung unterschiedliche Darstellungsformate eines Sendeplans durchaus in Betracht ziehen. Betrachtet man allerdings den Gesamtzusammenhang des SchedEx Viewers so lässt sich feststellen: Der Viewer ist eine relativ kleine Komponente des Gesamtprojekts, die zunächst einem kleinen Kreis von Anwendern zur Verfügung stehen soll. Insofern muss der Aufwand, der mit der Einrichtung einer Infrastruktur wie etwa Struts verbunden ist, verglichen werden mit dem Aufwand einer eigenen Implementation. Dazu lässt sich sagen, dass mit steigender Anzahl der Anwender und zunehmender Komplexität der Anwendung der Einsatz von Struts oder Cocoon empfehlenswert erscheint. Beide Frameworks haben bereits einigen Ent–wicklungsaufwand vorzuweisen und haben sich im praktischen Einsatz bewährt. Da jedoch der Schwerpunkt dieser Arbeit auf der Verwendung und der Einbindung von XSLT liegt, soll hier zunächst ein eigener Lösungsweg gefunden werden. Im Wesentlichen geht es darum, den Viewer best möglich und ohne hohen Aufwand in die bestehende Architektur einzubinden.

Für die unter Punkt 3 genannte Transformation wird schließlich XSLT eingesetzt werden.

[32] [STRUTS]
[33] [COCOON]

4.3.1. Die Abfrage der Sendeplandaten

Die Abfrage eines Sendeplans wird über das `TranformServlet` realisiert, das die folgenden Aufgaben hat:

- *Abfrage des Sendeplans bei SchedEx.*
  Die Anbindung an SchedEx folgt dem Mechanismus, der im vorigen Abschnitt beschrieben ist.

- *Weiterverarbeitung des von SchedEx empfangenen XML Streams.*
  Der XML Stream kann über eine Transformation in ein Objekt vom Typ `Source` umgewandelt werden. Bei `Source` handelt es sich um ein Interface, das in mit JAXP zur Verfügung gestellt wird. Damit liegt eine XML Quelle im richtigen Format für die eigentliche Transformation zur Erzeugung einer Darstellung vor.

- *Die Einbindung des XSLT Prozessors*
  Die im `Source` Objekt gehaltenen Daten können nun an den `Transformer` übergeben werden. Der `Transformer` wird wie in Abschnitt 2.3.5 beschrieben über JAXP eingebunden.

- *Die Rückgabe des Ergebnisses der Transformation an den Client*
  Mit XSLT ist es möglich, unterschiedliche Ausgabeformate als Ergebnis einer Transformation zu definieren: XML, HTML oder andere textbasierte Formate wie beispielsweise PDF. Es bietet sich an, das Ergebnis einer Transformation dem `ServletOutputStream` des `Response` Objektes innerhalb des Servlets zu übergeben und auf diese Weise einen angeforderten Sendeplan an den Client zu übertragen. Das `ServletOutputStream` Objekt kann als `Result` Parameter dem `Transformer` übergeben werden.

Zusätzlich ergeben sich aus den Anforderungen noch zwei weitere Funktionen, die die Handhabung des Sendeplans und die Ausführung von Transformationen zur Darstellung unterschiedlicher Ansichten erleichtern sollen:

- *Die Berechnung der Sendezeiten*
  Wie in den Anforderungen (FA 09) beschrieben, muss innerhalb eines Sendeplans überprüft werden, ob evtl. Fixzeit-Verletzungen auftreten. Da diese Informationen nicht direkt in den XML Daten des angeforderten Sendeplans stehen, aber in der erzeugten Ansicht dargestellt werden sollen, muss diese Funktion in einer XSLT Transformation umgesetzt werden.

- *Zwischenspeichern des Sendeplans mit den korrigierten Sendezeiten*
  Da die Berechung der Sendezeiten durchaus einige Zeit dauern kann, soll dieser Vorgang für jeden angeforderten Sendeplan nur einmalig und getrennt von der Erzeugung der Darstellung erfolgen. Das Ergebnis der Berechnung ist ein neues XML Dokument, das als Vorlage für alle Darstellungs-Transformationen dient. Diese Vorlage wird für die Dauer einer Anwender-Sitzung zwischengespeichert.

Im nachfolgenden Klassendiagramm werden die wesentlichen Klassen gezeigt, die vom `TransformServlet` verwendet werden:

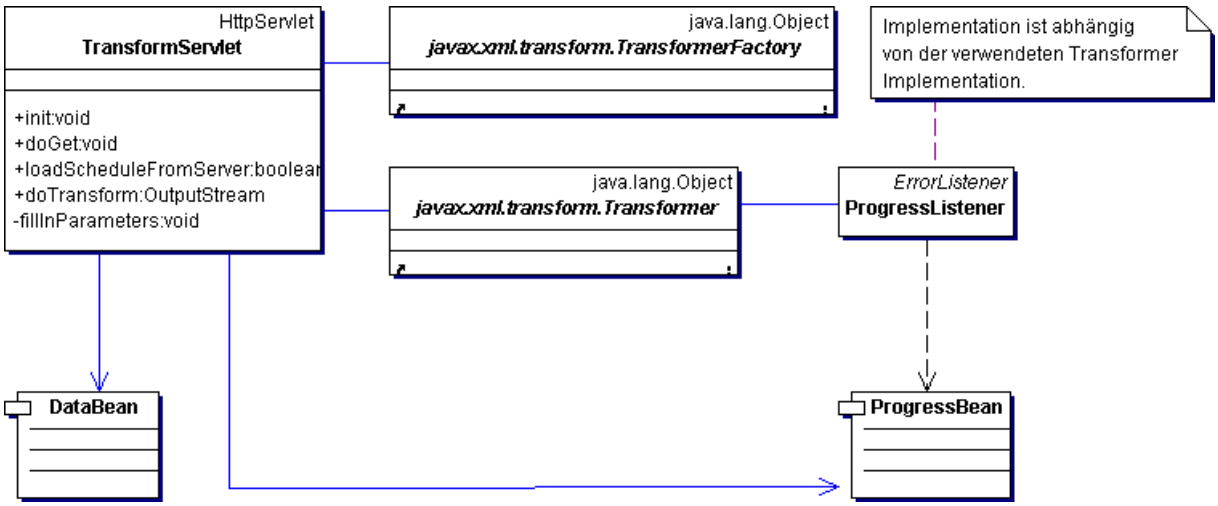

Abbildung 4-3 Verwendete Klassen in TransformServlet

`Transformer` und `TransformerFactory` sind Bestandteile der J2EE, die die Einbindung unterschiedlicher Prozessor-Implementationen zur Laufzeit ermöglichen. Dem `Transformer` kann ein implementierter `ErrorListener` zugewiesen werden, der Meldungen des Prozessors abfängt. Diese Meldungen werden in `ProgressBean` geschrieben und können dort von anderen Komponenten ausgelesen werden. Das `TransformServlet` kann als Controller betrachtet werden, der auf zwei Model Komponenten zugreift: Zum einen wird die Aktualisierung der Sendeplandaten in Form eines Aufrufs der SchedEx Komponente veranlasst und zum anderen werden Parameter aus `DataBean` abgefragt, die vom Controller der Benutzeroberfläche gesetzt werden, um das Anzeigen der von Anwender geforderten Version des Sendeplans zu veranlassen. Der Sendeplan mit den korrigierten Sendezeiten wird in `DataBean` zwischen gespeichert.

Zur Konfiguration wesentlicher Parameter, die im Servlet verwendet werden - etwa der URL des SchedEx Servlets oder die Verzeichnisse, in dem die XSLT Stylesheets liegen - können entsprechende Einträge als InitParameter in der `web.xml` der Webanwendung gemacht werden. Diese Parameter werden bei der Initialisierung des Servlets in der Methode `init()` ausgelesen.

Das nachfolgende Sequenz-Diagramm zeigt den prinzipiellen Ablauf für das Anzeigen eines Sendeplanes.

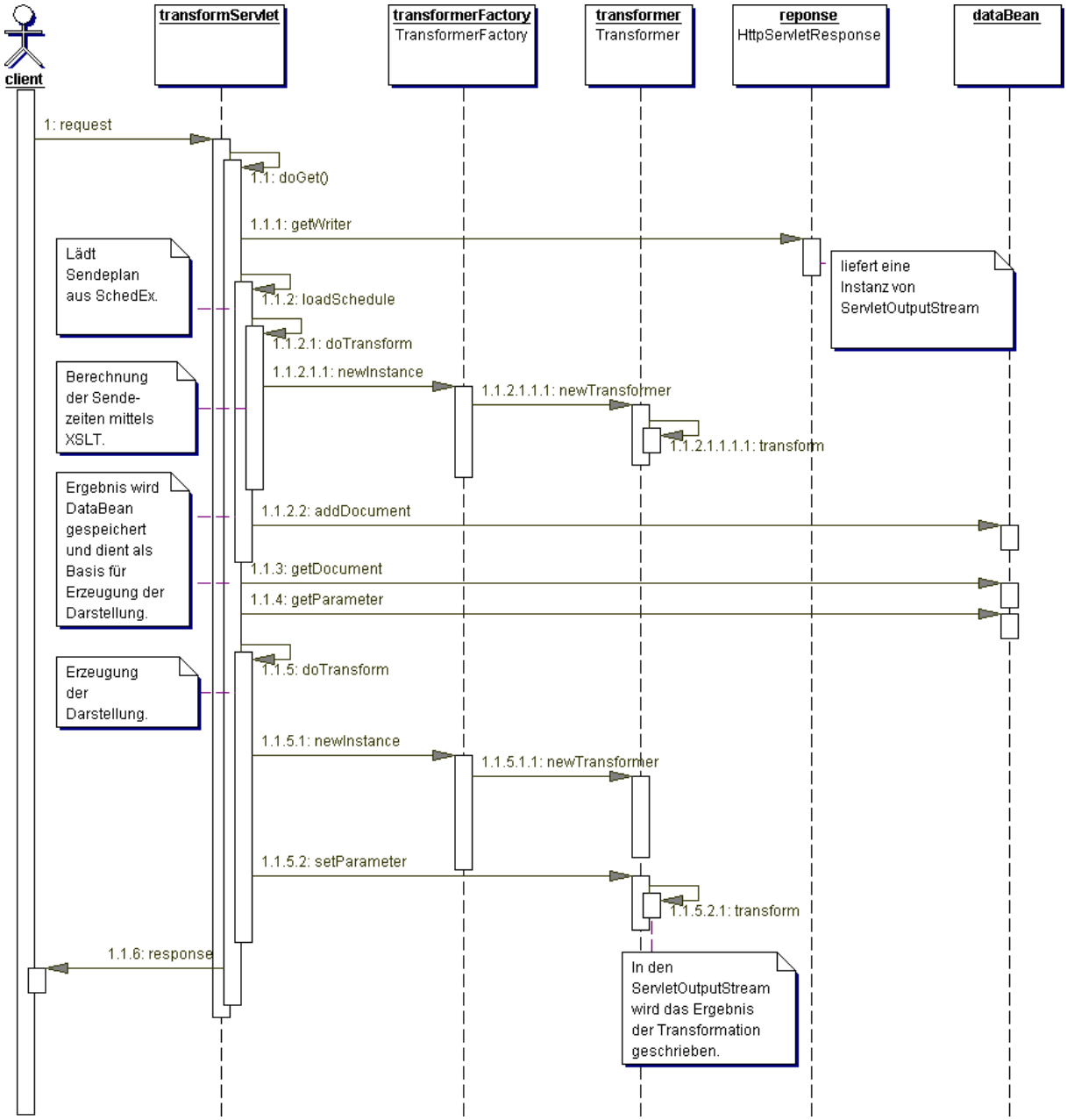

Abbildung 4-4 Prinzipieller Ablauf zur Darstellung eines Sendeplans

4.3.2. Die Benutzeroberfläche

Die Verwendung des MVC Patterns

Für den Entwurf von Benutzeroberflächen und deren Anbindung an zugrunde liegende Datenhaltungsschichten gibt es bereits bewährte Muster, die die Trennung einzelner Schichten sowie deren Kommunikation untereinander beschreiben. Das *Model-View-Controller* (MVC) Pattern ist sicherlich das bekannteste davon und ist auch mit den Techniken von J2EE umzusetzen. Ein dementsprechend modifizierter Ansatz ist das so genannte *Model 2*, das zur Umsetzung Java Server Pages (JSP) und Servlets verwendet.

Im Konzept des *Model 2* werden Servlets bzw. JSPs als Controller verwendet, die Benutzereingaben entgegen nehmen und diese auswerten. Als View werden JSP verwendet, die bei der Erzeugung der

Ansicht auf JavaBeans als Model zugreifen und die darin abgespeicherten Eigenschaften auslesen. Verändert wird das Model vom Controller, der auch eine Aktualisierung der View veranlasst.[34] Dieses Konzept lässt sich sehr gut für die Benutzerführung im Menü des SchedEx Viewers einsetzen und wird in der folgenden Abbildung dargestellt:

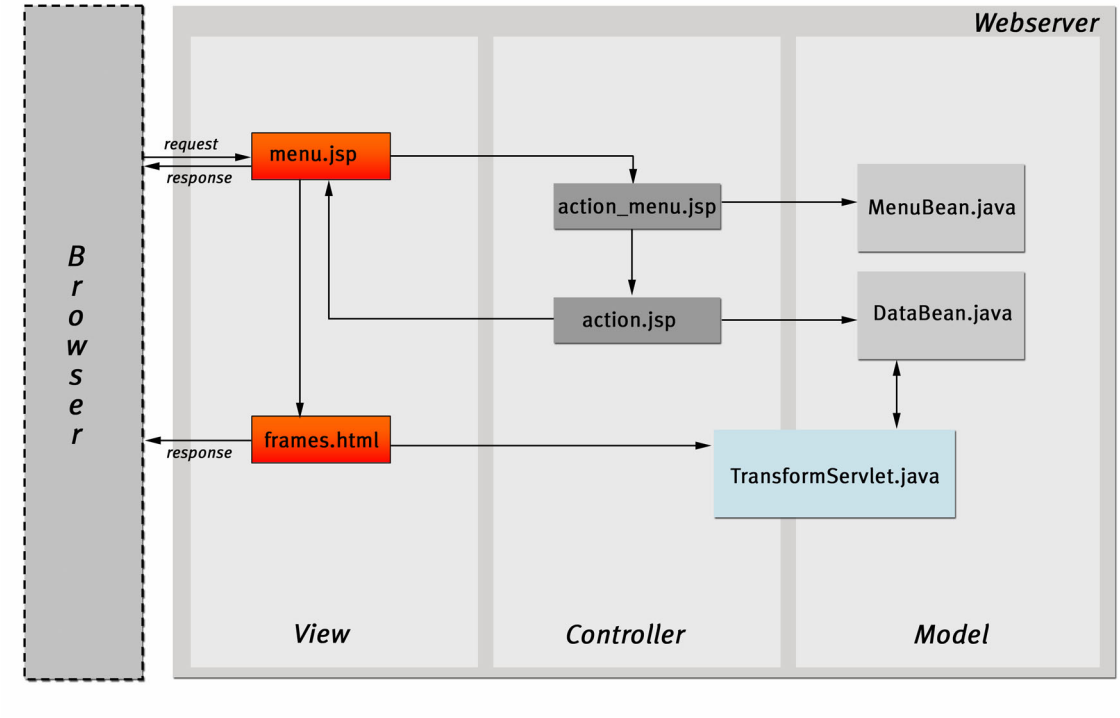

Abbildung 4-5 Das MVC Pattern bei der Benutzeroberfläche

Der Anwender nimmt Veränderungen in der View (durch Formulare) vor, die an den Controller weitergeleitet werden. Für Änderungen des Menüs ist `action_menu.jsp` zuständig (um etwa bestimmte Menüelemente anzuzeigen oder auszublenden), während `action.jsp` bei einer Anforderung oder Änderung eines Sendeplans angesprochen wird. Die Controller veranlassen eine entsprechende Änderung des Models und benachrichtigen die View Komponente, die den aktualisierten Zustand beim Model abruft und für den Anwender darstellt.

Beim Entwurf des Viewers ist nochmals eine Unterteilung der View und Controller Instanzen zu bemerken. Dabei werden die Controller des Menüs und die des Sendeplan-Models voneinander getrennt, ebenso wie die Ansicht des Menüs von der Ansicht der Sendeplandaten. Das hat zum Vorteil, dass die Controller jeweils nur für ein Merkmal des Viewer zuständig sind und somit bei der späteren Implementierung getrennt voneinander entwickelt werden können. Zudem wird die Darstellung eines Sendeplans in mehrere Teilsichten unterteilt (siehe dazu Abschnitt 4.3.3). Dazu können HTML Frames eingesetzt werden, für deren Aktualisierung ein Mechanismus gefunden werden muss, der konsequenterweise in einem Controller umgesetzt werden sollte.
Als weitere Besonderheit fällt auf, dass das `TransformServlet` sowohl Controller als auch Model Eigenschaften besitzt. Das liegt daran, dass es eine Zwischenschicht zwischen der View und dem eigentlichen Model (den Sendeplandaten in der Datenbank) darstellt. Für die View stellt es sozusagen das Model dar; dennoch es hat aber noch einige Logik-Bestandteile, die die Erzeugung der jeweils angeforderten Sicht steuern.

[34] [Model2]

Für die Implementierung der Logik in den Controllern können sowohl Servlets als auch JSPs eingesetzt werden, während JSPs für die Umsetzung der View in Frage kommen und lediglich Zugriffe auf das Model implementieren müssen, um den eigentlichen Inhalt für den Anwender darzustellen.

Darstellung dynamischer Inhalte

Werden in JSPs dynamische Inhalte dargestellt, so werden diese erst wieder aktualisiert, wenn der entsprechende Bereich des JSPs aktualisiert wird. Bei Aktualisierungen im Bereich von Minuten oder als Reaktion auf Benutzeranfragen ist dies performant mit JSP umzusetzen. Sollen aber Informationen im Intervall von wenigen Sekunden aktualisiert werden und dies über längere Zeit hinweg, so müssen andere Methoden gefunden werden. Eine Möglichkeit besteht darin, Java Applets zu verwenden. Diese lassen sich zwar sehr gut an eine Infrastruktur anbinden, die ohnehin aus Java Technologien besteht; der große Nachteil von Applets ist jedoch deren schlechte Performance im Vergleich zu anderen Technologien, die dazu geführt hat, dass die Akzeptanz Applets generell negativ bewertet werden kann. Als Alternative soll deswegen beim Entwurf dieser Anwendung Macromedia Flash verwendet werden. Die Einbindung von Flash Applikationen erfolgt über die Installation eines frei verfügbaren PlugIns, mit dem Vorteil, dass die Performance von Flash Anwendungen wesentlich besser ist als von Applets, die vergleichbare Funktionen ausführen. Zudem ist der Anteil der zeitkritischen Inhalte beim SchedEx Viewer überschaubar, so dass der Aufwand bei Entwicklung als relativ gering betrachtet werden kann. Hinzu kommt, dass der Viewer zunächst nur innerhalb der ProSiebenSat.1 Gruppe verwendet werden soll, so dass eine einheitliche Infrastruktur auf Seiten der Anwender (Internet Explorer mit Flash PlugIn) vorausgesetzt werden kann.

Flash Anwendungen sollen zunächst zwei Funktionalitäten abdecken:

1. *Die Anzeige der momentan ausgestrahlten Sendeplanelemente des aktuellen Sendeplans.* Diese Funktion wird als ItemTracker bezeichnet. Dabei kann ausgenutzt werden, dass Flash bereits Funktionen zur Behandlung von XML bietet. Somit kann über eine erneute Transformation auf dem Server eine XML Version eines Sendeplans erzeugt werden, die an die Flash Anwendung auf der Clientseite übertragen wird und die nur die Informationen enthält, die benötigt werden, um das aktuelle Element zu ermitteln und anzuzeigen.

2. *Die Anzeige des Fortschritts des Transformationsprozesses.*
   Zur Fortschrittsanzeige kann Flash Remoting MX eingesetzt werden. Remoting MX ist eine Erweiterung von Flash, die ebenfalls von Macromedia angeboten wird und die die Anbindung von Flash Anwendungen an Java Application Server ermöglicht. Bei Flash Remoting MX handelt sich um eine Macromedia-eigene Umsetzung von Web Services, die zur Übertragung ein an SOAP angelehntes Protokoll verwenden[35]. Dazu muss auf dem Applikationsserver das so genannte Flash Gateway in Form einer jar Datei installiert werden, das für die Kommunikation zwischen Flash Anwendungen auf der Client Seite und dem Java Server zuständig ist. Dies geschieht durch die Konvertierung von Flash-eigenen ActionScript Objekten in Java Objekte und umgekehrt. Um Forschrittsmeldung im Browser des Anwenders anzeigen zu können, müssen diese zunächst auf dem Server während einer Transformation erzeugt werden. Dies kann wie folgt geschehen:

[35] [FRMX]

a. Innerhalb von XSLT muss festgestellt werden können, wie weit die Transformation fortgeschritten ist. Dies kann z.B. dadurch erfolgen, dass die Nummer des aktuell bearbeiteten Elements mit der Gesamtanzahl aller Elemente eines Sendeplans verglichen und daraus der relative Anteil berechnet wird. Das Ergebnis der Berechung kann über das Element `<xsl:message>` an den XSLT Prozessor weitergegeben werden.

b. Der Prozessor gibt `<xsl:message>` Elemente standardgemäß in der Konsole aus, in der er ausgeführt wird. Es können allerdings eigene `Listener` Klassen implementiert werden, die Meldungen des Prozessors abfangen. Über diesen Mechanismus können Forschrittsmeldungen beispielsweise in eine JavaBean geschrieben werden, die durch Flash Remoting MX auch von Flash Anwendungen angesprochen werden kann. Auf diese Weise kann also die auf dem Server erzeugte Fortschrittsmeldung an den Client übertragen werden. Dieser Prozess lässt sich schematisch wie folgt abbilden:

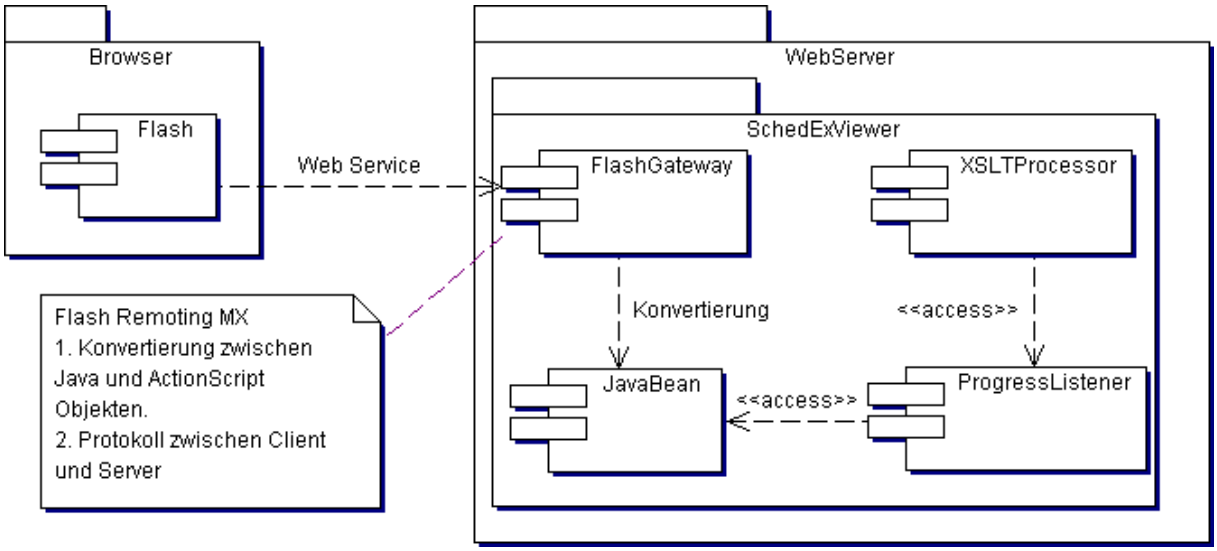

Abbildung 4-6 Die Verwendung von FlashRemoting MX

4.3.3. Die Erzeugung der Sendeplan-Darstellung

Transformationsschicht und Formatierungsschicht

Prinzipiell muss bei allen Überlegungen bezüglich XSLT die Tatsache berücksichtigt werden, dass das zugrunde liegende Datenmodell noch Änderungen unterworfen sein kann, die in den Stylesheets entsprechend berücksichtigt werden müssen. Insofern bietet sich eine Aufteilung innerhalb des Entwurfs der Transformationsmechanismen gemäß der Aufteilung von XSLT an, wie sie in Kapitel 2 beschrieben ist. Damit ist die Unterteilung in die Schritte Neustrukturierung und Formatierung der neuen Struktur gemeint. Das Ziel dieser Trennung ist es, alle Transformationen, die direkt auf die XML Datenstruktur eines Sendeplans zugreifen, an einer zentralen Stelle zu halten und zu bündeln und weitergehende Transformationen, die auf die neuen Daten zugreifen und diese in eine entsprechend formatierte Ausgabe umwandeln, davon zu trennen. Es soll also eine Kapselung nach Funktionalität ermöglicht werden und eine Zwischenschicht zwischen den ursprünglichen Daten und der Darstellung eines Sendeplans eingerichtet werden. Das Prinzip dieser Schichten ist in Abbildung 4-7 dargestellt.

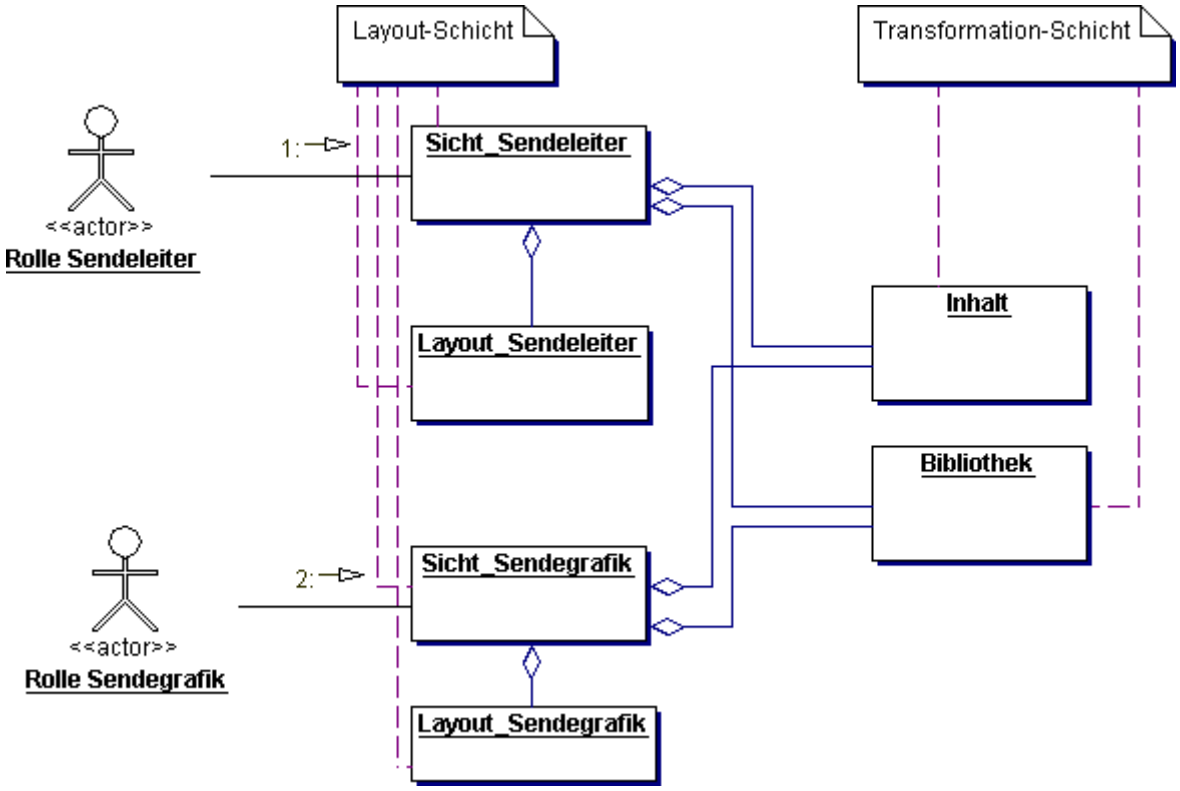

Abbildung 4-7 Aufteilung von Funktionalitäten

Damit wird bei der Erzeugung unterschiedlicher Sendeplan-Ansichten prinzipiell nur auf unterschiedliche Komponenten der Formatierungsschicht zugegriffen. Diese Schicht besteht zunächst aus einem Stylesheet, in dem angegeben wird, welche Stylesheets der Transformationsschicht bzw. welche Regeln daraus zur Erzeugung der Sicht angewendet werden sollen. Mit den Stylesheets der Layoutschicht erfolgt also eine Konfiguration einer rollenspezifischen Ansicht. Nur die Stylesheets der Transformationsschicht greifen direkt auf die XML Struktur eines Sendeplans zu und müssen das Datenmodell kennen.

Aufteilung einer Ansicht

Aus der Tatsache, dass in einem Sendeplan zumeist alle Sendeelemente enthalten sind, die an einem Tag ausgestrahlt werden und viele dieser Elemente eine Dauer von wenigen Sekunden haben, lässt sich folgern, dass die Menge der Daten zunächst in irgendeiner Weise gefiltert werden muss, da sonst eine halbwegs übersichtliche Darstellung kaum möglich ist. Daher bietet sich eine Aufteilung der Darstellung eines Sendeplans an. So kann in einer Art Auflistung aller Elemente die wesentliche zeitliche Abfolge der Elemente entnommen werden, während zusätzlich Information getrennt davon und evtl. erst nach Anforderung durch den Anwender angezeigt werden können.

Aus den Überlegungen zur Benutzeroberfläche und der Art und Weise, wie Informationen dargeboten werden (nämlich in bestimmte Kategorien gegliedert), ergibt sich für XSLT, dass einer Benutzerrolle eine spezifische Konfiguration einzelner Teilsichten angeboten wird, die alle relevanten Informationen enthalten und bei Bedarf getrennt voneinander betrachtet oder aktualisiert werden können. Diese Teilsichten werden von einem entsprechenden Satz von Stylesheets erzeugt, die in der Layoutschicht liegen. Diese Stylesheets wiederum greifen auf Daten zu, die von der erwähnten XSLT Transformationsschicht geliefert werden.

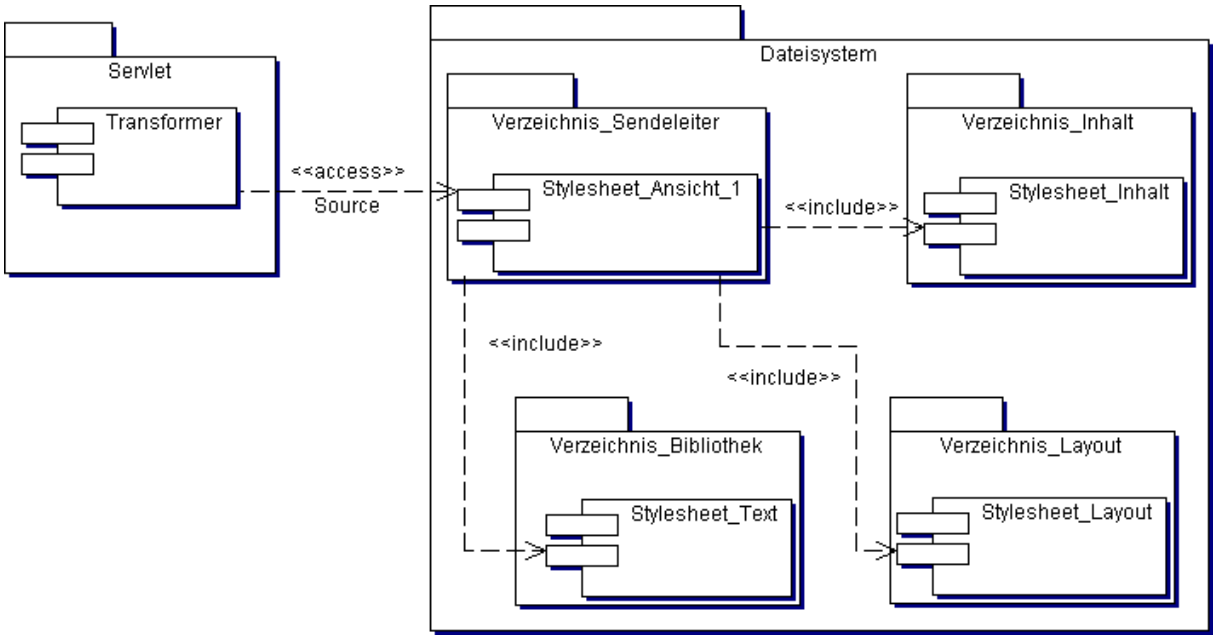

Abbildung 4-8 Aufteilung in unterschiedliche XSLT Komponenten

In Abbildung 4-8 soll der Mechanismus verdeutlicht werden, wie einerseits zwischen unterschiedlichen Anwenderrollen und andererseits zwischen Stylesheets mit unterschiedlichen Funktionen differenziert werden kann. Dabei werden die entsprechenden Stylesheets in verschiedenen Verzeichnissen abgelegt. Zum einen existiert für jede Rolle ein Verzeichnis, in dem die Stylesheets abgelegt sind, die eine Beschreibung enthalten, welcher Inhalt wie dargestellt wird (die Layoutschicht). Jedes dieser Stylesheets kann als übergeordneter Wurzelknoten repräsentiert werden, von dem aus auf andere Stylesheets zugegriffen wird (die sowohl in der Layout- als auch in der Transformationsschicht liegen können). Dies geschieht durch die Verwendung des Elements `<xsl:include>`, dessen Attribute `href` auf das einzubindende Stylesheet verweist. Durch den include Mechanismus werden mehrere Stylesheets zu einem Stylesheet zusammengefasst. Das bedeutet, dass in einem eingebundenen Stylesheet auf Templates und Variablen zugegriffen werden kann, die nicht in der gleichen Stylesheet-Datei stehen, sondern in einer externen Datei, die ebenfalls eingebunden wird. Das Prinzip der Schichten wird also über die unterschiedlichen Verzeichnisse realisiert.

Dem `Transformer` muss also als XSL Quelle der Verweis auf dieses übergeordnete Stylesheet übergeben werden. Die include Angaben innerhalb dieses Stylesheets erfolgen dann über relative Pfadangaben im Bezug auf das übergeordnete Stylesheet. Damit ist die Logik innerhalb von XSLT gänzlich unabhängig von der des Servlet und des verwendeten `Transformers`. Auf diese Weise lassen sich Änderungen gezielt an einem Stylesheet vornehmen, ohne dass dabei anderen Teile der Anwendung berücksichtigt werden müssen.

Entscheidend ist dabei die Zuordnung zwischen der Rolle eines Anwenders und dem entsprechenden Verzeichnis. Diese Zuordnung ist allerdings noch offen, da noch kein Rollenkonzept für den SchedEx Viewer bzw. innerhalb von SchedEx existiert. Die Verknüpfung von Anwender und Rolle sollte außerhalb des Viewers geschehen, die Verbindung von Rolle und entsprechendem Verzeichnis kann beispielsweise im Servlet erfolgen.

Anlegen einer XSLT Bibliothek

Mit der soeben beschriebenen Methode kann auch eine kleine Bibliothek mit Stylesheets angelegt werden, die allgemeine Funktionen in Form von benannten Templates anbietet (diese Bibliothek ist in Abbildung 4-7 und Abbildung 4-8 bereits enthalten). Diese Funktionen können Bereiche wie die Formatierung von Text oder die Konvertierung von Zeitangaben von einem Format in anderes abdecken. Diese Stylesheets können in einem extra Verzeichnis abgelegt und nach Bedarf in das vom `Transformer` aufgerufene Stylesheet eingebunden werden. Sie stellen somit Funktionalitäten zur Verfügung, die unabhängig sind von der Rolle eines Anwenders oder der gewünschten Art der Ausgabe.

Die Schwierigkeit dabei ist allerdings, dass mit XSLT keinerlei standardisierte Schnittstellen festgelegt werden können, über die eine Bibliothek eingebunden werden kann. Eine Überprüfung, ob eine Funktion bzw. ein Template auch tatsächlich vorhanden ist, findet erst zur Laufzeit im XSLT Prozessor statt. Zudem ist die Wartbarkeit der Stylesheets erschwert, wenn eine Verknüpfung von Stylesheets und Stylesheet-übergreifende Aufrufe von Templates nicht direkt im Stylesheet ersichtlich werden. Aus diesem Grund soll versucht werden, die Lesbarkeit und die Nachvollziehbarkeit von Stylesheets über die Namensgebung von Templates zu erleichtern. So kann einem Template, das beispielsweise im Stylesheet `util.xsl` definiert wird, das Präfix `UTIL` zugeordnet werden. Somit wird beim Aufruf des Templates (etwa über `<xsl:call-template name="UTIL_FORMAT_TEXT"/>` ersichtlich, dass sich dieses Template im Stylesheet `util.xsl` befindet.

4.4. Die Kommunikation der Komponenten

Bei der Kommunikation der einzelnen Komponenten müssen unterschiedliche Technologien berücksichtigt werden. Die Java Komponenten des SchedEx Viewers liegen auf dem Webserver in derselben Virtual Machine und können direkt aufeinander zugreifen. Clientseitige Bestandteile der Anwendung wie die Flash Anwendungen kommunizieren über HTTP mit dem serverseitigen Teil der Anwendung; die Synchronisation der einzelnen Frames und zwischen Flash und statischem HTML (beispielsweise bei der Aktualisierung des aktuell ausgestrahlten Elements) kann zudem über Javascript erreicht werden.

4.4.1. Klassendiagramm

Im unten stehenden Klassendiagramm sind alle wesentlichen Klassen des SchedEx Viewers abgebildet. Nicht dargestellt sind die JSPs, die als View oder als Controller dienen. Die JSP Controller greifen auf die JavaBeans zu. Die Beans dienen damit zur Kommunikation zum einen zwischen View und Controller und zum anderen zwischen den einzelnen Controllern.

Zur Simulation unterschiedlicher Rollenprofile dienen die Klassen `User` und `UserBean`. In `User` können einige Rollen implementiert werden, die von `UserBean` verwaltet und beim Anmelden überprüft werden können.

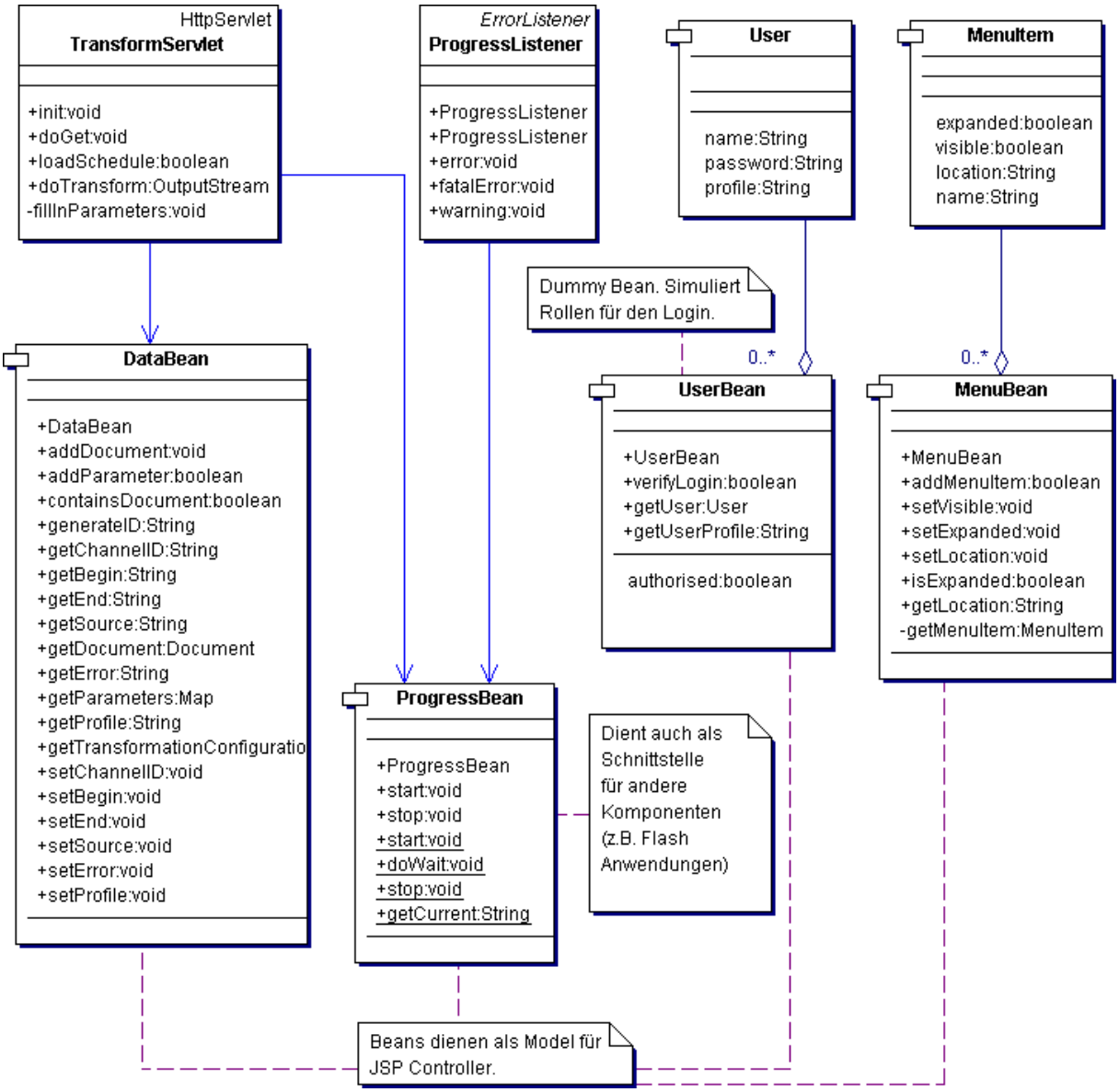

Abbildung 4-9 Klassendiagramm

4.4.2. Ablauf

Der Anwender greift über den Browser auf den Viewer in der Middleware zu. Konkret besteht ein solcher Zugriff zumeist aus der Anforderung eines Sendeplans zu einem bestimmten Sender und Sendetag. In den Controllern des SchedEx Viewer geschieht die Überprüfung der Berechtigung des jeweiligen Anwenders sowie die Gültigkeit der Anforderungsdaten. Sind alle Voraussetzung erfüllt, erfolgt die Abfrage des Sendeplans bei der Persistenzschicht über die SchedEx Komponente. Von der Abfrage bis zur Darstellung des Sendeplans im Browser des Anwenders werden innerhalb der Viewers mehrere Schritte durchlaufen:

- Prüfung der Rechte
- Prüfung der Daten der Anforderung
- Weiterleiten der Anfrage an die SchedEx Komponente
- Diese gibt als Ergebnis einer erfolgreichen Bearbeitung der Anfrage einen XML Stream zurück, der die Daten des Sendeplans enthält, die der Anforderung entsprechen.

- Dieser Stream wird innerhalb des Viewers gemäß der ursprünglichen Anforderung des Anwenders mittels XSLT in die gewünschte Darstellung gebracht. Dazu fragt das `TransformServlet` die Parameter in `DataBean` ab, die von den JSP Controllern eingetragen wurden. Bei der Transformation werden vor allem die Neustrukturierung und die grundlegende Formatierung der Daten mit XSLT abgedeckt.
- Das Ergebnis der Transformation wird an Client zurückgesendet. Im Browser des Anwenders erfolgt über die Verknüpfung mit entsprechenden CSS Stylesheets die Formatierung der endgültigen Ausgabe.

4.5. Überblick über den Gesamtentwurf

Abbildung 4-10zeigt den Gesamtentwurf des SchedEx Viewers, der alle einzelnen Komponenten der vorangegangenen Abschnitte beinhaltet und nochmals deren Abhängigkeiten aufzeigen soll:

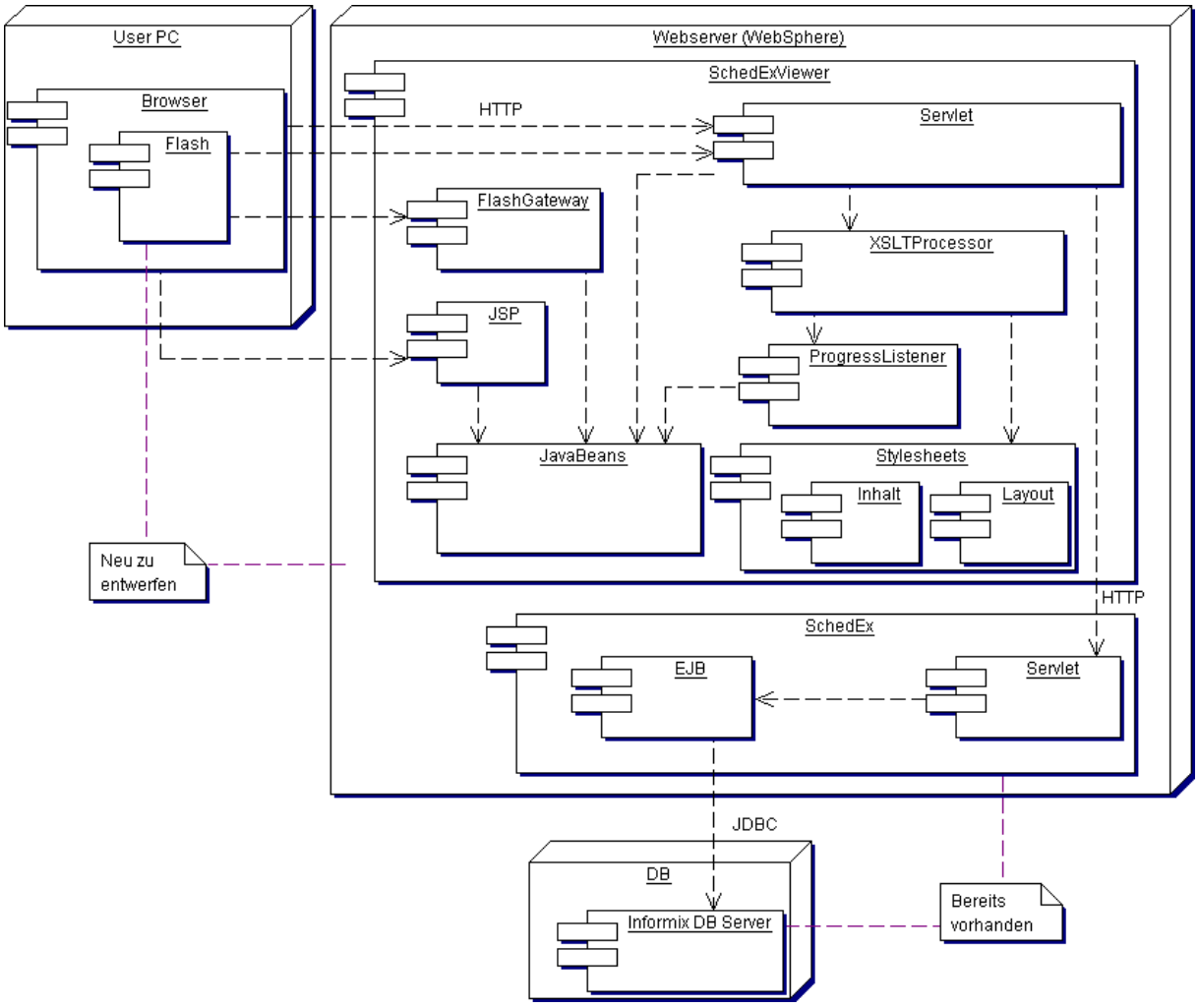

Abbildung 4-10 Der Gesamtentwurf

Dazu sei noch bemerkt, dass sich SchedEx und der SchedEx Viewer nicht auf demselben Webserver befinden müssen, da die Kommunikation zwischen beiden Komponenten über HTTP erfolgt.

4.6. Bewertung des Entwurfs

In diesem Abschnitt wird der vorgestellte Entwurf hinsichtlich seiner Stärken und Schwächen untersucht und bewertet. Dazu sei angemerkt, dass der Entwurf *eine* Möglichkeit aufzeigen soll, die geforderten Funktionen umzusetzen. Wie schon zu Beginn dieses Kapitels erwähnt, gibt es gerade im Bereich der J2EE eine Vielzahl an Möglichkeiten, eine Webanwendung zu entwerfen.

4.6.1. Stärken

- Aufgrund der Kapselung von Funktionalitäten innerhalb des SchedEx Viewers können einzelne Bereich weitgehend getrennt voneinander entwickelt werden. Hinzu kommt, dass der SchedEx Viewer nahezu unabhängig ist von SchedEx. Als Grundlage für die Entwicklung von Darstellungsformen eines Sendeplans wird das gemeinsame Datenmodell verwendet, so dass der Entwicklungsstand von SchedEx (und damit verbunden die verfügbare XML Struktur) keinen weiteren Einfluss auf den Viewer nimmt.
- Durch die Trennung der Funktionalitäten auch innerhalb von XSLT wird eine weitere und erhöhte Unabhängigkeit der einzelnen Bestandteile erreicht. Änderungen bezüglich des Datenmodells können zentral vorgenommen werden, während Änderungen einer bestimmten Rolle separat erfolgen können. Ebenso lassen sich neue Rollen ohne großen Aufwand hinzufügen.
- Dadurch dass die `web.xml` als externe Konfigurationsdatei verwendet wird, lassen sich wesentliche Änderungen der Architektur relativ einfach und zentral vornehmen, ohne dass der Code der Anwendung nochmals geändert werden müsste.
- Die Verwendung von Flash zur Umsetzung des ItemTrackers (siehe Abschnitt 4.3.2) verbessert die Performance der gesamten Anwendung. Das liegt zum einen daran, dass die Flash Anwendung einen relativ kleinen Umfang hat und zum anderen daran, dass die Ausführung der Anwendung auf dem Client erfolgt und der Server nicht zusätzlich belastet wird.[36]

4.6.2. Schwächen

- Die Verwendung unterschiedlicher Technologien erschwert die Wartbarkeit der Anwendung. Hier soll allerdings das Ziel, unterschiedliche Arten der Einbindung von XSLT aufzuzeigen, berücksichtig werden.
- Zur Kommunikation der Komponenten untereinander werden keine fest definierten Schnittstellen verwendet (abgesehen vom Datenmodell). Dadurch kann die Stabilität der Anwendung negativ beeinträchtigt werden.
- Das Rollenkonzept stellt zwar eine zentrale Anforderung an die Webanwendung, kann aber noch nicht berücksichtigt werden, da zum Zeitpunkt der Entstehung der Anwendung kein Konzept vorlag. Davon betroffen ist auch die Authentifizierung der Anwender. Das bedeutet, dass hier in Zukunft noch Änderungen erfolgen werden, die unter Umständen zu größeren Änderungen in der Anwendung führen.

[36] Die Darstellung von Fortschrittsmeldungen ist bezüglich der Verbindung von XSLT mit anderen Technologien zwar ein interessanter Aspekt, stellt jedoch keinen wesentlichen Bestandteil der Anwendung dar.

5. Die Umsetzung

5.1. Überblick

Nach den vorangegangenen Betrachtungen soll in diesem Kapitel die konkrete Umsetzung des Lösungsansatzes beschrieben werden. Dabei werden Problematiken hervorgehoben, die sich während der praktischen Umsetzung ergeben haben und die von dem in Kapitel 4 beschriebenen Entwurf abweiche. Bei den Ausführungen zur Implementation wird von außen nach innen vorgegangen: Zunächst soll die Benutzeroberfläche und deren interne Mechanismen vorgestellt werden, anschließend folgt der Blick ins "Innere" der Anwendung und die Erklärung wesentlicher Abläufe bei der Transformation.

5.2. Der Aufbau der Benutzeroberfläche – die Gliederung der Teilsichten

Da die gesamte Informationsmenge eines Sendplans zu umfangreich ist, um in einer einzigen Form übersichtlich dargestellt werden zu können, ist die Sicht auf einen Sendeplan in mehreren Teilsichten gegliedert. In der Regel bestehen diese aus einer Hauptsicht, die den Großteil der Informationen enthält und zusätzlichen Sichten, die zusätzliche Informationen enthalten und vom Anwender bei Bedarf angefordert werden können. Die Konfiguration der Gesamtsicht geschieht in Abhängigkeit der Rolle eines Anwenders und ist dem entsprechenden Bedarf einer Rolle angepasst. Abbildung 5-1 zeigt die Gliederung der Benutzeroberfläche in der Standard-Darstellung:

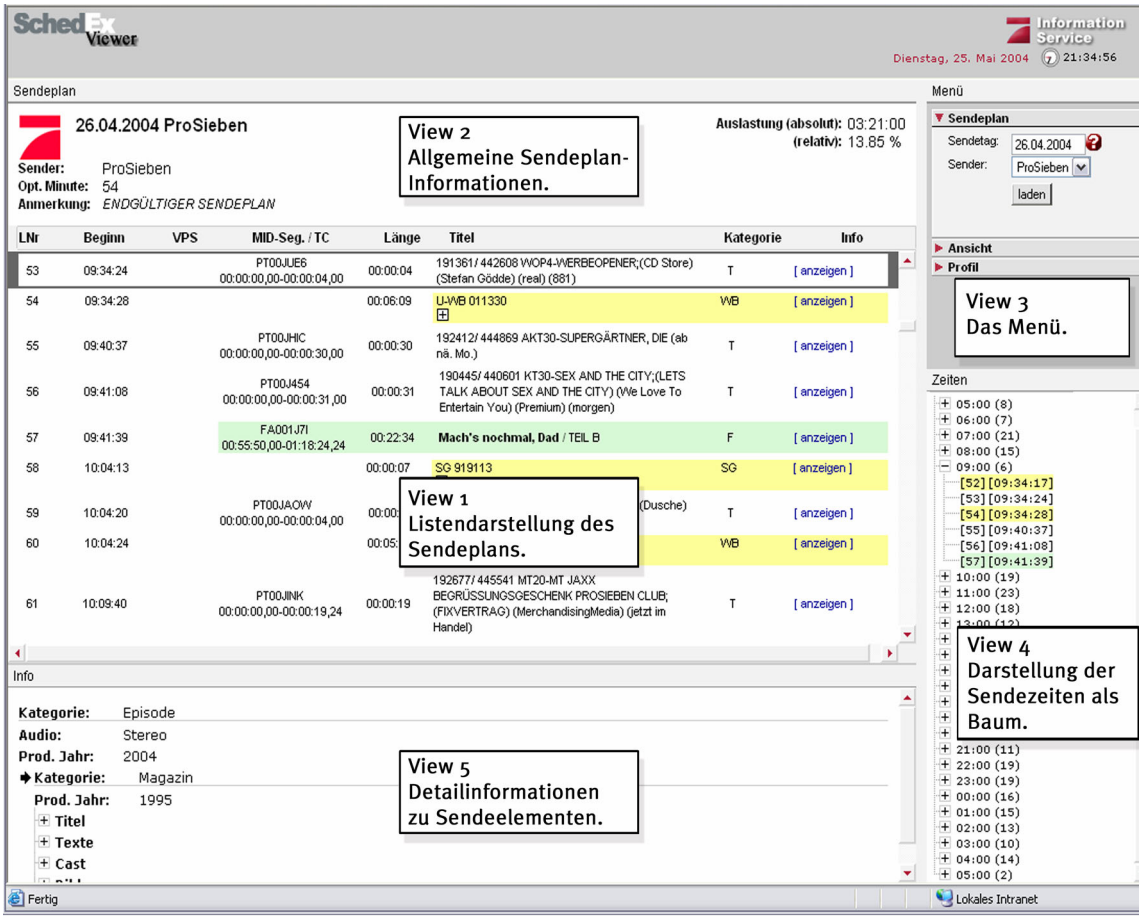

Abbildung 5-1 Der SchedEx Viewer

Für die technische Realisierung wurden HTML Frames verwendet, die über Javascript Befehle gezielt angesprochen werden können (siehe Kommunikation über Javascript in Abschnitt 5.5). Die folgende Übersicht zeigt die Hierarchie der ineinander verschachtelten Frames:

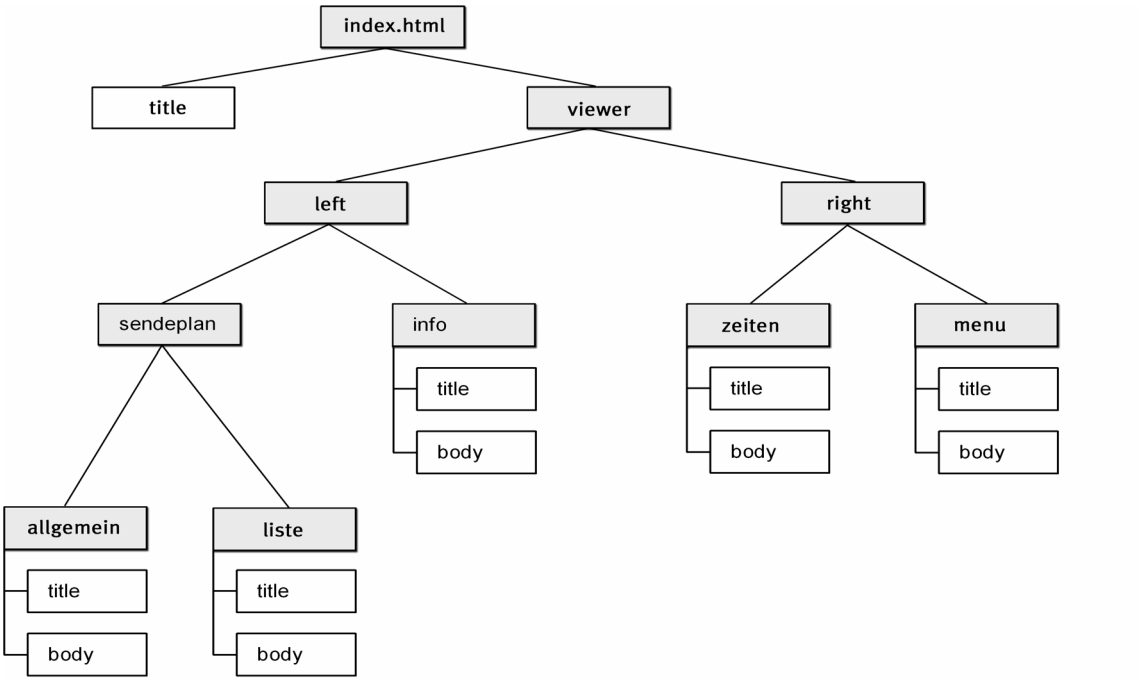

Abbildung 5-2 Hierarchie der Framesets und Frames

Grau hinterlegt sind die Framesets, die weitere Framesets oder Frames enthalten. In den `title` und `body` Elementen wird der Inhalt eines Sendeplans bzw. Teile davon dargestellt.

5.3. Der Ablauf einer Sendeplan-Anforderung

Die folgende Abbildung zeigt die konkrete Abfolge der Schritte, die nötig sind, um aus einer Anforderung eines Sendeplans durch den Benutzer eine Darstellung zu erzeugen:

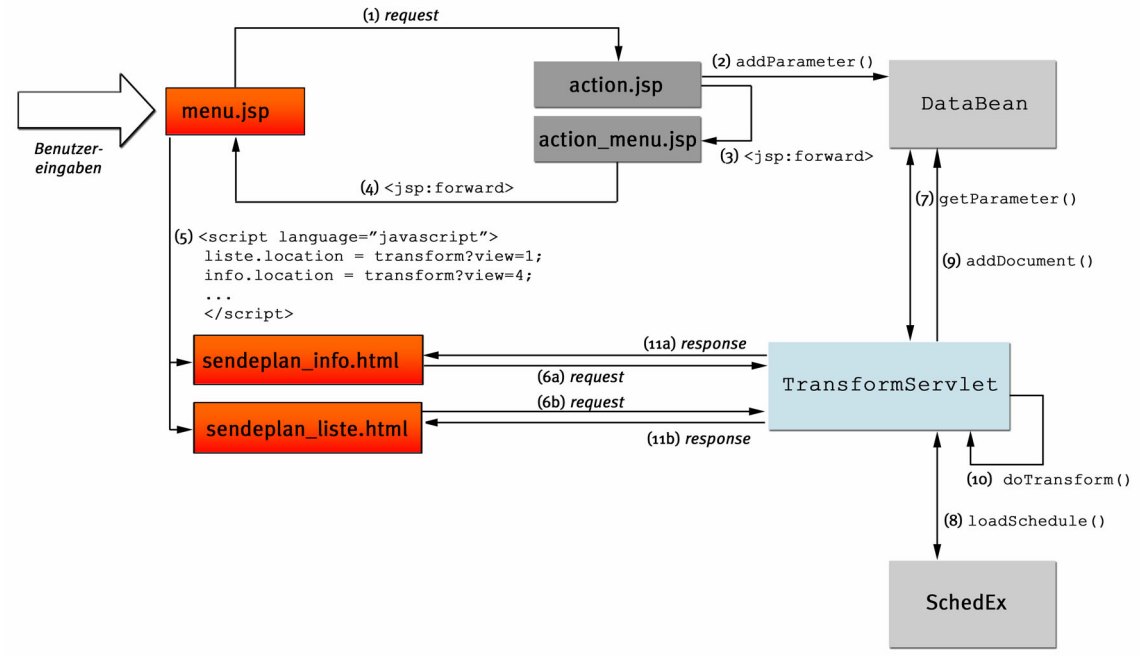

Abbildung 5-3 Ablauf einer Sendeplan-Anforderung

Die Beschreibung der einzelnen Schritte:

1. Der Anwender stellt eine Anfrage (als HTTP Request) mit der ID des Senders und dem Sendetag als Parameter.
2. Auswertung durch den Controller (action_menu.jsp)und Setzen der Parameter im Model (DataBean).
3. Weiterleiten an den Menü-Controller (action_menu.jsp). Hier wird der Status der in Punkt 2 erfolgten Auswertung abgefragt und evlt. eine Fehlermeldung erzeugt.
4. Weiterleiten an die Menü-View (menu.jsp) und Anzeigen von evtl. Fehlermeldungen.
5. menu.jsp veranlasst nun über Javascript die Aktualisierung der einzelnen Sendeplan-Ansichten.
6. Die Aktualisierung der Views erfolgt über den Aufruf des Servlets mittels URL.
7. Im Servlet werden die Parameter in der DataBean abgefragt.
8. Abfrage des Sendeplans bei SchedEx und Berechnung der Sendezeiten.
9. Nach erfolgreichem Laden des Sendeplans wird der Sendeplan mit neuen Sendezeiten in DataBean zwischen gespeichert und dient als Vorlage für alle darauf folgenden Darstellungs-Transformationen innerhalb der Session.
10. Transformation der Darstellung.
11. Über das Response Objekt wird das Ergebnis an den Browser zurück gesendet.

5.4. Verwalten mehrer Sendeplan Ansichten in einer Session

Die Parameter, die in der DataBean gespeichert werden, haben Gültigkeit innerhalb einer Session und bestimmen vor allem, welcher Sendeplan wie dargestellt wird. Öffnet ein Anwender innerhalb einer Session mehrere Browser-Fenster, um etwa unterschiedliche Sendepläne bzw. Sendeplan-ansichten parallel zu betrachten, führt dies zu unerwünschten Nebeneffekten, wenn der Anwender in einem Fenster Änderungen vornimmt, die in einem anderen nicht berücksichtigt werden sollen. Um dies zu vermeiden, bekommt jedes geöffnete Browser-Fenster bzw. jede Transformation eines Sendeplans eine eigene ID zugewiesen. Dies geschieht folgendermaßen:

1. Nach erfolgreichem Login und Auswahl des gewünschten Sendetags und des Senders, wird die Anforderung an action.jsp gesendet. Hier wird erkannt, dass ein neuer Sendeplan angezeigt werden soll und die Methode generateID() in DataBean aufgerufen. Diese liefert als ID einen aktuellen Timestamp.
2. Die übertragenen Parameter aus der Benutzeranforderung werden mit dieser ID in der DataBean abgespeichert. Konkret wird dies in DataBean mit einer HashMap realisiert.
3. Beim Weiterleiten zu action_menu.jsp bzw. menu.jsp wird die ID jeweils mit übertragen.
4. In menu.jsp erfolgt nun die Aktualisierung der eigentlichen Sendeplanansicht. Wie in Abschnitt 5.3 beschrieben, erfolgt dies über einen URL Aufruf, der mittels Javascript für die betroffenen Frames ausgeführt wird. Dem URL Aufruf wird wiederum die ID als Parameter mit übergeben, so dass das TransformServlet diese als Parameter entsprechend auswerten kann und nun bei der Konfiguration der Transformation die Parameter in DataBean mit der entsprechenden ID abfragen kann.
5. Bei allen anderen Transformationen, die sich auf einen bereits angezeigten Sendeplan beziehen, wird die ID mit der Anfrage übertragen und neue Parameter unter Verwendung der ID in der DataBean hinzugefügt. Es erfolgt die Bearbeitung der Schritte 3 und 4.

Die ID kann als Parameter ebenfalls an XSLT übergeben werden und so in die erzeugten HTML Seiten eingebunden werden. Soll von dieser HTML Seite aus eine Transformation gestartet werden (etwa über einen Javascript Befehl), kann die ID wieder an das Servlet übergeben werden.

5.5. Die Kommunikation über Javascript

Wie bereits erwähnt, ist die Oberfläche der Anwendung in mehrere Frames unterteilt. Um zum einen mehrere Frames gleichzeitig ansprechen zu können und zum anderen unterschiedliche Technologien wie Java, HTML und Flash (ActionScript) miteinander zu verbinden, wird beim SchedEx Viewer Javascript als Kommunikationsschicht verwendet.

5.5.1. Die Kommunikation zwischen Java, HTML und Javascript

Innerhalb des SchedEx Viewers ist es in bestimmten Ansichten wünschenswert, dass dynamische Aspekte (etwa ein Menü) angeboten werden, um zusätzliche Informationen anzuzeigen. Damit ist meistens eine neue Transformation und somit ein Aufruf des Servlets verbunden. Um bei der Transformation auf den richtigen Sendeplan zuzugreifen, muss die ID (siehe Abschnitt 5.4) bekannt sein. Diese ID wird bei der erstmaligen Darstellung eines Sendplans erzeugt und als Parameter an die JSPs weitergegeben, die die View erzeugen. Innerhalb der Views kann diese ID an Javascript weitergegeben werden, indem in einer Seite bei jedem Neuladen eine Javascript Variable neu geschrieben wird. Listing 5-1 zeigt dieses Verfahren in menu.jsp:

```
01: <%
02:   String id = request.getParameter("id");
03: %>
04: <script language="javascript" type="text/javascript">
05:     var viewID = '<%= id %>';
06: </script>
```

Listing 5-1

Wird nun per Javascript eine Transformation gestartet, kann die ID als globale Variable referenziert und wieder an das Servlet übergeben werden:

```
top.viewer.sendeplan.liste.body.location = transform?id='+viewID+'&view=2;
```

5.5.2. Die Kommunikation zwischen Flash und Javascript

Flash Anwendungen werden mittels <object> Tags in HTML eingebunden und bekommen eine eindeutige ID zugewiesen. Damit sind sie in HTML über Javascript ansprechbar. Flash besitzt eine Schnittstelle, die es ermöglicht, eine Untermenge von Javascript Funktionen auf Flash Anwendungen auszuführen, beispielsweise die Funktion SetVariable(name,value). Damit können Javascript Variablen an Flash übergeben werden. Auf der anderen Seite ist in Flash der Aufruf von externen Javascript Methoden möglich. Dies geschieht über den ActionScript Befehl getURL(url). Der Parameter url kann dabei als Javascript Aufruf dienen, z.B.

```
getURL("javascript:callThisFunction()");
```

Die entsprechende Funktion muss in der HTML Seite verfügbar sein, in der auch die Flash Anwendung eingebettet ist.

5.6. Die Generierung der Sendeplan Ansichten

5.6.1. Verwendete Versionen von XSLT

Bei der Umsetzung des Entwurfs im Bereich der XSL Transformationen stellt sich die Frage, welche Version dabei verwendet werden soll: Seit der Veröffentlichung der offiziellen Empfehlung von XSLT 1.0 im Jahre 1999 wurde XSLT weiter entwickelt. So existiert zum momentanen Zeitpunkt ein Entwurf von XSLT 1.1, der allerdings nicht weiter entwickelt wird, aber dennoch in einigen XSLT Prozessoren zumindest teilweise umgesetzt wird. Eine interessante Neuerung in XSLT 1.1 ist u.a. die Möglichkeit, die Ausgabe in mehrere Dokumente aufzuteilen sowie eine standardisierte Einbindung von *extension functions.*[37]

Weiterhin zeichnet sich eine Empfehlung von XSLT 2.0[38] ab, die wiederum vieles von dem aufgreift, was in XSLT 1.1 bereits beschrieben wurde. XSLT 2.0 hat dabei vor allem die zusätzliche Unterstützung von XML-Schemas, die vereinfachte Bearbeitung von Zeichenketten sowie eine erleichterte Verwendung von XSLT zum Ziel. Die Abwärtskompatibilität mit XSLT 1.0 soll dabei erhalten bleiben.[39]

Da das Ziel dieser Arbeit neben der Evaluierung von XSLT auch eine möglichst lauffähige Anwendung ist, wird bei der Umsetzung der Stylesheets die Version 1.0 von XSLT verwendet, da sie eine stabile Grundlage bildet und die größtmögliche Unabhängigkeit von unterschiedlichen Implementationen von XSLT Prozessoren bietet.

5.6.2. Verzeichnisstruktur der Stylesheets

Jeder Rolle ist ein Satz von XSLT Stylesheets zugeteilt. Jedes Stylesheet, das in einem solchen Satz enthalten ist, ist für die Darstellung des Inhalts einer Teilsicht zuständig. Die Zuordnung von Stylesheet und der entsprechenden Sicht geschieht über eine Ziffer am Ende des Dateinamens des Stylesheets. Der Satz von Stylesheets einer Rolle ist in einem gesonderten Verzeichnis abgelegt. Innerhalb einer Rolle wird nochmals zwischen den Ausgabeformaten HTML und PDF unterschieden, was ebenfalls in der Verzeichnisstruktur berücksichtigt ist.

So ist beispielsweise das Stylesheet, das für die Hauptsicht (View 1) der Rolle Sendeleiter (SL) unter dem folgendem Pfad und Namen zu finden (relativ zum Context der Webanwendung):

```
xslt/SL/html/transform_schedule_1.xsl
```

Alle Stylesheets einer Rolle sind im Wesentlichen für die Konfiguration des Layouts zuständig; hier wird als beschrieben, WIE die benötigten Informationen dargestellt werden. Zusätzlich ist im Verzeichnis `display` des Ausgabeformats einer Rolle noch evtl. ein Stylesheet enthalten, in dem allgemeine Parameter für die Formatierung des Layouts spezifiziert werden können.

Das eigentliche Auslesen der Informationen aus den XML Daten des Sendeplans findet in Stylesheets statt, die allgemein für alle Rollen verfügbar sind und wiederum in getrennten Verzeichnissen abgelegt sind. Konkret sind diese Stylesheets in den Verzeichnissen `ALL` und `UTIL`

[37] vgl. [KAY01], S. 42
[38] [XSLT20]
[39] vgl. [HOL02], S. 29f.

zu finden. In ALL werden zum einen Transformationen definiert, die den Inhalt auslesen und somit auf den eigentlichen Sendeplan zugreifen müssen (die Transformationsschicht) und zum anderen für alle Rollen gültige Layout-Definitionen festgelegt (die Layoutschicht). In UTIL befinden sich Stylesheets, die allgemein benötigte Funktionen zur Verfügung stellen, etwa die Konvertierung einer Startzeit aus UTC Millisekunden in das Format HH:MM:SS oder Funktionen zur Manipulation von String (die Umsetzung einer kleinen Bibliothek). Mit der Konzentrierung der Stylesheets an einer zentralen Stelle soll erreicht werden, dass Änderungen im Datenmodell des Sendeplans (der DTD) nur in diesen Stylesheets berücksichtigt werden müssen.

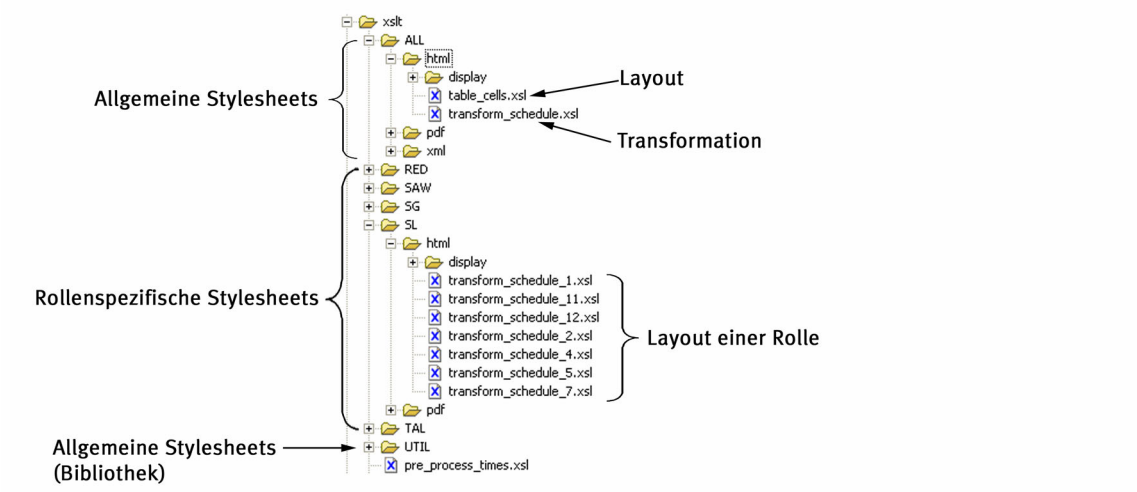

Abbildung 5-4 Verzeichnisstruktur der Stylesheets

5.6.3. Aufbau der Stylesheets

Um auf die eben erwähnten Funktionalitäten der allgemeines Stylesheets zugreifen werden können, werden diese mittels <xsl:include> in das jeweilige Stylesheet, das die Darstellung des Sendeplans übernimmt, eingebunden. Um dabei den Import etwas übersichtlicher zu gestalten haben die Templates, die aus Stylesheets der UTIL „Bibliothek“ stammen, den entsprechenden Präfix UTIL. Beispielsweise erfolgt der Aufruf einer Methode (eines Templates), die Zeitangaben konvertieren soll, in etwa so:

```
01: <xsl:include href="../../UTIL/times.xsl"/>
02: <!-- Dieser Aufruf erfolgt innerhalb des Darstellungs-Stylesheets -->
03:
04: <xsl:call-template name="UTIIL_FORMAT_TIME">
05:   <xsl:with-param name="time" select="$var_of_this_stylesheet"/>
06: </xsl:call-template>
```

Listing 5-2

Die folgende Abbildung soll das Zusammenspiel der einzelnen Stylesheets aus den unterschiedlichen "Schichten" nochmals verdeutlichen. Hier muss erwähnt werden, dass diese Schichten auf rein konzeptioneller Ebene existieren. Durch die <xsl:include> Anweisung werden zum Zeitpunkt der Transformation mehrere vereinzelte Stylesheets zu einem einzigen zusammengefasst. Beim Start der Transformation wird der Namen des übergeordneten Stylesheets an den Transformer übergeben, das die grundlegenden Definitionen zu Layout und Inhalt enthält und die erforderlichen Stylesheets einbindet.

Abbildung 5-5 Zusammenhang zwischen den einzelnen Stylesheets

5.6.4. Die Konfiguration einer Transformation

Das `TransformServlet` erwartet im Wesentlichen zwei Parameter, die bestimmen, welche Transformation ausgeführt werden soll:

1. Das Ausgabeformat (`ouput`)
2. Die Teilsicht (`view`)

Um auf Parameter wie Datum, Sender oder das aktuelle Benutzerprofil zugreifen zu können, die von den JSP Controllern in `DataBean` gesetzt werden, muss zudem noch eine `ID` übergeben werden, unter der diese Parameter abgespeichert wurden. Anhand der Parameter `view`, `output` sowie `profile` werden das Verzeichnis und der Dateiname des zu verwendenden Stylesheets bestimmt.

Als weiterer Parameter kann dem Servlet noch der Namen des XSLT Prozessors übergeben werden, der für diese Transformation verwendet werden soll. Als Standard Prozessor wird Saxon verwendet, da sich diese Implementation in der Praxis als deutlich performanter im Vergleich zu Xalan erwiesen hat. Der Prozessor wird über JAXP als System Eigenschaft gesetzt, bei der anschließenden Instantiierung des `Transformers` über `TransformerFactory` wird dabei die zuvor gesetzte konkrete Implementierung verwendet:

```
01: private static final String SAXON = "com.icl.saxon.TransformerFactoryImpl";
02: //…
03: String processor = SAXON;
04: System.setProperty("javax.xml.transform.TransformerFactory",processor);
```

Listing 5-3

In Abhängigkeit des angeforderten Ausgabeformats (HTML, XML oder PDF) wird der `ContentType` des `response` Objekts mit gesetzt. Handelt es sich dabei um HTML oder XML, ist es ausreichend, als

das ServletOutputStream Objekt als Ergebnis der Transformation anzugeben, wie in Listing 5-4 zu sehen ist:

```
01: public void doGet(HttpServletRequest request, HttpServletResponse response)
02: throws ServletException, IOException
03: {
04:   // …
05:   String o = request.getParameter("output");
06:   int output = Integer.parseInt(o);
07:   ServletOutputStream writer = response.getWriter();
08:   Result result = new StreamResult(writer);
09:   // Deklaration von xslSource und xmlSource
10:   doTransform(xslSource,xmlSource,result,output);
11: }
12:
13: public OutputStream doTransform(Source xslSource, Source xmlSource,Result
14: outResult, int output) throws TransformerConfigurationException,
15: TransformerException, IOException
16: {
17:   //…
18:   TransformerFactory tFactory = TransformerFactory.newInstance();
19:   Transformer transformer = tFactory.newTransformer(xslSource);
20:   transformer.transform(xmlSource,outResult);
21:   //…
22: }
```

Listing 5-4

Wie in Zeile 31 in Listing 5-5 zu sehen ist, gibt die Methode doTransform() ein Objekt vom Typ OutputStream zurück. Dieses Objekt ist dann ungleich null, wenn als Ausgabeformat PDF verlangt wird und das Ergebnis der Transformation (result) entsprechend gesetzt wird (siehe Zeile 15):

```
01: public void doGet(HttpServletRequest request, HttpServletResponse response)
02: throws ServletException, IOException
03: {
04:   // Nach dem Setzen der Parameter (siehe Listing 5-4)
05:   // Aufruf der transform() Methode
06:   ByteArrayOutputStream stream = (ByteArrayOutputStream)doTransform(
07:     xslSource, xmlSource,outResult,output);
08:   if(stream!=null)
09:   {
10:     response.setContentLength(stream.size());
11:     writer.write(stream.toByteArray());
12:     writer.flush();
13:   }
14: }
15: public OutputStream doTransform(Source xslSource, Source xmlSource,Result
16: outResult, int output) throws TransformerConfigurationException,
17: TransformerException, IOException
18: {
19:   ByteArrayOutputStream buffer = null;
20:   Result result = null;
21:   switch(output)
22:   {
23:     case PDF:
24:       buffer = new ByteArrayOutputStream();
25:       Driver driver = new Driver();
26:       driver.setRenderer(Driver.RENDER_PDF);
27:       driver.setLogger(new ProgressLogger());
28:       driver.setOutputStream(buffer);
```

```
29:       result = new SAXResult(driver.getContentHandler());
30:       break;
31:     default:
32:       result = outResult;
33:       break;
34:   }
35:   // nach der der Transformation (siehe Listing 5-4, Zeile 18-20)
36:   if(buffer != null)
37:     buffer.close();
38:   return buffer;
39: }
```

Listing 5-5

Im Internet Explorer kann je nach Version nicht davon ausgegangen werden, dass das Setzen des `ContentType` ausreicht, damit ein PDF erkannt wird und das Adobe PlugIn geöffnet wird. Vielmehr wird im Internet Explorer zusätzlich die Endung der zu öffnenden Datei untersucht. Um allen Versionen das automatische Öffnen des PlugIns garantieren zu können, wird in der web.xml des SchedEx Viewers das folgende Servlet-Mapping für das `TransformServlet` festgelegt:

```
01: <servlet-mapping>
02:   <servlet-name>TransformServlet</servlet-name>
03:   <url-pattern>/transform.pdf</url-pattern>
04: </servlet-mapping>
```

Listing 5-6

Die Transformation der Druckausgabe kann nun wie folgt über Javascript gestartet werden:

```
liste.body.location = transform.pdf?view=1&output=1;
```

5.6.5. Die Berechnung der Sendezeiten

Wie in den Anforderungen bereits spezifiziert (siehe FA 09), müssen die in einem Sendeplan enthaltenen Startzeiten der Sendeelemente bei der Darstellung überprüft werden und Lücken oder Überlappungen entsprechend dargestellt werden. Bei der Überprüfung der Startzeiten wird davon ausgegangen, dass in einem Sendeplan Elemente auftreten, deren Startzeit definitiv fest ist. Dies wird dadurch ausgedrückt, dass im Element `SCHEDULE_ITEM` ein Element `START` vorhanden ist, dessen Attribut `FIX` den Wert `1` zugewiesen bekommt. Für alle `SCHEDULE_ITEM`s, die zwischen zwei solchen Elementen liegen, muss nun die Startzeit in Abhängigkeit von der Startzeit des unmittelbar vorhergehenden fixen Elements und den Längen der anderen vorhergehenden `SCHEDULE_ITEM`s ermittelt werden. Hierbei kann es zu Abweichungen von der Startzeit kommen, die bereits in einem `SCHEDULE_ITEM` eingetragen sind (Fixzeit Verletzungen). Da ein Sendeplan durchaus in einer Version in der Datenbank gespeichert werden kann, in der solche Verletzungen auftreten, und diese Verletzungen erst bei der Betrachtung des Sendeplans von Bedeutung sind, findet diese Überprüfung während der Transformation mit XSLT statt.

Bei der Berechnung der Sendezeiten stellt sich allerdings das folgende Problem: Jedes SCHEDULE_ITEM *muss* ein Längenangabe im Format HH:MM:SS enthalten und *kann* zudem noch eine Längenangabe in Millisekunden enthalten. Eine einfache Berechnung der Längen mittels der XSLT Funktion `sum()` ist nur dann möglich, wenn die Längen in Millisekunden angegeben sind. Ist dies nicht der Fall, muss zunächst die Längeangabe im Format HH:MM:SS in Millisekunden

umgerechnet werden, bevor sie zur Gesamtsumme der Längen addiert werden kann. Genau hier macht sich allerdings die Tatsache, dass in XSLT keine mehrfache Zuweisung einer Variablen möglich ist, bemerkbar (siehe dazu Abschnitt 2.3.3). Hier kommt nun das Muster *Computional Patterns* zum Einsatz, das in Abschnitt 2.3.6 vorgestellt wird. Dabei wird in jedem SCHEDULE_ITEM wie folgt vorgegangen:

1. Ermittle die Position P_A des vorhergehenden SCHEDULE_ITEMs A, dessen Startzeit fix ist.
2. Ermittle die Position P_B des ersten nachfolgenden SCHEDULE_ITEMs B, dessen Startzeit fix ist.
3. Bilde die Summe der Längen aller SCHEDULE_ITEMs, deren Position größer P_A und kleiner als die eigene Position P_E ist.
4. Prüfe nun, ob in den in 3. ermittelten SCHEDULE_ITEMs Elemente enthalten sind, die keine Längenangaben in Millisekunden enthalten.
    a. Ist dies der Fall, wird die Berechung dieser Länge über den rekursiven Aufruf des Templates `<xsl:template name="CALCULATE_TIMES">` gestartet. Dieses Template erhält als Parameter die Position des ersten Elements, das keine Längenangabe in Millisekunden enthält, die Position des letzten zu überprüfenden (P_E-1) als Abbruchbedingung sowie die bisher ermittelte Summe der Längen übergeben. Im Template wird nun die Länge eines Elements in Millisekunden konvertiert und zur übergebenen Summe addiert. Ist die Position des soeben überprüften Elements kleiner als die Abbruchbedingung, ruft sich das Template erneut auf und übergibt dabei die neue Startposition und die aktualisierte Zwischensumme.
    b. Ist die Abbruchbedingung erfüllt, so gibt das Template die ermittelte Summe zurück. Diese wird mit der in 3. ermittelten Summe und der Startzeit von Element A addiert. Damit steht die Startzeit des SCHEDULE_ITEMs fest.
5. Ist P_E=P_B-1, wird die neu ermittelte Startzeit wird mit der bereits vorhanden verglichen. Kommt es hier zu Abweichungen, wird dies als FIX_ERROR im SCHEDULE_ITEM festgehalten. Bei allen anderen SCHEDULE_ITEMs wird die neu berechnete Startzeit direkt übernommen.

Da sich die Performance einer Transformation bei steigender Anzahl an Rekursionsschritten deutlich verschlechtert, wird, wie im Entwurf (siehe Abschnitt 4.3.1) bereits erwähnt, die Berechnung der Sendezeiten in eine explizite Transformation ausgelagert und der Sendeplan mit den neu berechneten Startzeiten als Grundlage für darauf folgende Transformationen zur Erzeugung der Darstellung verwendet. Um die Performance weiter zu verbessern, wird bei der Berechnung der Sendezeiten der XSLT Prozessor Saxon verwendet, der im Vergleich zu Xalan deutlich kürzere Zeiten bei der Transformation liefert.

5.6.6. Die Druckausgabe

Zur Erzeugung einer Druckausgabe wird auf die Funktionen von XSL-FO (Formatting Objects) zurückgegriffen und als Ergebnis ein Sendeplan im Portable Document Format (PDF) geliefert. XSL-FO bedient sich einer eigenen Syntax, die ebenfalls auf der XML Syntax basiert und verwendet standardgemäß das Präfix fo. Bei den Transformationen zur Erzeugung eines PDfs sind alle Formatierungsanweisung bezüglich der Druckausgabe in der Layoutschicht enthalten; zur Darstellung bzw. Auswertung des eigentlichen Inhalts wird auf die gleichen Stylesheets (in der Transformationsschicht) zugegriffen wie bei der Erzeugung der Bildschirmdarstellung.

Vorgehen

Zunächst werden bei der Erzeugung einer Druckausgabe die wesentlichen Eigenschaften des Dokuments festgelegt. Dazu werden im `<fo:root>` Element die folgenden Eigenschaften definiert:

1. `<fo:simple-page-master>`. Hier erfolgt die Definition von Seitentypen und deren Unterteilung in die folgenden Bereiche:
   a. `<fo:region-before>` Die Kopfzeile des Dokuments
   b. `<fo:region-body>` Der Bereich zur Darstellung des eigentlichen Inhalts.
   c. `<fo:region-after>` Die Fußzeile des Dokuments.
2. `<fo:page-sequence-master>` Hier wird die Reihenfolge von unterschiedlichen Seitentypen definiert werden. Konkret wird bei der Druckausgabe nur ein Seitentyp definiert.
3. `<fo:page-sequence>` Hier erfolgt unter Verweis auf einen `page-sequence-master` die Darstellung des eigentlichen Inhalts in den jeweiligen Bereichen (1.a bis 1.c) einer Seite. Hier wird mit `<fo:table>` Elementen gearbeitet, deren Spalteninhalt von den Templates in der Transformationsschicht geliefert wird.

Wie schon in Abschnitt 2.3.5 erwähnt, ist mit der verwendeten Version von fop keine vollständige Umsetzung der XSL-FO Spezifikation möglich. Dieser Umstand muss bei der Implementierung berücksichtigt werden und führt letztendlich dazu, dass die Stylesheets zur Generierung der Druckausgabe häufigen Tests unterzogen werden, um sicher zu stellen, dass alle Anweisungen vom Prozessor erkannt und umgesetzt werden können. Ein Auszug aus der Druckausgabe eines Sendeplans findet sich im Anhang dieser Arbeit.

5.7. Die Fortschritts-Anzeige

5.7.1. Die Erzeugung von Fortschrittsmeldungen

Da die Dauer der Darstellung eines Sendeplans inklusive Berechnung der Sendezeiten durchaus im Bereich mehrer Sekunden liegen kann, ist es sinnvoll, dem Anwender den Fortschritt eines Transformationsprozesses anzuzeigen. Dem verwendeten XSLT Prozessor können dazu `Listener` Klassen zugewiesen werden, die bestimmte Meldungen des Prozessors abfangen können. Diese Meldungen werden in XSLT über das Element `<xsl:message>` produziert. Beim Start einer Transformation wird eine `Transformer` Instanz erzeugt, die einen `Listener` zugewiesen bekommt. Der `Listener` wiederum schreibt alle Meldungen, die er vom Prozessor erhält, in eine `ProgressBean` Instanz, die einmalig in einer Session instanziiert wird. Das folgende Listing ist ein Auszug aus `TransformServlet`:

```
01: progress = (ProgressBean) session.getAttribute("progress");
02: //…
01: Transformer transformer = tFactory.newTransformer();
02: transformer.setErrorListener(new ProgressListener(progress));
```

Listing 5-7

In der `Listener` Klasse wird die Methode `warning()` implementiert, die die Meldung des `Transformers` in `ProgressBean` schreibt:

```
01: public void warning(TransformerException e){
```

```
02:   String message = e.toString();
03:   progress.setMessage(message);
04: }
```

Listing 5-8

Innerhalb der Stylesheets wird zunächst die Anzahl der gesamten Elemente ermittelt. Bei jedem Element das transformiert wird, wird der relative Fortschritt berechnet und mittels `<xsl:message>` an den `Listener` weitergeleitet. Dieser speichert den Wert über einen Methodenaufruf in der Klasse `ProgressBean` ab. Auf diese Klasse kann nun von anderen Stellen zugegriffen werden.

5.7.2. Die Anzeige des Fortschritts

Wie bereits im Entwurf erläutert, wird die Anzeige des Fortschritts von einer Flash Anwendung mit Hilfe von Flash Remoting MX übernommen. Über diesen Mechanismus lassen sich die in `ProgressBean` gespeicherten Fortschrittsmeldungen clientseitig abrufen und darstellen.

Um Flash Remoting nutzen zu können, muss zum einen Gateway auf dem Webserver installiert sein, das die Anbindung an die Webanwendung realisiert. Dazu wird eine entsprechende jar Datei von Macromedia in das Ressourcen Verzeichnis der Webanwendung abgelegt. Das Gateway behandelt vor allem die Konvertierung von Java Objekten in Objekte der Flash-eigenen Programmiersprache ActionScript und umgekehrt. Zum anderen müssen auf der Clientseite ActionScript Bibliotheken in die Flash Anwendung eingebunden werden, die die Funktionalitäten wie Verbindungsaufbau und Serialisierung von ActionScript Objekten bereitstellen. Diese Bibliotheken sind im Umfang von Flash Remoting MX enthalten.

Greift das Flash Gateway auf Java Objekte zu, wird bei jedem Methodenaufruf in Flash ein neues Java Objekt erzeugt. Somit würden beim Abfragen der Fortschrittsmeldungen stets neue `ProgressBean` Instanzen erzeugt, die keine entsprechenden Meldungen enthalten können. Um dies zu vermeiden, wird die Klasse `ProgressDelegate` zwischen Flash Gateway und `ProgressBean` geschaltet. `ProgressDelegate` erhält bei einer Transformation eine Instanz von `ProgressBean`, deren Referenz in einer statischen Variablen abgespeichert wird, so dass alle `ProgressDelegate` Instanzen stets auf dasselbe Objekt zugreifen und die darin enthaltenen Meldungen über das Gateway weiter an die Flash Anwendung im Client weiter vermitteln können.

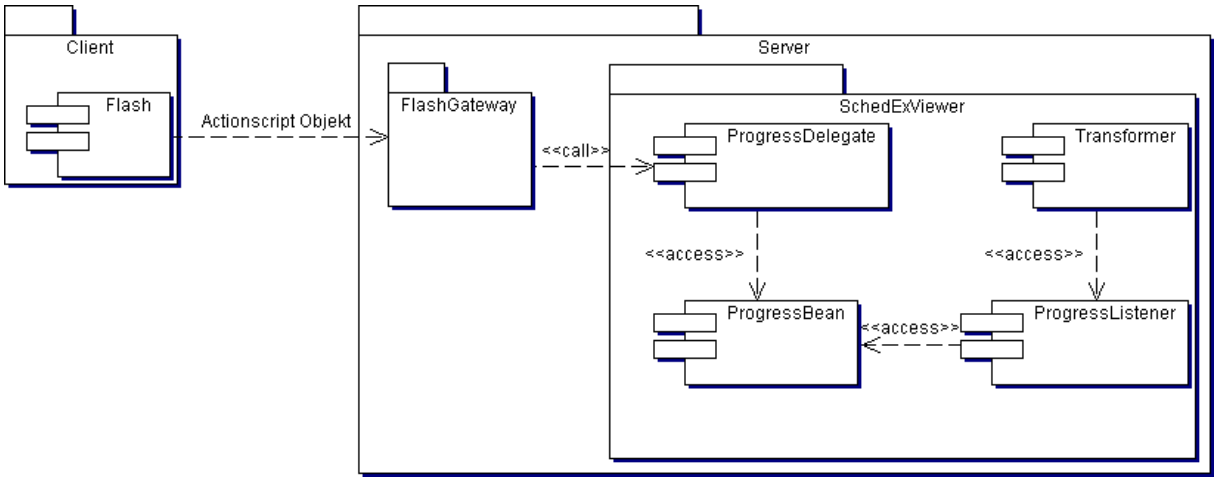

Abbildung 5-6 An der Fortschrittsanzeige beteiligte Komponenten

Dem Gateway muss mitgeteilt werdeb, welchen Dienst (welche Java Komponente) es unter welcher URL erreichen kann. Dies wird in diesem Fall zunächst in der Flash Anwendung festgelegt:

```
01: var serviceName = "com.prosieben.psi.procseco.schedex.viewer.ProgressDelegate";
02: var gatewayURL = "http://localhost:8080/SchedExViewer/gateway":
03: NetServices.setDefaultGatewayUrl(gatewayURL);
04: var gatewayConnnection = NetServices.createGatewayConnection();
05: var progressService = gatewayConnnection.getService(serviceName, this);
```

Listing 5-9

Auf der Variablen `progressService` (Zeile 5) können nun die `public` Methoden aufgerufen werden, die in `ProgressDelegate` deklariert sind und damit die Fortschrittsmeldungen in `ProgressBean` abgefragt werden. Unter der Adresse in der Variablen `gatewayURL` (Zeile 3 und 4) ist das Flash Gateway in Form eines Servlets zu finden, das die Konvertierung der Objekte übernimmt. Das Gateway wird von Macromedia zur Verfügung gestellt.

Abbildung 5-7 Die Fortschrittsanzeige

Bei den ersten Tests des SchedEx Viewers hat sich allerdings gezeigt, dass keine verlässliche Fortschrittsanzeige möglich ist, wenn die entsprechende Flash Anwendung in derselben Browser Instanz gestartet wird wie der SchedEx Viewer. Wird Flash als eigenständige Anwendung gestartet und fragt parallel zur Transformation die Meldungen auf dem Server ab, so werden die Fortschrittsmeldungen wie in Abbildung 5-7 zuverlässig angezeigt.

5.8. Der ItemTracker

Mit dem ItemTracker wird dem Anwender die Möglichkeit geboten, sich das zum momentanen Zeitpunkt ausgestrahlte Element anzeigen zu lassen bzw. nähere Informationen darüber zu erhalten.

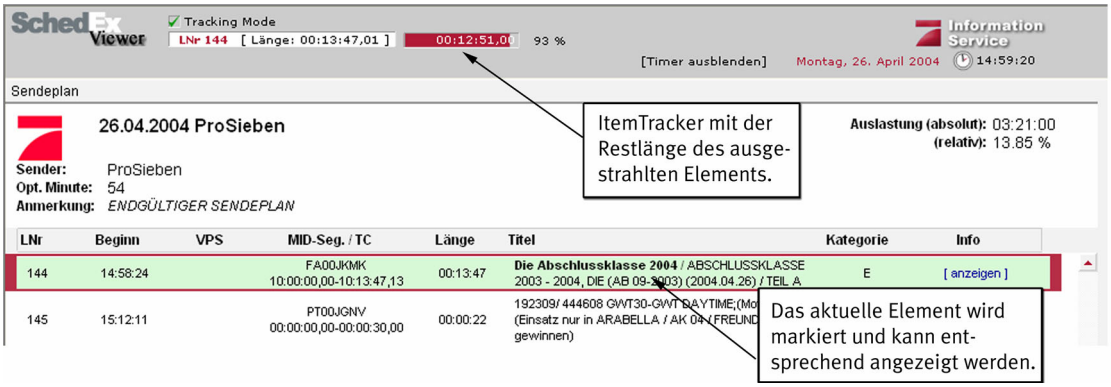

Abbildung 5-8 Der ItemTracker

5.8.1. Funktionsweise

Beim Laden eines Sendeplans wird der ItemTracker über Javascript benachrichtigt und kann wiederum über Javascript das Datum des angeforderten Sendeplans abfragen. Stimmt das Datum

mit dem tagesaktuellen Datum überein, so hat der Anwender die Möglichkeit, den ItemTracker zu starten.

Beim Start des ItemTracker erfolgt eine einmalige Anforderung des Sendeplans beim Server. Als Ergebnis der Anforderung wird ein XML Stream geliefert. Dabei kann der Vorteil genutzt werden, dass Flash eigene Objekte und Methoden zur Bearbeitung von Daten im XML Format bietet. In den XML Daten sind nur für den ItemTracker relevante Daten enthalten (beispielsweise die Startzeit und die Länge eines Elements), die Erzeugung des XML Formats erfolgt über eine erneute XSLT Transformation (zur Konfiguration einer Transformation siehe Abschnitt 5.6.4). Der ItemTracker ermittelt anhand der aktuellen Uhrzeit das momentan ausgestrahlte Element und die verbleibende Dauer. Dauer und Laufende Nummer werden im ItemTracker angezeigt, über den Aufruf von Javascript Methoden aus Flash heraus wird das aktuelle Element markiert und, wenn vom Anwender gewünscht, in der Hauptsicht angezeigt.

5.9. Der StreamingBrowser

Mit dem StreamingBrowser wird dem Anwender Zugriff auf das LowRes Material eines Sendetags zur Verfügung gestellt, das auf einem entsprechenden Streaming Server abgelegt ist (siehe FA11 in Kapitel 3). Der StreamingBrowser wird über einen Link im Menü gestartet und in einem neuen Fenster dargestellt. Der Einsatz des StreamingBrowser ist zunächst mit der Rolle RED (Redaktion) verknüpft und kann nur bei der Auswahl des entsprechenden Profils gestartet werden.

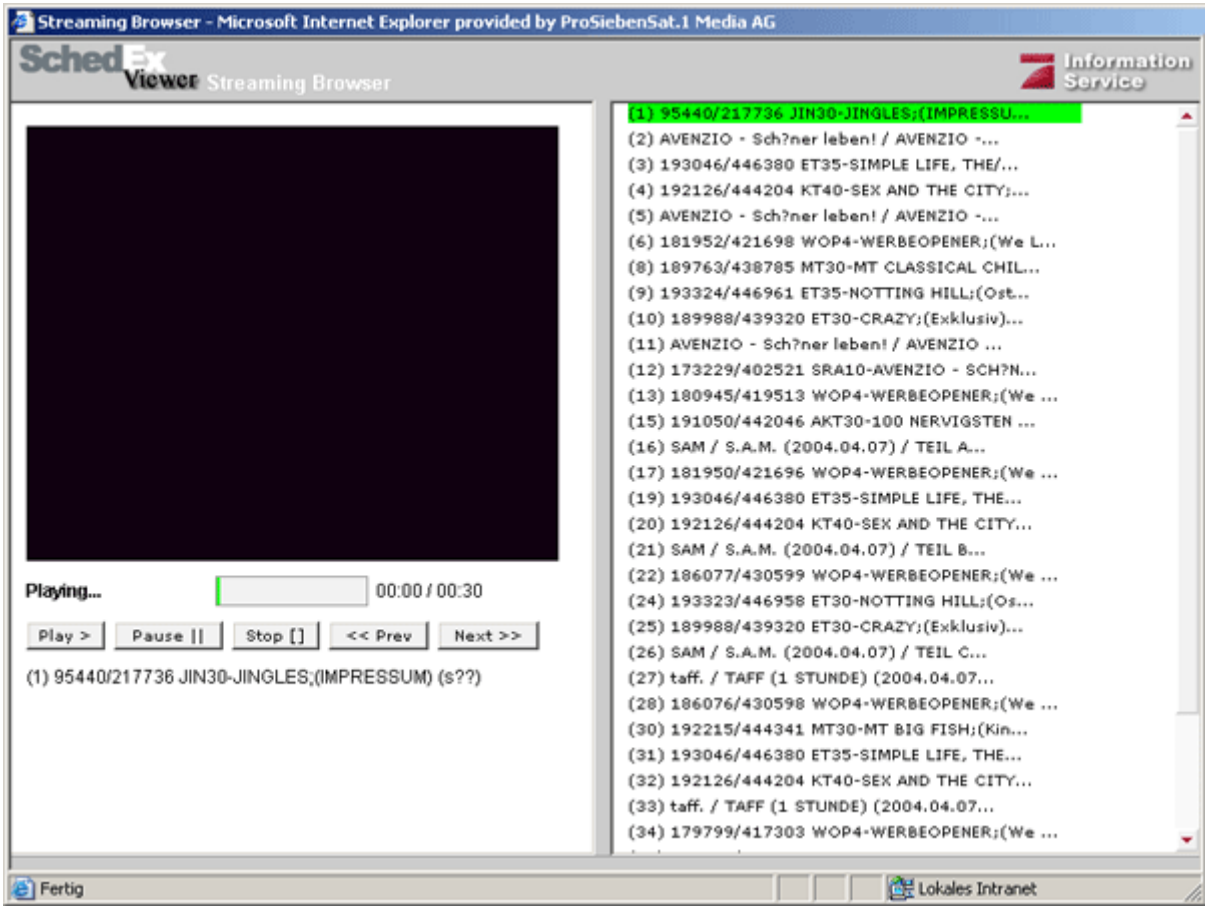

Abbildung 5-9 Der StreamingBrowser

5.9.1. Funktionsweise

Der StreamingBrowser greift auf den WindowsMedia Player (WMP) zum Betrachten von Video Streams zurück. Prinzipiell ließe die benötigte Funktionalität allein mit dem WMP abdecken; da bei der verwendeten Version 9.x des WMP die Funktionen der Playlist stark eingeschränkt sind, sobald auf Videostreams zugegriffen wird, die auf einem Server liegen, ist die Playlist Funktionalität nochmals in Javascript implementiert. Prinzipiell wird dabei erneut eine Anforderung an das `TransformServlet` gesendet, das über eine XSL Transformation aus dem aktuellen Sendeplan eine Playlist im ASX Format generiert und an den StreamingBrowser zurücksendet.

Der StreamingBrowser bedient sich JSPs und Frames bei der Darstellung. Im linken Frame wird der Player dargestellt und im rechten Frame die Playlist. Hat der Player die Playlist erfolgreich eingebunden, wird das Playback automatisch gestartet. Parallel dazu wird die Playlist als Javascript Objekt über einen Funktionsaufruf an die Playlist JSP übergeben. Dort wird sie entsprechend dargestellt und erlaubt zudem eine Änderung der Reihenfolge der Elemente. Diese Änderung wird nur lokal an dem Playlisit Objekt vorgenommen und bewirkt keinerlei Veränderungen an der Playlist des zugrunde liegenden Sendplans.

6. Fazit

Zum Schluss dieser Arbeit sollen die Erkenntnisse der einzelnen Bereich nochmals zusammengefasst werden und Ausblick auf mögliche zukünftige Änderungen und Erweiterungen gegeben werden.

6.1. Die Anwendung von XSLT

Die Bearbeitung und Erstellung von XSL Stylesheets ist aufwändig. Zwar gibt es inzwischen Tools, die eine automatisierte Erzeugung von Stylesheets bieten; bei der Verwendung von XSLT als Programmiersprache erweist sich XML Syntax, die ihre Vorteile in anderen Bereichen findet, bezüglich der Übersichtlichkeit und der schnellen Implementierung von Funktionen und Methoden als hinderlich.

Mit den in Abschnitt 2.3.6 vorgestellten Entwurfsmustern lassen auch komplexe Funktionen in XSLT umsetzen. Damit kann in XSLT eine wesentlich umfangreichere Neustrukturierung und Formatierung von XML und HTML Dokumenten betrieben werden als vergleichsweise mit CSS.

6.2. Die Einbindung

Das Angebot frei verfügbarer XSLT Prozessoren ist relativ konstant geblieben. Gerade bezüglich XSL-FO, also bei der Erzeugung einer Druckausgabe zeigt sich, dass die Spezifikation von XSL-FO sehr komplex ist. Dementsprechend wenig Implementationen, die eine volle Unterstützung dieses Standards bieten, sind zu finden. Hierbei sind kommerzielle Produkte in den Punkten Konformität und Performance durchaus im Vorteil; deren großer Nachteil sind allerdings die teilweise hohen Kosten, die mit der Anschaffung verbunden sind. Somit sind sie wohl erst dann von Interesse, wenn die Erzeugung einer Druckausgabe

- wesentlicher Bestandteil der Anwendung ist,
- die Anzahl der Anwender eine Größenordnung erreicht, in der die Performance ausschlaggebend für die sinnvolle Nutzung der Anwendung ist und außerdem
- die Art und Weise der Druckausgabe so beschaffen ist, dass ein FO Prozessor mehr Unterstützung bieten muss, als die momentan frei verfügbaren Implementationen.

Ist dies nicht der Fall, so müssen wohl bei Entwicklung von Stylesheets Abstriche hinsichtlich Komfort und Anwenderfreundlichkeit gemacht werden.

Mit JAXP werden für die Einbindung von Prozessoren in Java gute Mechanismen zur Verfügung gestellt, die eine flexible Anpassung an unterschiedliche Implementationen ermöglicht.

6.3. Die Webanwendung

Zu einem Last-Test der Anwendung ist es während der Entstehung der Arbeit nicht mehr gekommen, so dass sich Vergleiche mit anderen professionellen Frameworks nicht möglich sind. Es lässt sich allerdings feststellen, dass sich die Gesamtperformance der neuen Anwendung deutlich verbessern ließ im Vergleich zum ersten Prototyp der Anwendung. Dabei spielen mehrere Faktoren eine Rolle:

- Die Auswahl des XSLT Prozessors. Hierbei erweist der Einsatz von JAXP als sehr vorteilhaft, da damit eine Unabhängigkeit der Anwendung von Prozessoren eines bestimmten Herstellers erreicht werden kann und zudem die Verwendung zukünftiger, JAXP konformer Prozessoren unterstützt wird.
- Die Funktionalitäten, die mittels XSLT abdeckt werden sollen. Das betrifft vor allem die Rekursion bei der Berechnung der Sendezeiten - wird dieser Teil nicht mehr benötigt, weil die XML Vorlage schon dementsprechende Informationen bzw. Informationen in einen günstigeren Format bereit hält, so lassen sich die anfänglichen Ladezeiten des Sendeplans noch einmal deutlich verkürzen.
- Zudem erweist sich die Kombination von XSL und CSS als sinnvoll, da bestimmte Bereiche des Layouts nicht "hart codiert" werden müssen, sondern noch einmal ausgelagert werden können.
- Ebenso hat sich die Trennung der Transformation in unterschiedliche Bereiche (Darstellung und anfängliche Berechnung) als vorteilhaft für die Performance erwiesen.

Die Performance des SchedEx Viewers lässt sich in Zukunft weiter verbessern. Mögliche Ansätze hierzu sind unter anderem:

- Das Zwischenspeichern (Cachen) von Transformer Instanzen. Zur Darstellung eines Sendeplanswird ein Transformer zusammen mit einem Stylesheet instanitiert. Diese Instanzen können in einem Cache gehalten werden, so dass bei einer erneuten Transformation mit dem gleichen Stylesheet die bereits vorhandene Transformer Instanz verwendet werden kann.
- Bei der Berechnung der Sendezeiten kann ein weiterer Zwischenschritt eingeführt werden, in dem zunächst die Sendezeiten in ein einheitliches Format gebracht werden. Damit ließe sich die Berechung ohne Rekursion vornehmen und damit die Performance deutlich verbessern.

Bezüglich der Wartbarkeit und Erweiterbarkeit der Stylesheets ist der beschriebene modulare Aufbau von mehreren kleineren Stylesheets der Verwendung von einigen wenigen, dafür umso größeren Stylesheets, sicherlich vorzuziehen; dies gilt insbesondere dann, wenn man sich nochmals die zuvor erwähnte mangelnde Nachvollziehbarkeit von in XSLT programmierten Funktionen vor Augen hält. Hier macht sich das Fehlen einer einheitlichen Dokumentation des Quellcodes, wie es etwa mittels Javadoc bei Java Programmen möglich ist, deutlich bemerkbar.

Zusammenfassend lässt sich feststellen, dass die von XSLT gebotenen Möglichkeiten sehr umfangreich sind. In Kombination mit XML und XSLT lässt sich eine Trennung von Inhalt und Darstellung von Daten recht schnell, recht einfach und konsistent erreichen. Somit scheinen sich die Bemühungen des w3 Konsortiums um eine Verbesserung des ursprünglichen SGML und DSSSL Ansatzes gelohnt zu haben. Die Tatsache, dass sowohl XML als XSLT (und alle dazugehörigen Substandards) kontinuierlichen Entwicklungsprozessen unterworfen sind, kann als Zeichen eines weit verbreiteten Interesses an diesen Technologien gesehen werden und dementsprechend positiv bewertet werden.

Abkürzungsverzeichnis

API	Application Programming Interface
CSS	Cascading Stylesheet
DOM	Document Object Model
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
EJB	Enterprise JavaBean
FA	Funktionale Anforderung
IRI	Internationalized Resource Identifier
J2EE	Java 2 Enterprise Edition
JSP	Java Server Page
MVC	Model-View-Controller
NFA	Nicht-Funktionale Anforderung
PDF	Portagble Document Formate
PSI	ProSieben Information Service
RUP	Rational Unified Process
RED	Redaktion
SAW	Sendeabwicklung
SAX	Simple API for XML
SG	Sendegrafik
SGML	Standard Genralized Markup Language
SL	Sendeleiter
SOAP	Simple Object Access Protocoll
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WMP	Windows Media Player
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language
XPath	XML Path Language
XSL	eXtensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	eXtensible Stylesheet Language Transformations

Literaturverzeichnis

[BIE02] Bien, Adam. *J2EE Patterns. Entwurfsmuster für die J2EE*. Addison-Wesley Verlag, München, 2002

[COCOON] The Apache Cocoon Project. http://cocoon.apache.org (Letzter Zugriff: 25.05.04)

[FOP] Fop. http://xml.apache.org/fop/ (Letzter Zugriff: 25.05.04)

[FRMX] Macromedia Inc. Flash Remoting Remoting MX Konfigurationshandbuch 2. Ausgabe. September 2002. Download unter http://www.macromedia.com/support/flash_remoting/documentation/using_flash_remoting.html#deutsch (Letzter Zugriff: 25.05.04)

[GoF] Gamma, Erich; Helm, Richard; Johnson, Ralph und Vlissides John. *Design Patterns*. Addison-Wesley Verlag, 1995

[HOL02] Holzner, Steven. *XSLT – Anwendung undReferenz. XML-Transformationen, XPath, Einsatz mit Java, JSP und ASP*. Markt+Technik Verlag, München2002

[JAXP] JAXP. http://java.sun.com/xml/jaxp/index.jsp (Letzter Zugriff: 25.05.04)

[KAY01] Kay, Michael. *XSLT Programmer's Reference*. Wrox Press, Birmingham, 2001

[KRU02] Krüger, Guido. *Handbuch der Java Programmierung*. 3. Auflage. Addison-Wesley Verlag, 2002

[Model2] Seshadri, Govind. *Understanding JavaServer Pages Model 2 architecture*. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html (Letzter Zugriff: 25.05.04)

[RUP] Rational Software White Paper. *Rational Unified Process*. Download unter http://www-106.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf (Letzter Zugriff: 25.05.04)

[SAXON] Saxon. http://saxon.sourceforge.net/ (Letzter Zugriff: 25.05.04)

[STRUTS] The Apache Struts Web Application Framework. http://jakarta.apache.org/struts/ (Letzter Zugriff: 25.05.04).

[VLI03] vam der Vlist, Eric. *XML Schema*. O'Reilly Verlag, Köln, 2003

[WIL01] Wille, Stefan. *Java Server Pages*. Addison-Wesley, München, 2001

[XALAN] Xalan. http://xml.apache.org/xalan-j/ (Letzter Zugriff: 25.05.04)

[XML10] W3C. *XML 1.0*. Third Edition. http://www.w3.org/TR/2004/REC-xml-20040204/. 04.02.04

[XMLCan10] W3C. *Canonical XML 1.0*. http://www.w3.org/TR/2001/REC-xml-c14n-20010315 15.03.01

[XMLInfo10] W3C. *XML Information Set 1.0*. Second Edition. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/. 04.02.04

[XMLName11] W3C. *XML Namespaces 1.1*. http://www.w3.org/TR/xml-names11/ . 04.04.04

[XPATH10] W3C. *XML Path Language 1.0*. http://www.w3.org/TR/xpath. 16.11.99

[XSL] W3C. *XSL*. http://www.w3.org/Style/XSL/ (Letzter Zugriff: 25.05.04)

[XSLFO] W3C. *XSL-FO*. http://www.w3.org/TR/xsl/. 15.10.01

[XSLT10] W3C. *XSLT 1.0*. http://www.w3.org/TR/xslt. 16.11.99

[XSLT20] W3C. *XSLT 2.0*. http://www.w3.org/TR/xslt20/. 12.11.03

Abbildungsverzeichnis

Listings

Anhang

LNr	Beginn	VPS/PV	MID - Seg. / TC	Länge	Titel
1	05:23:39		PT003TLY 00:00:00:00 - 00:00:29:24	00:00:29	95437/ 217733 JIN30-JINGLES;(IMPRESSUM) (sachlich)
2	05:24:09		FA00JKMF 00:02:31:20 - 00:18:05:07	00:15:33	**SAM /** S.A.M. (2004.04.23)/ Teil A ..... [G] Sonderlogo: SAM / S.A.M. (2004.04.23)/ Teil A
3	05:39:43		PT00JMCE 00:00:00:00 - 00:00:30:14	00:00:30	193059/ 446414 ET30-MICHAEL MITTERMEIER - BACK TO LIFE - FOLGE 2 (heute)
4	05:40:14		PT00JBRQ 00:00:00:00 - 00:00:41:02	00:00:41	191638/ 443203 KT40-BULLYPARADE;(TRAMITZ AND FRIENDS) (heute)
5	05:40:55		FA00JKMF 00:18:50:10 - 00:35:13:14	00:16:23	**SAM /** S.A.M. (2004.04.23)/ Teil B ..... [G] Sonderlogo: SAM / S.A.M. (2004.04.23)/ Teil B
		00:00:00 ^			
6	05:57:18		PT00JBTV 00:00:00:00 - 00:00:29:23	00:00:29	191655/ 443268 ET30-KUBANER KÜSSEN BESSER;(Exklusiv) (Do) (20:15)
7	05:57:48		PT00J15L 00:00:00:00 - 00:00:50:00	00:00:48	190135/ 439845 AKT45-SPIELFILME IM MAI;(BLOCKBUSTER) (Kiss of the Dragon/ Frequency/ Scary Movie 2/ The Score/ Highlander/ Not a Girl/ Driven) (im Mai auf Pr
8	05:58:36		FA00JKMF 00:36:05:00 - 00:49:45:00	00:13:40	**SAM /** S.A.M. (2004.04.23)/ Teil C ..... [G] Sonderlogo: SAM / S.A.M. (2004.04.23)/ Teil C
9	06:12:16		FA00JKME 00:02:29:10 - 00:23:37:20	00:21:08	**taff. /** TAFF (1 STUNDE) (2004.04.23) / TEIL A ..... [G] Sonderlogo: taff. / TAFF (1 STUNDE) (2004.04.23) / TEIL A
10	06:33:25		PT00JT7S 00:00:00:00 - 00:00:04:00	00:00:04	191362/ 442609 WOP4-WERBEOPENER;(Macho) (Arabella) (real) (884)
11	06:33:29			00:02:00	U-WB 010600
12	06:35:29		PT00JU2L 00:00:00:00 - 00:00:30:00	00:00:30	191312/ 442502 SIG30-SIGNATION;(Mission Entertainment) (Overground) (real)
13	06:35:59		PT00JMCE 00:00:00:00 - 00:00:30:14	00:00:30	193059/ 446414 ET30-MICHAEL MITTERMEIER - BACK TO LIFE - FOLGE 2 (heute)
14	06:36:29		PT00J454 00:00:00:00 - 00:00:31:00	00:00:31	190445/ 440601 KT30-SEX AND THE CITY;(LETS TALK ABOUT SEX AND THE CITY) (We Love To Entertain You) (Premium) (morgen)
15	06:37:00		FA00JKME 00:24:28:08 - 00:51:08:20	00:26:40	**taff. /** TAFF (1 STUNDE) (2004.04.23) / TEIL B ..... [G] Sonderlogo: taff. / TAFF (1 STUNDE) (2004.04.23) / TEIL B
		00:02:00 ^			
16	07:03:40		PT00JT7R 00:00:00:00 - 00:00:04:00	00:00:04	191360/ 442607 WOP4-WERBEOPENER;(Mission Entertainment) (Overground) (real) (885)
17	07:03:44			00:02:58	U-WB 010610