FACHHOCHSCHULE STUTTGART
HOCHSCHULE DER MEDIEN

Fachhochschule Stuttgart
Hochschule der Medien
Studiengang Medieninformatik

Diplomarbeit

Überführung eines in XML definierten applikationsneutralen Schnittstellenmodells für Bordnetzdaten mittels XSLT in das standartisierte Graphikformat SVG

vorgelegt von:	Katharina Weimer
Matrikelnummer:	11421
vorgelegt am:	26. Mai 2004
Hochschulbetreuer:	Prof. Dr. Martin Goik
Firmenbetreuer:	Dipl.– Ing. Anton Virag

gedas deutschland GmbH

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig und in Eigenverantwortung erstellt habe. Zur Erarbeitung habe ich nur die angegebene Literatur verwendet. Zitate oder sinngemäße Wiedergabe von Inhalten dieser Literatur wurde gekennzeichnet. Diese Arbeit wurde noch nicht an einer anderen Stelle für Prüfungszwecke vorgelegt.

Stuttgart, den 26. Mai 2004 _____

Katharina Weimer

Inhaltsverzeichnis

1 Einleitung

1.1 Vorstellung gedas deutschland GmbH

Die gedas deutschland GmbH, bei der diese Diplomarbeit erstellt wurde, ist ein hunderprozentiges Tochterunternehmen des Volkswagen Konzerns mit Hauptsitz in Berlin. Der Aufgabenbereich ist breit gefächert und reicht von der Automobil- und Fertigungsindustrie über den Logistikbereich bis hin zu Produktentwicklung, Ressourcenmanagement, Vertrieb und Kundendienst. Thematisch ist die vorliegende Diplomarbeit in den Bereich des Product Lifecycle Management (PLM) einzuordnen.

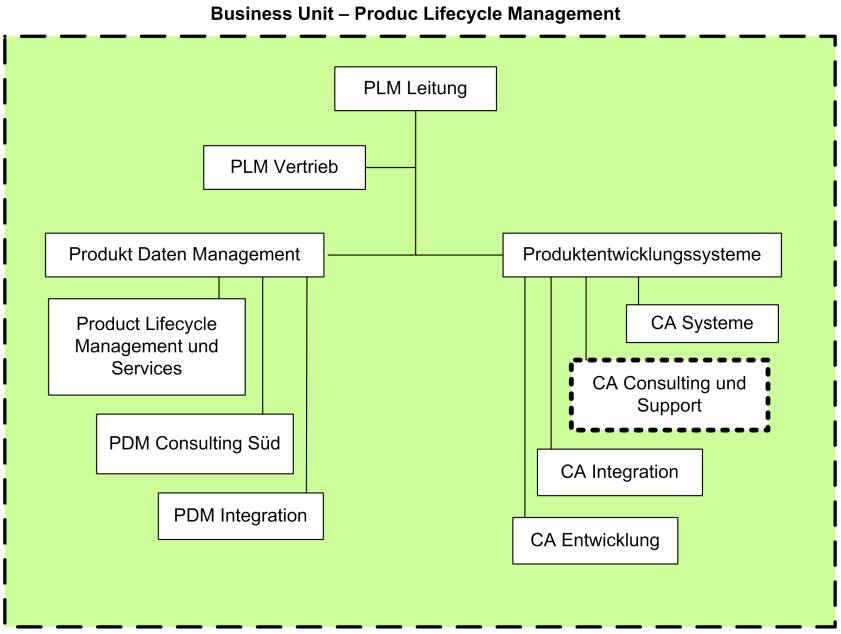

Abbildung 1.1: Struktur der Business Unit PLM

Die Business Unit PLM ist in verschiedene Fachgebiete und diese dann jeweils in Fachbereiche eingeteilt. Innerhalb des Fachbereiches „CA Consulting und Support“ – in der

Übersichtsgraphik 1.1 durch eine gestrichelte Linie gekennzeichnet – wird das Projekt Bordnetzentwicklung mit dem Volkswagen Bordnetz Entwicklungs System (VOBES) behandelt. Ein Schwerpunkt dieses Fachbereichs, der zugleich den Rahmen der vorliegenden Diplomarbeit vorgibt, ist die „[...] Berücksichtigung zukünftiger Technologien in der Bordnetzentwicklung und in der Datenverwaltung [...]“.[1]

1.2 Bordnetzentwicklung

Ein Teilbereich der Bordnetzentwicklung in der Automobilindustrie beschäftigt sich mit der Entwicklung und Anpassung von Leitungssträngen eines Fahrzeugs in der Planungsphase (virtuelles Fahrzeug). Die Vielzahl der elektrischen Systeme, die mittlerweile in einem Auto enthalten sind und miteinander kommunizieren, sind sehr komplex. „Bis zu 2000 Meter an Leitungen werden für die Verbindungen aller Systemkomponenten eines Mittelklasse-PKWs – vom Motor über ABS, Lichtmaschine und Klimaanlage bis hin zum Radio – benötigt.“[2] Die Leitungstränge eines virtuellen Fahrzeuges unterliegen während des Entwicklungsprozesses ständigen Veränderungen und Anpassungen. Vor diesem Hintergrund entwickelte die gedas in Zusammenarbeit mit der Volkswagen AG das so genannte Volkswagen Bordnetz Entwicklungssystem (VOBES).
VOBES ist der Überbegriff für das Zusammenspiel von CAD-Systemen (Computer Aided Design), Erweiterungen dieser CAD-Systeme und Richtlinien für die Automobilkonstruktion. Der gesamte Zyklus einer Bordnetzentwicklung wird damit unterstützt und geleitet. Einen Überblick über die verschiedenen Produktarten von VOBES, sowie deren Zusammenhänge und Abhängigkeiten, verschafft Abbildung 1.2[3] auf Seite 3. In den folgenden Abschnitten wird auf den Entwicklungsprozess eines Bordnetzes mit VOBES näher eingegangen. Zu Beginn der „VOBES Prozesskette“ werden mit dem Schaltplan[4]-Editor LCable[5] Systemschaltpläne erstellt. Ein Systemschaltplan, in der Abbildung 1.2 durch die Abkürzung „SYS“ gekennzeichnet, enthält die logischen Verbindungen elektrischer Komponenten des Fahrzeugtyps,

[1] Quelle: gedas Intranet, Business Unit PLM.
[2] [VIRA2002] Virag, Anton: *VW und gedas entwickeln Lösung für virtuelles Bordnetzdesign*, in: Automotive Engineering Partners, Februar 2001, Ausgabe 02.
[3] [GVEM2001] Grigor, A., Valentini, C., Eysoldt, K., Mikosch, O.: *Vortrag über die Bordnetzentwicklung*, Fachhochschule Braunschweig/Wolfenbüttel, 04.12.2001.
[4] Ein Schaltplan (auch Schaltbild) ist der Plan eines Gerätes, einer Anlage oder allgemein einer elektrischen Schaltung, auf dem die Bauteile durch genormte Symbole dargestellt sind.
[5] LCable (Logical Cable) ist eine umfassende Softwareumgebung des Unternehmens Mentor Graphics für die Entwicklung, Analyse und Dokumentation elektrischer Systeme mit dem Schwerpunkt Leitungsstrang.

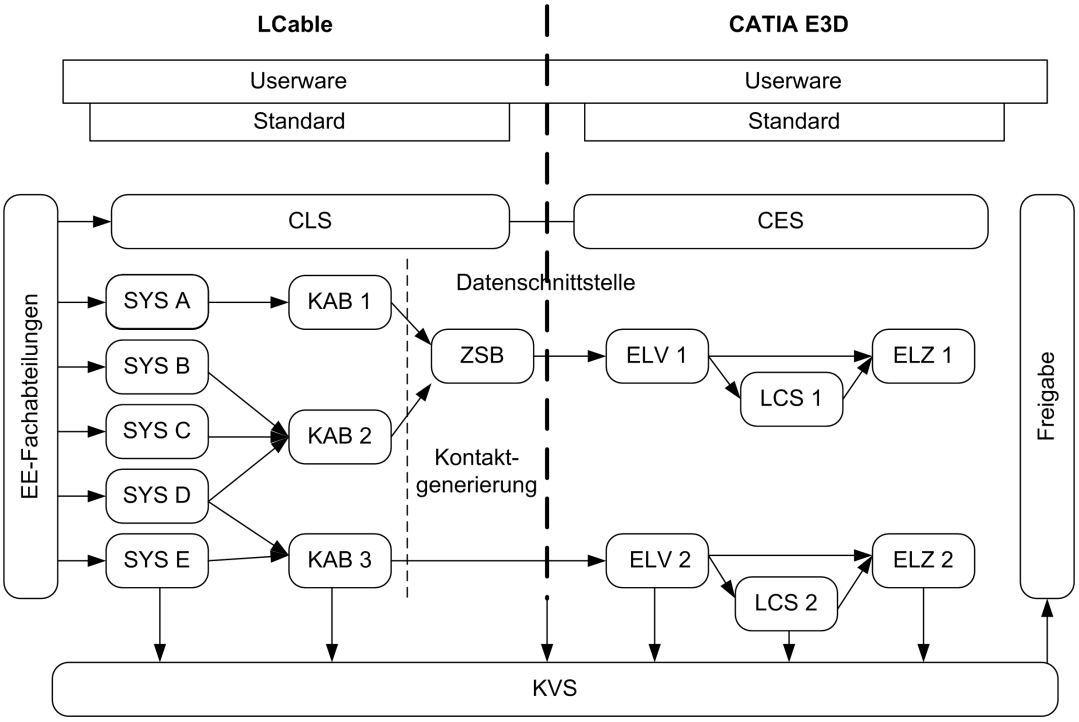

Abbildung 1.2: Volkswagen Bordnetz Entwicklungssystem (VOBES)

unabhänig von technischen Details. Dem Namen nach wird in einem Systemschaltplan ein gesamtes System abgebildet, wie z.B. die Zentralverriegelung. Der nächste Schritt ist das Anfertigen der Kabelschaltpläne (in der Abbildung 1.2 mit „KAB“ abgekürzt) in LCable. Dabei werden aus den logischen Verbindungen des Systemschaltplanes physikalische Leitungen für das Fahrzeug definiert. In der Darstellung 1.3 ist ein solcher KAB beispielhaft abgebildet. Nach der Fertigstellung der Kabelschaltpläne werden alle Daten, die für die Verlegung in CATIA[6] notwendig sind, mittels des Schnittstellenprogramms E3LCable erzeugt und zum Abgleich mit der Verlegung in CATIA bereitgestellt. Komplexere Leistungsstränge werden durch mehrere Kabelschaltpläne erfasst. Im VOBES-Prozess ist dies durch einen Zusammenbau möglich (in der Abbildung 1.2 durch „ZSB“ abgekürzt). Ebenso können die Ausstattungen der einzelnen Kabelschaltpläne in einer übergeordneten Ausstattung des Zusammenbaus festgelegt werden. Die Verlegung der Leitungsstränge des Fahrzeugs in einem dreidimensionalen Raum wird als nächstes durchgeführt.

[6]CATIA (Computer Aided Three-Dimensional Interactive Application) ist ein professionelles CAD-Programm, das ursprünglich für den Flugzeugbau entwickelt wurde und sich auch im Fahrzeugbau als Standard etabliert hat. Mit CATIA ist es möglich, zweidimensionale Zeichnungen zu erstellen und dreidimensionale Modelle zu entwickeln.

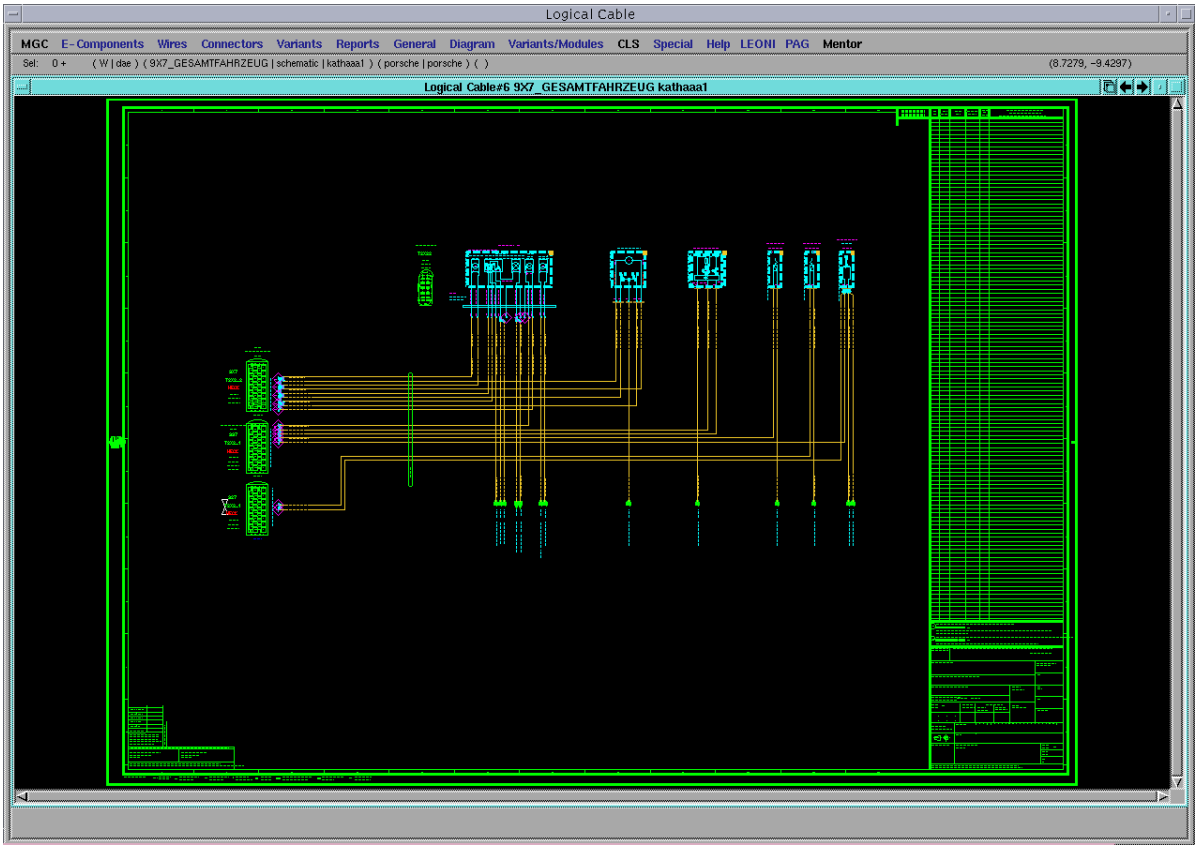

Abbildung 1.3: Beispiel für kabelorientierten Schaltplan

Im VOBES-Prozess ist dafür das CAD-System CATIA E3D zuständig, welches an die Bedürfnisse des Volkswagen Konzerns angepasst wurde. Anschließend erfolgt die Analyse oder Fehlerkorrektur der entstandenen virtuellen Fahrzeugprototypen. Beispielsweise werden Leitungen im virtuellen Abbild eines Fahrzeugtyps verlegt und somit die Durchmesser der Leitungsstränge berechnet, die später im Fahrzeug berücksichtigt werden müssen. Nur so kann gewährleistet werden, dass die Leitungsstränge im Serienfahrzeug Platz finden. Dadurch ergibt sich ein vollständiges Datenmodell, aus dem eine abschliessende dreidimensionale Zeichnung des Leitungsstranges generiert wird, wie beispielhaft in Abbildung 1.4 zu sehen ist. Nach weiteren Arbeitsschritten erhält man eine vollständige Dokumentation der Leitunsgsstränge, bestehend aus genauen Maßen und Berichten mit Listen aller verwendeten Bauteile. Die Grundlage dieses Entwicklungsprozesses stellen Bibliotheken mit Komponenteninformationen dar. Auf der LCable-Seite ist dies das Cable-Library-System (CLS), bei CATIA die CES-Bibliothek (Catia-Electrical-System-Bibliothek).

Diese werden an zentraler Stelle allen am Entwicklungsprozess beteiligten Mitarbeitern zur Verfügung gestellt. Sie sind aufeinander abgestimmt und gewährleisten somit die Konsistenz der Daten.[7]

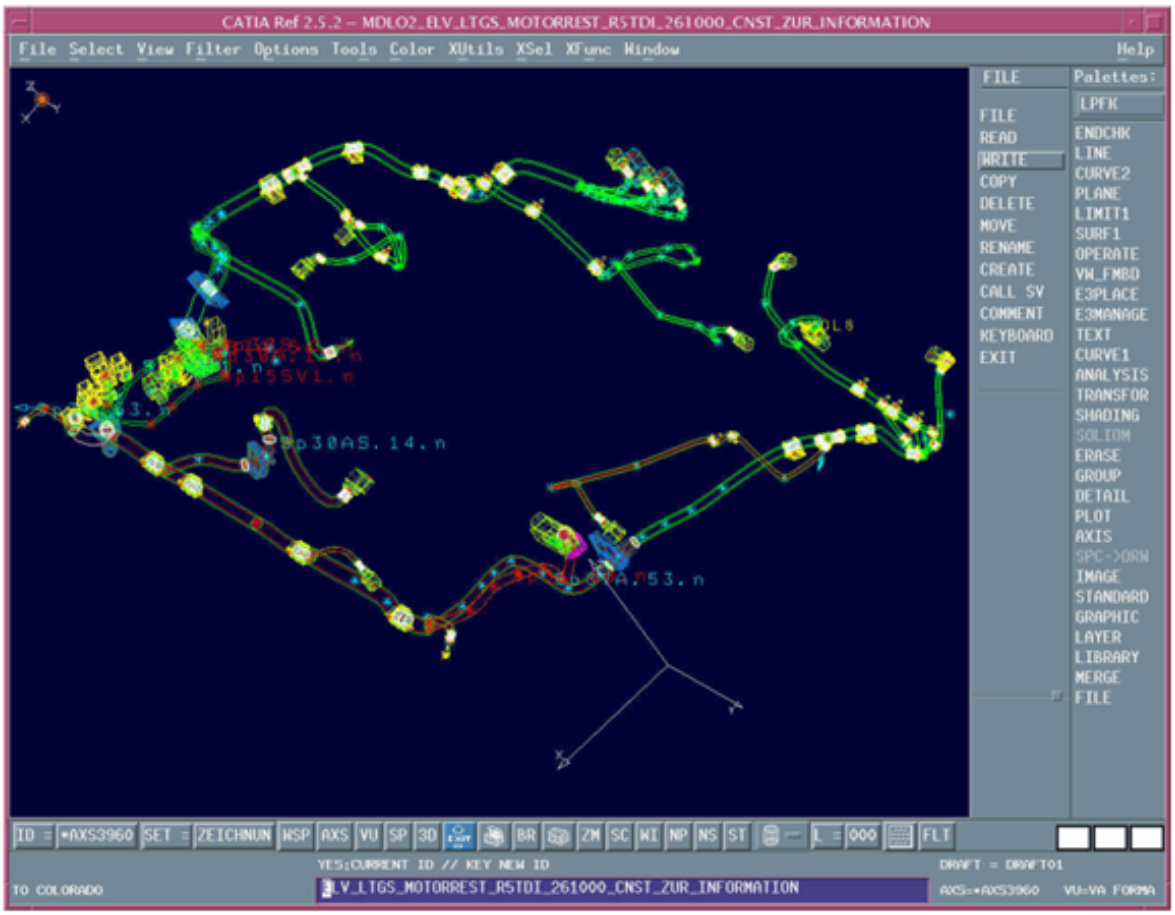

Abbildung 1.4: Beispiel für Einzelstrangverlegung

[7] vgl. [VIRA2002]

2 Aufgabenstellung und Zielsetzung

Aufgabenstellung

In der heutigen Bordnetzentwicklung treten eine Vielzahl von Problemen auf, die Zeitverzug in der Entwicklung und Fertigung zur Folge haben und zu Fehlern führen können, die sich erst am Ende der Entwicklung oder, schlimmer noch, erst beim Kunden auswirken. Dies wird durch einen Entwicklungsprozess hervorgerufen, der sich wie folgt charakterisieren lässt:

- verteilte Entwicklung der Bordnetze
- Einsatz verschiedenster Applikationen
- Heterogene Dokumentenstrukturen

Durch die oben beschriebene Problemstellung und die weiterhin anwachsende Komplexität in der KFZ-Bordnetzentwicklung, wird der Einsatz von neuen Beschreibungssprachen und Applikationen notwendig.
Der erste Schritt in diese Richtung wurde mit der Entwicklung einer Export-Schnittstelle für den Schaltplaneditor LCable getan. Durch diesen ist es möglich, Daten eines Bordnetzschaltplanes nach XML (Extensible Stylesheet Language) oder SVG (Scalable Vector Graphics) zu exportieren. Die exportierten XML-Daten beinhalten dabei einen logischen Teil eines Bordnetzes, die SVG-Daten hingegen graphische Teildaten. Der nächste Schritt war die Entwicklung zweier HTAs (Hyper Text Applikationen), dem BDX Project Report und dem BDX Project Viewer, dargestellt in Abbildung 2.1. BDX steht hierbei für die Bordnetz Dokumentation basierend auf XML. Basis dieser Applikationen sind die aus LCable exportierten Daten, wobei der BDX Project Report die XML-Daten nutzt und der BDX Project Viewer auf den SVG-Daten aufsetzt. Diese Teildaten enthalten jeweils nur einen Bruchteil der zur Bordnetzspezifikation notwendigen Daten. Der BDX Project Report wurde dazu entwickelt, mit Hilfe von Reportformatvorlagen Berichte über die Inhalte eines Schaltplans erzeugen zu können. Der BDX Project Viewer hingegen wird dazu eingesetzt, applikationsunabhängig Schaltpläne in SVG anzuzeigen.

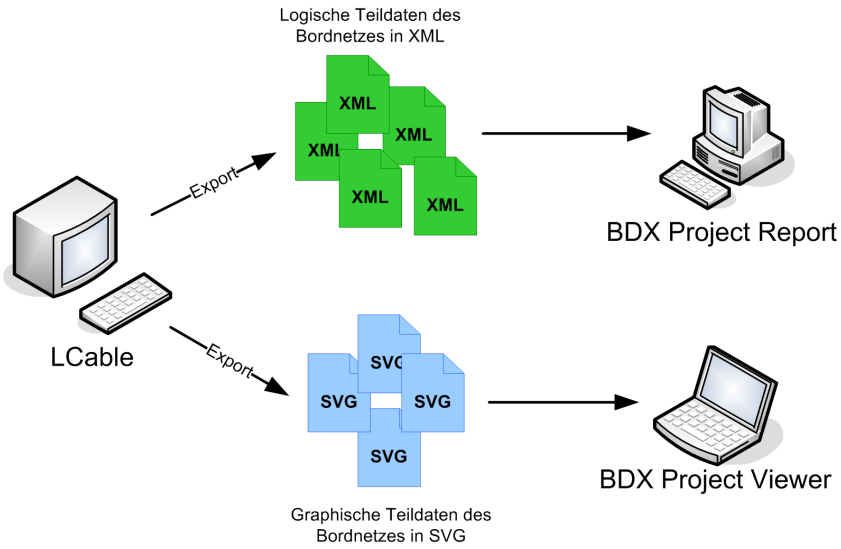

Abbildung 2.1: Problemstellung der Diplomarbeit

Beide Applikationen werden momentan produktiv in der Entwicklung eingesetzt. Ein Nachteil dieses Ansatzes ist jedoch, dass nicht gewährleistet werden kann, dass die Datengrundlage der exportierten Daten auf derselben Schaltplanversion beruht. Beispielsweise ist es möglich, dass nach Änderungen am Schaltplan zwar der XML-Export, nicht aber der SVG-Export angestossen wurde. Dadurch werden unterschiedliche Varianten von Schaltplandaten in den Applikationen verwendet, wodurch Inkonsistenzen der Inhalte auftreten können. Ein weiterer negativer Aspekt offenbart sich bei nachträglichen Veränderungen der exportierten XML-Daten. Dazu ist es nötig, die Export-Schnittstelle in LCable entsprechend anzupassen und den Export-Vorgang neu zu starten, was insgesamt ein zeitintensiver Vorgang ist. Des Weiteren ist es nicht möglich, aus der einen Applikation auf die Daten der anderen zuzugreifen. Dies bedeutet, dass es im BDX Project Report keine Möglichkeit gibt, auf graphische Informationen von Schaltplankomponenten zuzugreifen, da diese in den exportierten XML-Daten nicht enthalten sind. An diesem Punkt setzt das Thema der vorliegenden Diplomarbeit an. Über eine Export-Schnittstelle werden die gesamten Bordnetzdaten anhand eines applikationsneutralen Schnittstellenmodelles in XML abgebildet und exportiert. Die Bordnetzdaten sollen mittels der Extensible Stylesheet Language Transformation (XSLT) in verschiedene Ausgabeformate, wie z.B. das standardisierte Graphikformat SVG, HTML, XML oder PDF transformiert werden. In Abbildung 2.2 ist dies graphisch dargestellt. Vorteilhaft an diesem Ansatz ist die gemeinsame Datenbasis. Diese enthält applikationsunabhängig die gesamten Daten, sowie die Struktur eines Bordnetzes.

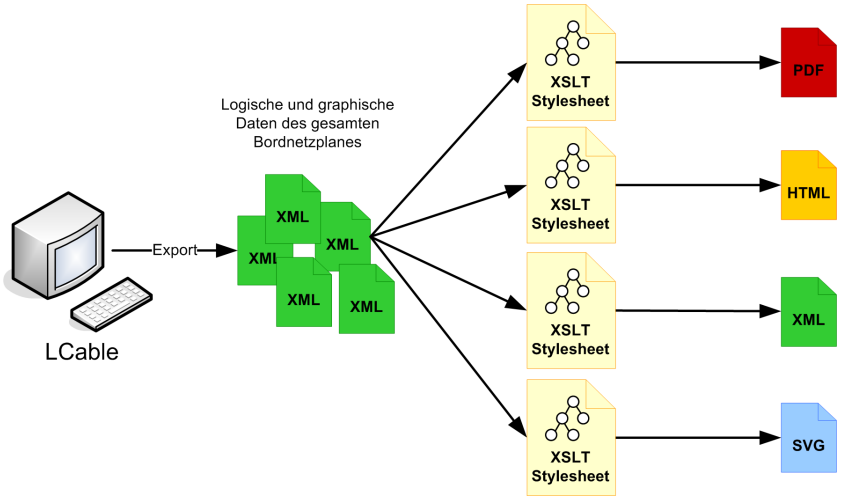

Abbildung 2.2: Ansatz der Aufgabenstellung der Diplomarbeit

Je nach Verwendungszweck können Teilinformationen des Bordnetzes extrahiert und in einem gewünschten Ausgabeformat dargestellt werden. Inkonsistenzen bezüglich der Bordnetzversionen können aufgrund der gemeinsamen Datenbasis nicht mehr auftreten. Darstellungsänderungen bedürfen nicht länger der Anpassung der Export-Schnittstelle in LCable, sondern lediglich des jeweiligen XSLT-Stylesheets.

Zielsetzung

Für die Verbesserung und Unterstützung des Bordnetzentwicklungsprozesses wurde im vorherigen Abschnitt der Ansatz besprochen, Bordnetzentwicklungsdaten in einem standardisierten Format applikationsneutral zu halten und verschiedene Sichten auf diese Informationen zu erzeugen. Hierbei sind folgende Fragen zu beantworten.

- Ist es möglich, Bordnetzdaten in XML applikationsneutral abzubilden und zu halten?
- Sind verschiedene Sichten auf das Bordnetz ableitbar?
- Sind XML und XSLT für die Speicherung und Aufbereitung der Bordnetzinformationen geeignet?
- Ist XML als alleiniges natives Datenformat in der Bordnetzentwicklung geeignet?

Zur Beantwortung dieser Fragen soll ein Prototyp realisiert werden. Mit Hilfe dieser prototypischen, entwicklungsbegleitenden Plattform soll die Generierung verschiedener Sichten auf

Bordnetzdaten ermöglicht werden. Dabei wird der Fokus auf der dynamischen Generierung von Schaltplänen in SVG liegen. Beispielhaft soll die Durchführung einer dynamischen Analyse und eines Reports erfolgen. Darunter sind ebenfalls Sichtgenerierungen zu verstehen, wobei Bordnetzinformationen für Auswertungszwecke extrahiert und in übersichtlicher Form dargestellt werden sollen.

3 Grundlagen und Voraussetzungen

Dieses Kapitel befasst sich mit der Erläuterung der Begriffe „W3C“, „XML“, „XSL“ und „SVG“, sowie mit der Betrachtung des Einsatzes eines Protoypen in einem Softwareentwicklungsprozess. Zu den einzelnen Technologien wird ein gewisses Grundwissen vorausgesetzt. Dadurch sind an dieser Stelle sind keine vollständigen Referenzen oder Ausarbeitungen zu finden, sondern lediglich grobe Überblicke, welche vorhandenes Wissen auffrischen und zum Verständnis der Arbeit beitragen sollen.

3.1 World Wide Web Consortium (W3C)

Das World Wide Web Consortium (W3C) ist ein internationales Industrie-Konsortium, das gemeinsam vom MIT Laboratory for Computer Science (LCS) in den USA, dem Institut National de Recherche en Informatique et Automatique (INRIA) in Frankreich und der Keio-Universität in Japan geführt wird. Seit 1994 entwickelt es einheitliche, das Internet betreffende, Technologien(Spezifikationen, Richtlinien, Software und Software Tools).[1] Gegenwärtig sind mehr als 330 Organisationen Mitglieder des Konsortiums, wobei diese Zahl einem stetigen Wachstum unterliegt. Innerhalb der letzten Jahre gingen Standards wie HTML (Hypertext Markup Language), XML, XSL und DOM (Document Object Model) aus der Arbeit des W3C hervor, welche seitdem kontinuierlich weiterentwickelt wurden. Der Einsatz von Technologien des W3C ist somit durchaus sinnvoll, da dieses Konsortium ein Garant für Standardisierungen ist, welche von einem grossen Anwenderkreis akzeptiert und verwendet werden. Weitere Informationen über das World Wide Web Consortium sind unter `www.w3.org` zu finden.

[1] vgl. [FRFD2004] Fraunhofer Institut für integrierte Publikations- und Informationssysteme und FH Darmstadt: *Das W3C*, FH Darmstadt, Sommersemester 2004, URL: `http://www.ipsi.fraunhofer.de/~putz/FHD/XML/PDFs_2004/W3CundSprachen.pdf`, Datum des Zugriffs: 02.03.2004.

3.2 XML

An dieser Stelle erfolgt ein Überblick über die Möglichkeiten von XML. Ausführliche Informationen zu den angesprochenen Themen XSLT und SVG sind in den entsprechenden Referenzkapiteln zu finden. Auf HTML, ASCII (American Standard Code of Information Interchange), WML (Wireless Markup Language), XLink (XML Linking Language) und XPointer (XML Pointer Language) wird nicht näher eingegangen - für genaue Informationen wird an dieser Stelle auf das Literaturverzeichnis verwiesen.

3.2.1 Definition XML

Die Extensible Markup Language (XML) wurde einerseits für die Definition von Auszeichnungssprachen[2] bzw. Datenaustauschformaten und andererseits für die Beschreibung von Daten entwickelt. In XML ist es möglich, Strukturen abzubilden und eigene Auszeichnungssprachen für verschiedene Dokumenttypen zu erstellen.
XML ist in dem Sinne erweiterbar, dass es sich nicht wie bei HTML um ein vordefiniertes Format handelt. Durch eine Document Type Definition (DTD) oder ein XML Schema kann die Syntax, Struktur und Bedeutung der XML-Tags[3] definiert werden. Die graphische Repräsentation der Daten erfolgt z.B. mit der Extensible Style Sheet Language (XSL) oder Cascading Style Sheets (CSS). Inhalte von XML-Dateien werden beispielsweise mit der Extensible Stylesheet Language Transformation (XSLT) nach HTML, PDF (Portable Document Format), SVG, WML usw. transformiert.
XML-Anwendungen[4] sind unter anderem für die Darstellung in Web-Browsern geeignet – also als Ersatz oder Ergänzung von HTML. Weiterhin sind XML-Anwendungen auch als Datenquelle für Applikationen einsetzbar, die auf logischer Informationsverarbeitung basieren, etwa Tabellenkalkulationen oder Datenbanken.

[2] Eine Auszeichnungssprache (auch Markup Languange (ML)) dient zur Beschreibung von Informationen oder des Verfahrens oder der Schritte, die zur Darstellung nötig sind.

[3] Tag: Ein von spitzen Klammern umschlossener Elementname, der zur Auszeichnung eines Dokuments verwendet wird. In XML kann der Elementname frei gewählt werden.

[4] „Unter einer XML-Anwendung oder einer XML-Applikation versteht man die Festlegung von XML-Befehlen für eine Klasse von XML-Dokumenten gleicher Struktur[...]“. Beispiele dafür sind SVG, WML und HTML. Vgl. [PART2000] Partl, Hubert: *XML - Extensible Markup Language*, September 2000, URL: `http://www.boku.ac.at/htmleinf/xmlkurz.html?applikationen`, Datum des Zugriffs: 20.03.2004.

3.2.2 Entwicklung XML

Die Entwicklung von XML begann 1996 und war im Wesentlichen von den Erfahrungen mit SGML und HTML geprägt:

- Prinzipiell hatte sich SGML bewährt, konnte sich aber aufgrund der hohen Komplexität und dem Umfang der Sprachkonstrukte über professionelle Kreise hinaus kaum verbreiten.
- HTML erfreut sich zwar einer starken Verbreitung und einer großen Akzeptanz, besitzt aber folgende Nachteile: Es ist nicht erweiterbar, enthält nur eine minimale Realisierung des Hypertext-Konzepts (nur unidirektionale 1-zu-1-Links) und vermischt Struktur und Layout von Dokumenten.

Nach [MSH2002][5] wurde XML aus diesen Erfahrungen heraus mit dem Ziel geschaffen, die besten Eigenschaften dieser „beiden Welten“ zu übernehmen. Dazu zählt einerseits die Reduktion der Komplexität von SGML durch das Weglassen selten benötigter Konstrukte und andererseits die Nutzung der Akzeptanz und Verbreitung von HTML. Weiterhin wurde die Entwicklung von XML von der Strukturiertheit und Erweiterbarkeit von SGML beeinflusst, sowie der 20-jährigen Erfahrung im Umgang mit diesem. Im Februar 1998 wurde die Entwicklung abgeschlossen und der XML-Standard Version 1.0[6] vom W3C verabschiedet.

3.2.3 XML in der Bordnetzentwicklung

Es stellt sich die Frage, weshalb XML verwendet werden soll, um Bordnetzdaten abzubilden. Folgende Auflistung ist eine Übersicht der relevanten Merkmale von XML und des Nutzens, der sich jeweils für die Bordnetzentwicklung ergibt.

Offenes Format XML wurde vom W3C entwickelt und ist ein offener Standard. Im Gegensatz zu proprietären Formaten bestimmter Applikatonshersteller wie z.B. LCable oder CATIA sind XML-Daten unabhängig von einem einzelnen Softwareprodukt oder -anbieter. Der große Vorteil ist, dass unabhängig vom Datenformat einer Applikation Informationen bzw. Daten gespeichert werden können. Es besteht somit keine

[5] vgl. [MSH2002] Middendorf, Stefan; Singer, Reiner; Heid, Jörn: *Programmierhandbuch und Referenz für die Java-2-Plattform, Standard Edition*, dpunkt.Verlag, Heidelberg, 2002, URL: `http://www.dpunkt.de/java/Programmieren_mit_Java/XML/11.html`, Datum des Zugriffs: 02.04.2004.

[6] Nähere Informationen zum Standard XML 1.0: `http://www.w3.org/TR/2004/REC-xml-20040204/`.

Abhängigkeit von proprietären Formaten, wenn die Applikation XML unterstützt. Die Struktur der dargestellten Informationen ist frei bestimmbar.

Medienneutralität Sofern einmal alle Informationen und Daten in XML vorliegen, können diese je nach Bedarf in das gewünschte Ausgabeformat transformiert werden (z.B. HTML, PDF, SVG, WML u.v.a.). XML-Dokumente enthalten keine Formatierungsangaben und sind somit unabhängig vom Ausgabeformat (Papier, Monitor usw.). Dadurch wird eine hohe Flexibilität für die Darstellung der Informationen aus den XML Daten gewährleistet.

Lizenzfreiheit Da XML als W3C-Entwicklung lizenzfrei ist, fallen keine Lizenzgebühren für Applikationen an. Geeignete, auf spezielle Bedürfnisse abgestimmte Applikationen können gesucht oder wahlweise eigens entwickelt werden.

Applikationsneutralität Durch die Applikationsneutralität von XML sind keine speziellen Programme (und somit Lizenzen) notwendig, um XML Daten zu erstellen, visualisieren oder verwenden zu können. Vorteilhaft ist ebenfalls, dass ein Anwender auch dann noch Zugriff auf die Daten hat, wenn die Software, mit der die Dokumente erstellt wurden, nicht mehr in Gebrauch ist.

Gewährleistung der Datenintegrität Die logische Korrektheit und Gültigkeit (Validierbarkeit) von Informationen, die in XML gehalten werden, kann automatisiert geprüft werden. Somit kann sichergestellt werden, dass die betreffenden XML-Daten der gewünschten Struktur entsprechen und keine Unstimmigkeiten aufweisen.

Erweiterbarkeit Es ist jedem jederzeit möglich, die momentan verwendeten Auszeichnungssprachen (z.B. XML, XSLT, XPath ...) durch Hinzufügen von neuen Auszeichnungssprachen (XLink, XPointer, SVG) zu erweitern. Dadurch wird eine Anpassung an neue Bedürfnisse möglich.

Datenaustausch XML-Applikationen eignen sich als Plattform- und Software-unabhängiges Austauschformat für Daten zwischen verschiedenen Applikationen, die andernfalls nicht miteinander kommunizieren könnten.

Lesbarkeit Da sie als Textdateien (im ASCII-Format) vorliegen, sind XML-Dokumente in denen Daten strukturiert und gespeichert werden, nicht nur für Maschinen, sondern auch den Menschen lesbar. Das erlaubt die unkomplizierte Handhabung, Korrektur

und Bearbeitung der Daten. Die in XML zur Abgrenzung der Daten verwendeten Tags können, wenn sie sinnvoll gewählt sind, gleichzeitig als Meta-Informationen über die in ihnen enthaltenen Daten genutzt werden. Die ständige Weiterentwicklung von XML und XML-Applikationen gewährleistet Planungssicherheit und Stabilität.

3.2.4 XML Syntax

Im Zusammenhang mit den nun folgenden Erläuterungen zur XML-Syntax gibt es zwei grundlegende Konzepte. Das erste besagt, dass jedes XML-Dokument wohlgeformt sein muss. Ein wohlgeformtes Dokument ist ein Dokument, bei dem jeder geöffnete Tag auch wieder geschlossen wird, bei dem die Verschachtelungsreihenfolge nicht durchbrochen wird und das hinsichtlich der Spezifikation syntaktisch korrekt ist. Um eine XML-Instanz auf ihre Wohlgeformtheit zu prüfen, werden XML-Parser[7] verwendet, wobei es zwei Ansätze gibt, diese zu implementieren.[8] Ein Ansatz ist der ereignisgesteuerte Parser, bei dem die Simple API for XML (SAX) eine standartisierte Schnittstelle darstellt. Ein weiterer Ansatz ist der Baum-orientierte Parser, bei dem das Document Object Model (DOM) als Plattform- und Sprachen-neutrale Schnittstelle zur Manipulation des Dokumentbaumes zur Verfügung steht. Das zweite grundlegende Konzept in Hinsicht auf XML-Dokumente besagt, dass diese gültig (valid) sein können, es aber nicht sein müssen. Ein gültiges Dokument ist ein Dokument, das einer Document Type Definition (DTD) gehorcht. Die DTD definiert die Grammatik und Tag-Menge für eine bestimmte XML-Anwendung.

Abbildung 3.1: Handskizze eines einfachen Schaltplanes

[7] Ein XML-Parser wird für die Verarbeitung von XML-Dokumenten eingesetzt. Er zerlegt den Quelltext, baut eine interne Datenstruktur auf und prüft diese nach festen Regeln. Die Daten können an andere Programme weitergegeben oder über die SAX- oder DOM-Schnittstelle verwendet werden.

[8] vgl. [ABM2000a] Anderson R., Birbeck M., Kay M. u.a.: *XML Professional, Kapitel 6: SAX 1.0: Die Simple API[9] für XML*, 1.Auflage, MITP Verlag GmbH Bonn, 2000, Seite 197ff.

Wenn ein Dokument ein DTD/XML Schema spezifiziert und den darin festgelegten Regeln folgt, so ist dieses Dokument ein gültiges XML-Dokument. XML-Dokumente können alternativ durch ein Schema spezifiziert werden.[10] Ein kurzes Beispiel wird diese Zusammenhänge näher erläutern. Die Skizze in Abbildung 3.1 zeigt einen Schaltplan, der in XML abgebildet werden soll.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT schaltplan (komponenten)>
<!ELEMENT komponenten (komponente)>
<!ELEMENT komponente (pin+)>
<!ATTLIST komponente
        name ID #REQUIRED
>
<!ELEMENT pin EMPTY>
<!ATTLIST pin
        id ID #REQUIRED
>
```

Abbildung 3.2: Beispiel-DTD eines Schaltplanes - schaltplan.dtd

Die DTD aus Abbildung 3.2 beschreibt die Basis eines solchen Schaltplanes. Auf Grundlage dieser DTD können somit Schaltpläne erstellt werden, die aus mehreren Komponenten mit 1-n Pins[11] bestehen können. Eine mögliche Instanz dieses Schaltplanes auf Grundlage der DTD aus Abbildung 3.2 ist in Abbildung 3.3 zu erkennen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schaltplan SYSTEM "D:schaltplan.dtd">
<schaltplan>
 <komponenten>
  <komponente name="lampe1">
   <pin id="P6"/>
   <pin id="P5"/>
  </komponente>
 </komponenten>
</schaltplan>
```

Abbildung 3.3: Beispiel-XML-Instanz für einen Schaltplan

In der ersten Zeile steht die so genannte XML-Deklaration, in der die verwendete XML-Version angegeben wird. Die darauffolgende Zeile verweist auf die externe DTD „schalt-

[10] vgl. [MCLA2001] McLaughlin, Brett: *Einführung in XML*, O'Reilly Verlag 2001, URL: `http://www.oreilly.de/artikel/xml_einf.html`, Datum des Zugriffs: 02.02.2004.

[11] Ein Pin ist ein Anschlusstift in einem Schaltplan, wo die Möglichkeit besteht, eine Leitung anzuschliessen.

plan.dtd“. Dies besagt, dass dieses XML-Dokument der Struktur, die in der DTD angegeben ist, genügen muss. Des Weiteren muss es in jeder XML-Datei ein Element[12] geben, dem alle anderen Elemente untergeordnet sind - in diesem Fall <schaltplan>. Unterhalb dieses Elements folgt das Element <komponenten>, das wiederum ein Kindelement[13] <komponente> besitzt, usw. Diese XML-Instanz beschreibt einen Schaltplan, der aus einer Komponente mit dem Namen „lampe1“ besteht, die wiederum zwei Pins mit den ID's „P5“ und „P6“ besitzt . Diese beispielhafte Abbildung einer Schaltplanstruktur in XML zeigt ansatzweise, wie mittels XML Informationen strukturiert werden können.

3.3 SVG

3.3.1 Definition SVG

Scalable Vector Graphics (SVG) ist der offizielle Vektorgraphikstandard des W3C. Aufbauend auf XML arbeitet SVG mit den anderen XML-Basistechnologien wie DTD, Schema, XSL, Namespaces, etc. zusammen. Mithilfe von SVG können zweidimensionale Vektorgraphiken mitsamt Animation und Interaktion, z.B. für das Internet, beschrieben werden. In SVG erstellte Graphiken sind skalierbar und können in jeder möglichen Auflösung gedruckt werden.

3.3.2 Entwicklung SVG

Eine offizielle Empfehlung des W3C wurde SVG 1.0 am 04. September 2001. Zuvor wurden unterschiedliche Konzepte erarbeitet, wobei von diesen die meisten Vorteile zusammengefasst, ergänzt und optimiert wurden. Die ausführliche Spezifikation zu SVG 1.1, der momentan aktuellen Version, ist unter www.w3.org/TR/SVG zu finden. Das Konsortium, das am SVG-Standard arbeitet, besteht aus den Mitgliedern des W3C, Forschungsinstituten und Firmen aus dem Graphik-, Multimedia- und Netzwerkumfeld. Zu diesen gehören unter anderem Adobe, Apple, Autodesk, Canon, Corel, HP, IBM, Kodak, Macromedia, Netscape/AOL, Quark,

[12] Ein Element ist die grundlegende Informationseinheit in einem XML-Dokument. Sie bestehen aus einem, den Typ des Elements benennenden, Start-Tag, dem Elementinhalt und dem End-Tag. Das Start-Tag kann dabei auch eine unbestimmte Zahl von eindeutigen Attributen enthalten. Der Elementinhalt kann direkt aus Zeichendaten bestehen oder wiederum aus untergeordneten Elementen oder aus einer Mischung aus Zeichendaten und Elementen.

[13] Ein Kind-Element ist ein Unterelement eines anderen Elements. Es steht also innerhalb des öffnenden und schließenden Tags des Eltern-Elements.

Sun und Xerox. Das Interesse an SVG ist somit deutlich zu erkennen. Einige Softwarehersteller implementieren SVG bereits in ihren Produkten, wie z.B. der Browser Mozilla 1.7 Alpha[14]. Dadurch entfallen zusätzliche Installationen für Plugins zum Anzeigen der SVG-Graphiken.

3.3.3 SVG in der Bordnetzentwicklung

Da SVG ein offener Standard ist, der Vektorgraphik, Rastergraphik, Animation und Interaktivität unterstützt, sind die Einsatzmöglichkeiten sehr breit gefächert und somit ideal für die Bordnetzentwicklung. So können beispielsweise technische Zeichnungen dargestellt werden. Theoretisch ist es möglich, durch einfache geometrische Grundformen wie Linien, Rechtecke, Kreise und Pfade Bordnetz-Schaltpläne in SVG darzustellen. In einfachen Texteditoren sind diese Graphiken editierbar, wodurch wiederum Lizenzkosten gespart werden und keine Abhängigkeit von speziellen Applikationen oder Plattformen besteht.
Weiterhin unterstützt SVG Interaktivität und Animation. Mit deren Hilfe können Benutzer „geführt“ und Inhalte verfolgt werden, wie z.B. anhand der Sichtbarkeit einer Linie in einer komplexen Gesamtgraphik. Beim Überfahren kann sich die Liniendicke verändern, wodurch die Linie aus dem übrigen Schaltplan herausragt. Ein Nutzen der Tatsache, dass SVG-Graphiken reine Text-Dateien sind, ist, dass Mensch und Maschine gleichermaßen die Dateien „lesen und interpretieren“ können. Es wird also nie passieren, dass SVG Graphiken nach vielen Jahren nicht mehr „lesbar“ sind, weil die Anzeige-Programme (Interpreter) dafür nicht weiterentwickelt wurden. Ebenfalls vorteilhaft für die Bordnetzentwicklung ist die Durchsuchbarkeit von SVG-Graphiken. Wurde ein Schaltplan in SVG abgebildet, ist dieser nach Merkmalen abfragbar, z.B. nach dem Wert einer ID. Bei Rastergraphiken ist das undenkbar.
Ähnlich wie bei XML, können während der Bordnetzentwicklung alle Beteiligten SVG als einheitliches Austausch- und Schnittstellenformat verwenden. SVG ist ebenfalls vom W3C standardisiert und somit lizenzfrei, wodurch es eine geeignete Basis darstellt.
SVG ist generierbar. Für die Bordnetzentwicklung bringt das den Vorteil, dass Bordnetz-Entwicklungssysteme als einzige Voraussetzung den Export von ASCII-Dateien unterstützen müssen. Zudem sind alle notwendigen Informationen vollständig vorhanden und ein Konverter ist implementierbar.

[14] [MOZI2004] Mozilla Organisation: *What's New in Mozilla 1.7 Alpha*, URL: `http://www.mozilla.org/releases/mozilla1.7a/README.html`, Datum des Zugriffs: 16.01.2004.

3.3.4 SVG-Struktur und graphische Möglichkeiten

Da die SVG Spezifikation sehr umfangreich ist, erscheint hier nur ein Auszug der wesentlichen Aspekte. Anhand eines Beispiels werden die grundlegenden Prinzipien der Erstellung von SVG-Graphiken vermittelt. Der zuvor in XML abgebildete Schaltplan mit der Lampe und den zwei Pins, inklusive der Pinbeschreibung (siehe beispielhafte Skizze in Abbildung 3.1 auf Seite 14), wird in SVG beschrieben. Das SVG-Grundgerüst ist in Abbildung 3.4 dargestellt.

Aufbau einer SVG-Datei

```
1 <?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
3  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
4  <svg xmlns="http://www.w3.org/2000/svg" width="800" height="600">
5   <!--...ich bin ein Kommentar...-->
6  </svg>
```

Abbildung 3.4: Grundgerüst einer SVG-Datei

Der XML-Prolog in der ersten Zeile
`<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>` enthält die Attribute `encoding` und `standalone`. Mit `encoding` wird eine Codetabelle angegeben , auf die das Dokument zurückgreift. In unserem Fall wählen wir die Tabelle ISO-8859-1-Standard, so dass auch Sonderzeichen und Umlaute im Quelltext verwendet werden können. Mit `standalone="no"` wird die Abhängigkeit von einem anderen Dokument ausgedrückt, in diesem Fall der DTD. Im `DOCTYPE` in der zweiten Zeile wird auf die SVG-DTD verwiesen, die öffentlich (`PUBLIC`) abgelegt ist[15]. Wie jedes XML-Dokument muss auch ein SVG-Dokument ein „Root-Element“ besitzen. In einer SVG-Datei ist dies das <svg>-Tag, wie beispielhaft in den Zeilen 4 und 6 zu sehen ist. Der Bezeichner für den eindeutigen Namensraum von SVG ist `http://www.w3.org/2000/svg`. Über die Attribute `width` und `height` werden die Außenmaße in Pixeln für das SVG-Fenster festgelegt.

Wo wird die SVG-Graphik gezeichnet?

Der Ursprung des Koordinaten-Systems, in dem eine SVG-Graphik abgebildet wird, liegt in der linken oberen Ecke. Der Grund dafür ist, das dies der einzige konstante Bezugspunkt

[15] `http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd`

einer Online-Darstellung ist. (Daher muss bei der Übertragung von Graphiken aus dem kartesischen Koordinatensystem den Y-Werten relativer Pfadbeschreibungen ein anderes Vorzeichen gegeben werden. Für absolute Pfadbeschreibungen müssen die Y-Werte entsprechend umgerechnet, oder die gesamte Graphik einheitlich transformiert werden.)

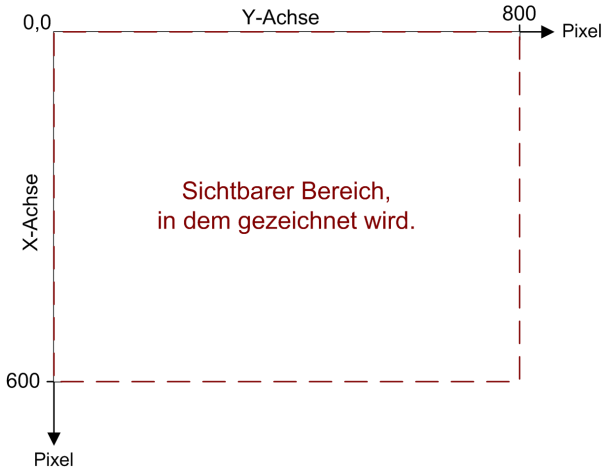

Abbildung 3.5: Zeichnungsrahmen der Beispiel-SVG-Graphik

In Abbildung 3.5 ist der sichtbare Bereich der Beispielgraphik zu sehen, ausgehend vom Koordinatenursprung (0,0) der linken oberen Ecke. Dieser wird 800 Pixel in der Breite und 600 Pixel in der Länge betragen. In Zeile 5 des SVG-Grundgerüsts ist ein Kommentar dargestellt, wie es bisher bei XML-Auszeichnungssprachen üblich ist
(`<!-- ..ich bin ein Kommentar.. -->`). Dieser soll den Bereich kennzeichnen, in dem die Kindelemente des `svg`-Tags und deren Attribute eingefügt werden, die für die eigentliche Darstellung einer SVG-Graphik notwendig sind.

Welche Graphikelemente gibt es?

Um nun also die Lampe, die zwei Pins und deren Pinbeschreibung des Schaltplanes in SVG darstellen zu können, wird das SVG-Grundgerüst erweitert. Die nachfolgende Auflistung des SVG-Quellcodes soll dies verdeutlichen. In den Zeilen 4-15 wurden nun SVG-Tags eingefügt, die den zuvor erläuterten Schaltplan beschreiben. Um die einzelnen Teile des Schaltplanes zu gruppieren, wird das SVG-Element <g> verwendet. Dadurch entsteht eine Einheit aller weiteren SVG-Elemente, die innerhalb dieses <g>-Elementes eingefügt werden. Eine Verschachtelung ist ebenfalls möglich, wie in den Zeilen 5-10 zu sehen ist. Dadurch werden in diesem Beispiel die SVG-Elemente `<line>` und `<circle>` zusammengefasst, die die Lampe

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
3  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
4  <svg xmlns="http://www.w3.org/2000/svg" width="800" height="600">
5   <title>SVG Transformationstest</title>
6    <g>
7     <g>
8      <line x1="450" y1="130" x2="450" y2="210" style="stroke:black; stroke-width:2px;"/>
9         <circle cx="450" cy="170" r="30" fill="yellow" stroke="black"/>
10        <line x1="430" y1="147" x2="473" y2="190" style="stroke:black; stroke-width:1px;"/>
11        <line x1="427" y1="189" x2="470" y2="147" style="stroke:black; stroke-width:1px;"/>
12    </g>
13    <circle cx="450" cy="210" r="5" fill="black" stroke="black"/>
14    <circle cx="450" cy="130" r="5" fill="black" stroke="black"/>
15    <text x="463" y="213" style="font-size:10px;fill:black; text-anchor:bottom">P6</text>
16    <text x="463" y="133" style="font-size:10px;fill:black; text-anchor:bottom">P5</text>
17   </g>
18 </svg>
```

Abbildung 3.6: SVG Quellcode eines Beispielschaltplanes

darstellen.

Das <line>-Element besitzt die Attribute x1, y1 und x2, y2 womit die Koordinaten (Startpunkt und Endpunkt) einer Linie beschrieben werden. Optional kann mithilfe des style-Attributes die Darstellung beeinflusst werden - in diesem Fall sind die Linien schwarz und 2 Pixel dick. Kreise werden durch das <circle>-Element beschrieben. Die Attribute cx und cy bezeichnen die Koordinaten des Kreismittelpunktes, r den Radius, fill die Füllfarbe und stroke die Linienfarbe. Die Lampe des Schaltplanes wird somit durch einen gelben Kreis symbolisiert, welcher von einem schwarzen Kreis mit einem Radius von 30 Pixeln umrandet ist und von zwei schwarzen, 2 Pixel dicken Linien gekreuzt wird. Zur Darstellung von Text im Browser dient in SVG das <text>-Element. Dadurch werden die textuellen Beschreibungen der Pins, wie in den Zeilen 13 und 14 zu sehen ist, dargestellt. Die Attribute x und y beschreiben den linken unteren Startpunkt der Grundlinie, auf der der Text „geschrieben“ wird.

Wie kann eine SVG-Graphik angezeigt werden?

SVG ist zwar als plattform- und browserunabhängiger Standard konzipiert, lässt sich im Internet aber momentan nur unter Verwendung von Plugins darstellen. Die native Unterstützung in Browsern ist momentan leider noch nicht vorhanden, es wird aber daran gearbeitet (wie zuvor erwähnt bei Mozilla 1.7). Der Adobe SVG Viewer 3.0, ein Plugin für den Internet Explorer, unterstützt momentan die für die Bordnetzentwicklung relevante Funktionalität des SVG-Standards und wird daher bevorzugt eingesetzt. Die SVG-Datei wird mit dem Suffix *.svg* abgespeichert. Dadurch ist es möglich, diese z.B. im Internet Explorer mit entsprechendem

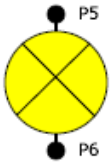

Abbildung 3.7: SVG Darstellung eines Beispiel-Schaltplanes

Plugin darzustellen. Abbildung 3.7 zeigt den zuvor in SVG erstellten Schaltplan.
Es gibt noch weitere Grundelemente und Darstellungsmöglichkeiten von SVG, auf die hier nicht weiter eingegangen wird, wie z.B. das Viereck und freistehende Pfade.

3.4 XSL

Wie im Kapitel 3.2 beschrieben, ist XML entwickelt worden, Informationen zu strukturieren, zu speichern und zu senden. In dieser Form werden somit die „reinen“ Daten gehalten. Es wird weiterhin eine Möglichkeit benötigt, die Darstellung der Informationen ebenfalls bestimmen zu können. Zu diesem Zweck entwickelte das World Wide Web Consortium die Extensible Stylesheet Language (XSL).

3.4.1 Definition XSL

XSL verbindet Sprachen, mit denen XML-Daten transformiert, gefiltert und sortiert werden können. Mittels der XML Path Language (XPath) ist es möglich, Knoten[16] in einem XML-Dokument zu adressieren. Somit ist XPath eines der wichtigsten Werkzeuge, um Daten aus einem XML-Dokument auszuwählen. Zusammenfassend besteht XSL aus drei Teilen:

- XSLT, um XML Dokumente zu transformieren
- XPath, um XML Dokumente zu adressieren
- XSL-Formatting Objects (XSL-FO), um XML Dokumente zu formatieren

[16] Die Objekte, aus denen DOM seine Baumstruktur aufbaut, um die logische Struktur eines XML-Dokuments zu präsentieren, werden Knoten genannt. XPath verwendet eine ähnliche Baumrepräsentation des XML-Dokuments.

3.4.2 Funktionsweise XSLT und XPath

Der Fokus der weiteren Erläuterungen liegt auf den beiden Teilsprachen XSLT und XPath. Mit XSLT ist es möglich, unterschiedliche XML-Strukturen entweder in weitere XML-Strukturen oder in Textdokumente zu transformieren. Ein XML-Quelldokument bildet dabei einen so genannten Dokumentenbaum, da die Elemente innerhalb des Dokumentes hierarchisch angeordnet sind. XSLT kann nun aus diesem Eingabebaum mittels Transformationen einen neuen Ausgabebaum erzeugen. Dieser Ausgabebaum kann entweder wieder ein XML-Dokument oder auch beispielsweise ein reines Textdokument oder ein PDF-Dokument sein. Die Transformation, die XSLT vornimmt, erfolgt durch sogenannte „Template-Regeln“. Eine Template-Regel besteht aus einem Muster und einem Template. Das Muster gibt an, auf welche Knoten in dem XML-Eingabebaum das jeweilige Template angewendet werden soll. Das Template wird entsprechend in den Ausgabebaum eingefügt, falls eine Übereinstimmung durch das Muster getroffen wurde. Ein solches Template kann entweder normale Literale enthalten oder auch spezielle XSLT Befehle, welche dann wieder auf Unterknoten des aktuell durch das Muster identifizierten Knotens angewendet werden können. Die Erkennung der Knoten durch die Muster erfolgt mit Hilfe von XPath, einem der drei Bestandteile von XSL. XPath kann Knoten in einem XML-Dokument anhand ihrer Position im Dokumentenbaum, anhand des Namens, des Inhalts, usw. identifizieren. Eine komplette Einführung in XPath würde den Rahmen dieser Ausarbeitung übersteigen, es soll jedoch erwähnt werden, dass XPath zwischen sieben unterschiedlichen Knotentypen in einem XML-Dokument unterscheiden kann: Wurzelknoten, Elementknoten, Attributknoten, Textknoten, Kommentarknoten, Verarbeitungsanweisungsknoten und Namensraumknoten. Da es sich bei einem XSL-Stylesheet auch um ein XML-Dokument handelt und dieses nicht eigenständig ein Ausgabedokument formatieren kann, wird ein XSLT-Prozessor benötigt. Dieser interpretiert in der Regel für einen Lauf die gesamte Vorschrift und erzeugt aus den eingehenden XML-Daten neue Strukturen. XSLT generiert somit eine Sicht auf das Dokument.
Für XSLT-Prozessoren existieren unterschiedliche Implementierungen. Sie tun in der Regel alle das Gleiche, unterscheiden sich jedoch bezüglich Geschwindigkeit, Erweiterbarkeit und der Menge der Daten, die sie verarbeiten können. Zur Veranschaulichung des Transformationsprozesses ist in Abbildung 3.8 die Transformation eines XML- in ein PDF-Dokument dargestellt.

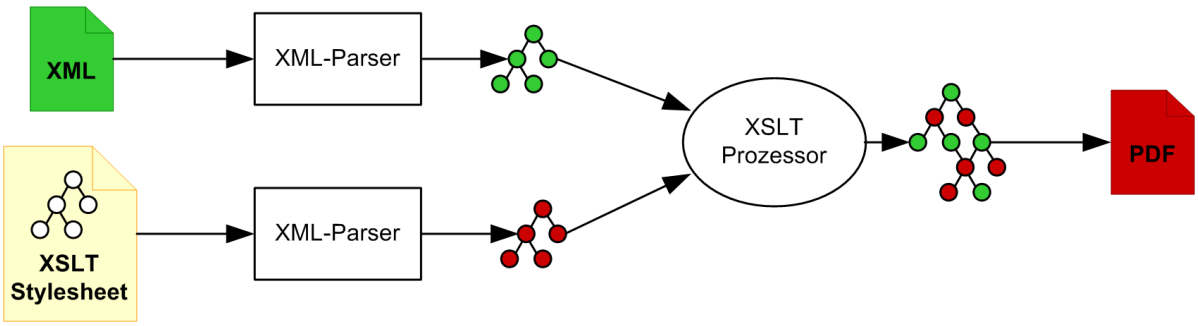

Abbildung 3.8: Grundsätzliche Arbeitsweise von XSLT

3.4.3 XSL in der Bordnetzentwicklung

Um die Vorzüge, die XML für die Bordnetzentwicklung bietet, im vollen Umfang ausschöpfen zu können, ist eine Technik notwendig, die XML-basierte Dokumente für die verschiedenen Plattformen aufbereitet. Durch XSLT lassen sich XML-Dokumente in beliebige, z.B. die für die Bordnetzentwicklung relevanten Formate HTML, PDF und SVG umwandeln und dabei umstrukturieren (aggregieren, sortieren usw.).
Bei funktionalen Programmiersprachen, zu denen XSLT zählt, erfolgt die Programmierung Rechner-unabhängig. Hier sind Parallelen zu den Vorteilen von XML erkennbar – unabhängig von Applikationen und Plattformen ist es möglich, mit XSLT Verarbeitungslogik in einem XML-Format abzubilden. Zum anderen steht den Entwicklern mit XSLT-Stylesheets ein Werkzeug zur Verfügung, um Datenbestände in einer standardisierten Weise weiter zu verarbeiten. Stylesheets bieten hier die Möglichkeit eine einheitliche Verarbeitung von Daten zu gewährleisten. Ein möglicher Nachteil von XSLT besteht in der recht gewöhnungsbedürftigen Syntax und der zum Teil schwierigen (rekursiven) Umsetzung einfacher Problemlösungen. Bedingt durch die rekursive Verarbeitung finden sich hier Programmierer besser zurecht, die Erfahrungen mit funktionalen Programmiersprachen (z.B. Prolog oder Lisp) haben. [17]

3.4.4 XSL Struktur und Syntax

Aufbauend auf den Beispielen der vorangegangenen Kapitel, wird nun eine Beispieltransformation dargestellt: Der in Abbildung 3.1 gezeigte Schaltplan soll nach SVG transformiert werden. Grundlage dafür ist die Beispiel-SVG-Graphik aus Kapitel 3.3. Abbildung 3.9 stellt

[17] vgl. [FIWO] Fitschen, Arne und Lezius Wolfgang: *Sonnige Aussichten, Einführung in XML und XSLT, Teil 2; Ein starkes Team?*, XML Magazin, ohne Jahrgang, URL:`http://www.ims.uni-stuttgart.de/projekte/corplex/paper/lezius/EinStarkesTeam2.pdf`, Datum des Zugriffs: 05.02.2004.

das XSLT-Stylesheet dar, das XML-Daten aus dem Beispiel-XML-Dokument des Kapitels 3.2 ausliest und, entsprechend der SVG-Graphik aus Kapitel 3.3, in SVG darstellt. Der Aufruf

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0"
3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4    xmlns:svg="http://www.w3.org/2000/svg">
5    <xsl:output method="xml"/>
6     <xsl:template match="/">
7      <!-- Erstellen des SVG Grundgerüsts und Aufrufen der Templates für Kindelemente -->
8      <svg:svg width="800" height="600" xmlns="http://www.w3.org/2000/svg">
9       <svg:title>SVG Transformationstest</svg:title>
10       <!-- Aufrufen der Templates für alle Kindelemente (hier für das Element "komponenten") -->
11       <svg:g>
12        <xsl:apply-templates/>
13       </svg:g>
14      </svg:svg>
15     </xsl:template>
16     <xsl:template match="komponenten">
17      <!-- Aufrufen der Templates für alle Kindelemente (hier für das Element "komponente") -->
18      <xsl:apply-templates/>
19     </xsl:template>
20     <xsl:template match="komponente">
21      <xsl:variable name="name_k" select="@name"/>
22      <!-- Komponente für Lampe zeichnen -->
23       <svg:g>
24        <svg:line x1="450" y1="130" x2="450" y2="210" style="stroke:black; stroke-width:2px;"/>
25        <svg:circle cx="450" cy="170" r="30" fill="yellow" stroke="black"/>
26        <svg:line x1="430" y1="147" x2="473" y2="190" style="stroke:black; stroke-width:1px;"/>
27        <svg:line x1="427" y1="189" x2="470" y2="147" style="stroke:black; stroke-width:1px;"/>
28       </svg:g>
29      <!-- Pins zeichnen -->
30       <svg:circle cx="450" cy="210" r="5" fill="black" stroke="black"/>
31       <svg:circle cx="450" cy="130" r="5" fill="black" stroke="black"/>
32      <!-- Pinbezeichnungen zeichnen -->
33       <svg:text x="463" y="213" style="font-size:10px;fill:black; text-anchor:bottom">
34        <xsl:value-of select="pin[1]/@id"/>
35       </svg:text>
36       <svg:text x="463" y="133" style="font-size:10px;fill:black; text-anchor:bottom">
37        <xsl:value-of select="pin[2]/@id"/>
38       </svg:text>
39     </xsl:template>
40 </xsl:stylesheet>
```

Abbildung 3.9: Beispiel-XSLT-Stylesheet für eine Transformation von XML nach SVG

eines XSLT-Prozessors bewirkt, dass dieser eine Baumstruktur des XML-Dokuments aufbaut. Das Stylesheet wird eingelesen und der Quellbaum in einen Zielbaum (in diesem Fall SVG) umgesetzt, der den Angaben des Stylesheets entspricht. Anhand der in Zeile 12 adressierten Ergebnismenge des XML-Quell-Dokumentes wird dies erläutert. Ein Konverter prüft, ob für jedes Element dieser Ergebnismenge ein passendes Template definiert ist. Tritt dies ein, werden die Angaben des entsprechenden Templates durchgeführt. In diesem Fall werden SVG-Tags erzeugt, die eine Lampe in einem Schaltplan symbolisieren. Dabei werden Attributwerte aus dem XML-Dokument ausgelesen und ebenfalls in die SVG-Graphik eingefügt. Das Ergebnis des Transformationsprozesses entspricht der in Kapitel 3.3 vorgestellten SVG-Graphik.

3.5 Prototypeinsatz bei einem Softwareentwicklungsprozess

„Ein Softwareentwicklungsprozess selbst ist eine Tätigkeit, deren Ziel die Entwicklung oder die Weiterentwicklung von Software ist.“[18]

Allerdings gibt es unterschiedliche Vorgehensmodelle, die den Entwicklungsprozess von Softwareprodukten transparenter und somit planbar, nachvollziehbar und kontrollierbar machen. Für das Softwareprodukt bedeutet dies höhere Qualität, effizientere Produktion und bessere Wartbarkeit. Diese unterschiedlichen Vorgehensmodelle haben die folgenden vier grundlegenden Gemeinsamkeiten:

- Analyse (Anforderungsspezifikation)
- Softwareentwurf und -implementierung
- Softwarevalidierung (Tests)
- Weiterentwicklung

Jede einzelne dieser Phasen wird geplant, realisiert, überprüft und dokumentiert.
Phasenübergreifend wird das gesamte Projekt geleitet (Planung, Führung, Überwachung), Qualitätssicherung gewährleistet und der Projektablauf dokumentiert.

3.5.1 Gründe für den Einsatz eines Prototyps

Da im Rahmen der Diplomarbeit ein Prototyp für die Transformation von Bordnetzdaten aus XML mittels XSLT nach SVG realisiert wurde, wird in diesem Zusammenhang etwas näher auf die Rolle eines Prototyps bei einem Softwareentwicklungsprozess eingegangen.
Vielfach fällt es dem Anwender schwer, sich ein Bild von den bereits vorhandenen Anforderungen zu machen. Hier helfen Prototypen, die einen ersten Eindruck vermitteln, eine Basis für weitere Verbesserungsvorschläge oder neue Anforderung darstellen und mögliche Missverständnisse der beteiligten Parteien deutlich machen. In manchen Vorgehensmodellen kommt der Einsatz eines Prototyps nach der Analyse zum Tragen. Prototypen sind sehr gut

[18] [UNIV2003] Universität Stuttgart: *Softwareentwicklungsprozesse und Vorgehensmodelle*, Lehrmaterial zur Vorlesung Softwaretechnik I, 2003, URL:`www.ias.uni-stuttgart.de/st1/lehrmaterialien/umdruck/kap003.pdf`, Datum des Zugriffs 26.02.2004.

geeignet, eigene Ideen zu präsentieren. Sie zeigen, wie sich der aktuelle Ansatz verhält, wie er eingesetzt werden kann und welche Möglichkeiten zusätzlich bestehen. Oftmals entstehen Ideen und Anforderungen erst nach der Präsentation eines Prototyps. Anstelle umfangreicher Projektpapiere stellt der Prototyp dadurch ein anschauliches und erfahrbares Zwischenprodukt dar.

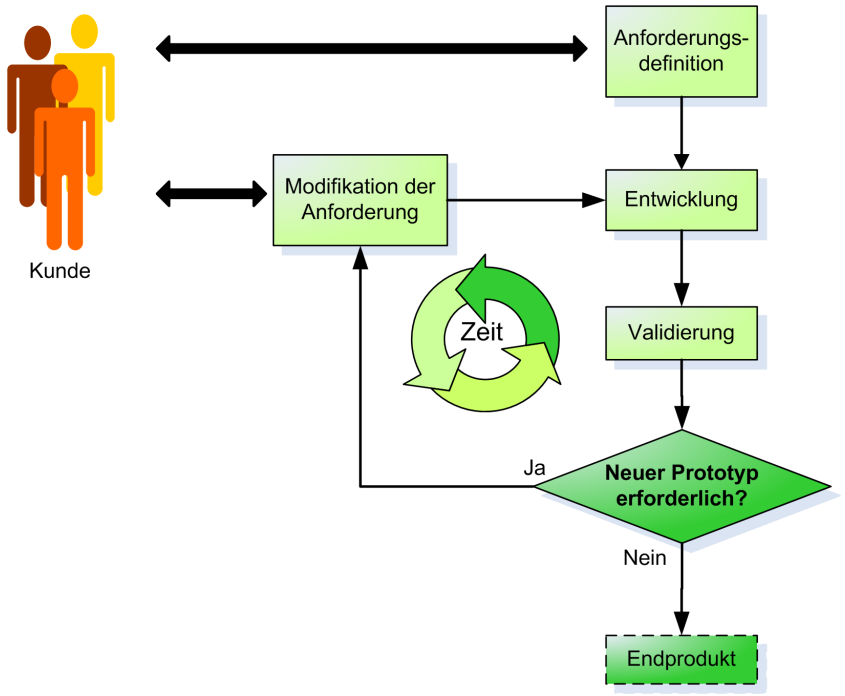

Abbildung 3.10: Beispielhaftes Vorgehensmodell bei der Softwareentwicklung mit Einsatz eines Prototyps

Abbildung 3.10 beschreibt, wie die bisher entwickelten Programmversionen korrigiert, verfeinert und weiterentwickelt werden. Nach der Realiserung des ersten Prototyps wird dieser in einem Feldtest, d.h. in der Realumgebung, begutachtet. Als Ergebnis ergibt sich die Spezifikation des zweiten Prototyps, und der Prozess beginnt von neuem. Dies wird so lange wiederholt, bis ein für die Anwender akzeptables Produkt entwickelt wurde und ein aussagekräftiges Muster der Anwendung entstanden ist. Darauf basierend erfolgt der nächste Schritt - die Implementierung der eigentlichen Software.

3.5.2 Bezug eines Prototypeinsatzes zur Diplomarbeit

Der während der Diplomarbeit zu erstellende Prototyp und das zugehörige Konzept sollen einerseits die technische Umsetzbarkeit der in Kapitel 2 genannten Zielsetzung prüfen. Andererseits sollen ein Anstoß zu neuen Ideen gegeben und die Möglichkeiten aufgezeigt werden, wie Bordnetzentwicklungsdaten zur Verfügung gestellt werden können (z.B. Recherchen, Analysen, Graphiken).

4 Analyse und Konzeption zur Erstellung des Prototyps

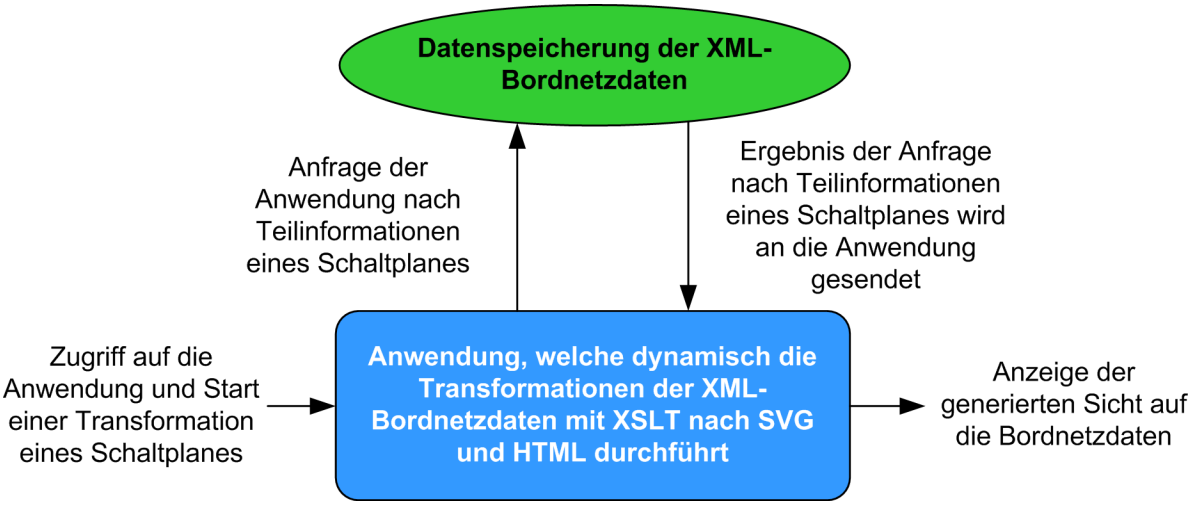

Abbildung 4.1: Ansatz für die Entwicklung des Prototyps

Eine der Aufgabenstellungen dieser Diplomarbeit ist die Entwicklung einer prototypischen Plattform, die dynamisch XML-Bordnetzdaten nach SVG und HTML transformiert. Abbildung 4.1 stellt den Ansatz dar, wie der entwicklungsbegleitende Prototyp umgesetzt werden soll. Es wird eine Anwendung erstellt, die auf die XML-Bordnetzdaten zugreift und diese dynamisch in verschiedene Ausgabeformate transformieren kann. Über eine Web-fähige Applikation sollen die Transformationen gestartet und die generierten Sichten angezeigt werden können. In diesem Kapitel werden nun die Anforderungen an den Prototyp aus Entwicklersicht analysiert und ein bestimmter Lösungsweg für die Umsetzung konzipiert.

4.1 Anforderungsanalyse

Der erste Teil der Analyse und Konzeption für die Prototypumsetzung befasst sich mit den Anforderungen, die an das zu entwickelnde System gestellt werden. Zunächst muss die Frage geklärt werden, ob die Anwendung im Internet oder einem lokalen (Firmen-) Netzwerk, dem Intranet, ablaufen soll. Grundlegend ist die Struktur von Internet und Intranet ähnlich, wobei sie sich hauptsächlich in den Bereichen Sicherheit, Benutzerkreis und Geschwindigkeit unterscheiden. Da der Prototyp als eine entwicklungsbegleitende Applikation entwickelt werden soll, kann der Benutzerkreis auf die Personen eingeschränkt werden, die entweder aktiv oder passiv an der Bordnetzentwicklung beteiligt sind. Dabei handelt es sich hauptsächlich um Firmenmitarbeiter, wodurch die Web-Anwendung innerhalb eines Firmennetzwerkes zum Einsatz kommen wird. Die Systemvorraussetzungen der Clientrechner lassen sich somit schon im Vorfeld ermitteln. Dabei handelt es sich um Desktop-PCs mit dem Betriebssystem Windows und unter anderem dem Internet Explorer 5.xx oder höher. Die minimale Geschwindigkeit des LANs wird 10 MBit/s betragen. Da die Umsetzung in Form eines Prototyps geschieht, wird kein Nutzer- und Rollenkonzept erarbeitet, sondern allen Firmenmitarbeitern der Zugriff auf die Web-Anwendung gewährt. Es werden keine Sicherheitsmaßnahmen getroffen, um die zu übertragenden Daten zu verschlüsseln.
Ein weiterer zu erörtender Aspekt ist die Frage, ob das Anwendungsprogramm, über das die Transformationen gestartet werden können, lokal auf dem Client oder zentral auf einem Server laufen soll. Beide Varianten bringen positive und negative Aspekte mit sich, über die nun ein grober Überblick gegeben wird. Zentral ablaufende Anwendungen kennzeichnen sich durch geringe Hardwareanforderungen an den Client-Rechner aus. Für Web-Anwendungen wird lediglich der Browser benötigt, der mit der eigentlichen Anwendung auf dem Server kommuniziert. Im einfachsten Fall gibt er die Resultate der Anwendung auf dem Bildschirm des Clients aus und liefert Benutzereingaben an die Anwendung zurück. Da bei einer Web-Applikation neben dem Betriebsystem nur der zuvor angesprochene Web-Browser läuft, bedeutet dies, dass alle Clients bezüglich der Anwendung einheitlich konfiguriert werden können. Die Wartung beschränkt sich somit auf den oder die Server, da auf den Clients mit wenig Aufwand die ursprüngliche Konfiguration wiederhergestellt werden kann. Bei lokal ablaufenden Anwendungen müssen für eine Aktualisierung der Software alle Applikationen auf allen Clients separat gewartet werden. Steigt die Benutzerzahl, müssen die Anwendungen auf den zusätzlichen Clients installiert werden. Eine charakteristische Eigenschaft von lokalen Anwendungen ist die Performanz. Wie bei einem lokalen Datenbestand kann direkt

auf die Festplatte oder den Arbeitsspeicher zugegriffen werden. Dadurch ist die Serverlast sehr gering, der Server muss nur die Daten liefern, die zentral gespeichert sind. Grundsätzlich können die zuvor genannten Möglichkeiten kombiniert werden. Entweder liegen die Daten und die Anwendung an zentraler Stelle, eine zentral laufende Anwendung greift auf lokale Daten zu oder eine lokale Anwendung arbeitet mit zentralen Daten.
Da die Umsetzung des Prototyps in Form einer Web-Anwendung erfolgt, fällt die Entscheidung auf ein zentrales Anwendungsprogramm. Dabei übernimmt der Internet Explorer 5.xx die Kommunikation mit dem Server. Weiterhin soll außer dem Adobe SVG-Plugin 3.0 auf den Client-Rechnern keine zusätzliche Software installiert werden, um die Anforderungen an die Client-Seite so gering wie möglich zu halten.
Schaltplandaten der Editoren LCable (2D) und Catia (3D) sollen in XML gespeichert werden, um die Grundlage der Transformationsprozesse zu bilden. Diese XML-Daten sollen konsistent gehalten werden, wobei eine Aussage getroffen werden muss, ob innerhalb der prototypischen Plattform der Datenbestand lokal oder zentral gehalten werden soll.[1] Greifen mehrere Benutzer gleichzeitig auf eine Anwendung zu, befindet sich der Datenbestand üblicherweise an einer zentralen Stelle. Arbeiten die Benutzer jedoch größtenteils an unterschiedlichen Daten oder Dateien, ist eine lokale Datenhaltung möglich. Innerhalb der gegebenen Aufgabenstellung ist es sinnvoll, aufgrund des Mehrbenutzerbetriebes der Web-Anwendung einen zentralen Datenbestand zu verwenden. Die Konsistenz der Daten kann dadurch gewährleistet werden, dass die Daten für andere Benutzer gesperrt werden, sobald sie von einem Benutzer verwendet werden. Durch die zentrale Speicherung können regelmäßig Sicherungen aller Daten durchgeführt werden. Bei einer lokalen Datenhaltung ist dies nicht immer trivial, da hierbei die Datenbestände auf vielen Rechnern verteilt sind. Ein Versionierungskonzept der XML-Bordnetzdaten ist innerhalb des Prototyps nicht vorgesehen, vorhandene Daten werden mit der aktuelleren Version überschrieben. Bei einem produktiven Einsatz kann eine Versions-Überprüfung jedoch notwendig sein.
Die Verarbeitung der XML-Bordnetzdaten soll innerhalb der XML-Familie geschehen. Für die Durchführung der Transformationen von XML mittels XSLT gibt es drei verschiedene Szenarien.[2]

[1] vgl. [SCHM2003a] Schmiderer, Johannes: *Web-Applikationen mit Unterstützung von XML*, Kapitel 2.3 Verteilte Computersysteme, URL: `http://www.schmiderer.cc/xmlwebapp/chapter_2_3.php`, Datum des Zugriffs: 12.11.2003.
[2] vgl. [TECC2001] Tecchannel: *Server- oder Client-seitige Verarbeitung*, Tecchannel, 13.03.2001, URL: `http://www.tecchannel.de/internet/670/2.html`, Datum des Zugriffs: 02.03.2004.

Client-seitig Bei der Client-seitigen Verarbeitung wird dem Web-Browser ein XML-Dokument übergeben, welches einen Verweis auf ein XSLT-Stylesheet enthält. Der Web-Browser besitzt einen internen XSL-Prozessor, kann somit die Transformationen entsprechend der Stylesheetvorgaben durchführen und stellt das Ergebnisdokument lokal dar.

Server-seitig In diesem Fall wird dem XML-Dokument durch den Server ein XSLT-Stylesheet zugewiesen. Die Transformation wird auf dem Server durchgeführt und das transformierte Dokument an den Client gesendet.

Drittprogramm Dieses Szenario beschreibt ein Programm, das beispielsweise von einem Applikationsserver gesteuert wird. Das Programm transformiert ein XML-Dokument unabhängig von Client und Server, bevor sie auf dem Server veröffentlicht werden.

Derzeit steht die Server-seitige Verarbeitung im Mittelpunkt des Interesses. Ein wichtiger Punkt dafür ist der hohe Ressourcenverbrauch bei Transformationen. Ein Server kann des Weiteren relativ einfach die Darstellungsmöglichkeiten eines Clients einlesen und die Informationen entsprechend aufarbeiten. Die Client-seitige Verarbeitung scheitert an der momentan noch mangelnden bzw. (zu) verschiedenartigen Unterstützung dieser Technik in den Browsern. Ideal wäre die Kombination aus Client- und Server-seitiger Verarbeitung. In diesem Fall kann diese entweder Client-seitig, oder Server-seitig erfolgen, immer in Abhängigkeit von der Leistungsfähigkeit des Clients. Schlussfolgernd soll die Leistung der jeweiligen Seite optimal genutzt werden, wodurch die Serverlast minimiert werden kann.
Für die Durchführung der Transformationen innerhalb der Diplomarbeit fällt die Wahl auf eine Kombination aus serverseitiger Verarbeitung und einem Drittprogramm, da der Aspekt der Ressourcenschonung des Clients sehr wichtig ist. Die Transformationszeiten selbst sollten niedrig und die Antwortzeiten schnell und je nach Komplexität akzeptabel sein.
Alternativ zu den zuvor genannten Möglichkeiten können die Anwendung oder die Daten (meist beides) auf verschiedene Standorte verteilt werden. Der Client stellt keinen Unterschied zu einem System fest, das die Daten und die Anwendung zentral bereitstellt. Die Auswahl der jeweils nächstgelegenen Server übernehmen entweder die Protokolle selbst, oder dies wird durch manuelle Konfiguration erreicht. Vorteilhaft an dieser Variante ist die Ausfallsicherheit. Fällt an einem Standort ein Server aus, können Anfragen von Clients automatisch oder manuell auf einen anderen Server umgeleitet werden. Verwendet jeder Client den ihm am nächsten gelegenen Server, führt dies zu kürzen Antwortzeiten und insgesamt zu einer geringeren Netzbelastung.

Ebenfalls nicht zu unterschätzen ist die Sicherheit gegenüber Angriffen bei einem solchen System. Fällt ein Server überlastet durch einen gezielten Angriff aus, kann ein anderer Server aus einem anderen Datenzentrum die Anfragen übernehmen. Ein weiterer Vorteil ist die Möglichkeit der Lastverteilung. Dabei wird die Last auf mehrere Server verteilt, wodurch letztendlich mehr Anfragen bewältigt werden können. Eine solche Dezentralisierung der Server hat allerdings neben höheren Kosten noch einen weiteren schwerwiegenden Nachteil: Die Datenbestände der einzelnen Server müssen ständig abgeglichen werden, um Dateninkonsistenzen zu vermeiden. Aus diesem Grund bietet sich diese Methode hauptsächlich dann an, wenn die Daten von den Clients möglichst nicht abgeändert werden. Meist wird ein Hauptserver (Master) eingesetzt, auf dem die Daten gepflegt werden. Mehrere Spiegelserver (Mirrors) gleichen dann automatisch ihre Daten mit dem Hauptserver ab.
Für die prototypische Umsetzung der Diplomarbeit wird eine verteilte Anwendungs- und Datenhaltung nicht realisiert, da zunächst die Priorität bei der Durchführung der Transformationen liegt und nicht bei der Optimierung der zu beantwortenden Anfragen. Diese Möglichkeit kann in Betracht gezogen werden, wenn ein großer Benutzerkreis parallel auf die Anwendung zugreift und die Performanz unter der Bearbeitung dieser Anfragen leidet.
Eine weitere Anforderung an die Umsetzung der genannten Aufgabenstellung ist, dass keine zusätzlichen Kosten für Hard- oder Software aufgewendet werden dürfen. Nach [SUNM2004][3] sollen Entwickler von kundenspezifischen Applikationen auch Komponenten des freien Marktes einsetzen, die von Experten des entsprechenden Teilgebiets entwickelt wurden. Durch diese Möglichkeiten ergibt sich automatisch eine höhere Produktivität. Es sprechen aber noch weitaus mehr Gründe dafür, Open Source Produkte einzusetzen. Laut [WINK2004][4] gewährleistet der Einsatz freier Software mehr Qualität, Stabilität und Portabilität[5]. Das verteilte Arbeiten von Entwicklern an Software bringt den weiteren Vorteil mit sich, dass Fehler schnell beseitigt und Programme rasch weiterentwickelt werden können. Mehrere unabhängige Entwickler erhöhen die Qualität der Software und verringern die Fehleranfälligkeit.

[3] [SUNM2004] Sun Microsystems Inc.: *Designing Enterprise Applications with the J2EE Platform*, URL: `http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/index.html`, Datum des Zugriffs: 29.03.2004.

[4] [WINK2004] Winkler, Christoph: *Open Source Projekte im XML/JAVA-Umfeld*, Zentrum der Medizinischen Informatik, Uniklinikum Frankfurt, URL: `http://www.mug-d.de/download/ft_2001/open_source_xml.pdf`, Datum des Zugriffs: 01.03.2004.

[5] Als Portabilität wird der Grad der Plattformunabhängigkeit eines Computerprogramms bezeichnet, nicht nur der bestehenden Plattformunabhängigkeit, sondern auch unter Einbeziehung des eingeschätzten Arbeitsaufwandes, der benötigt würde, um das Programm in ein vollständig plattformunabhängiges umzuwandeln.

Open Source Entwicklungen bringen mehr Planungssicherheit mit sich – Fehler können im Notfall selbst beseitigt werden. Ebenso existiert die Software auch dann noch, wenn das früher beauftragte Unternehmen nicht mehr existiert. Zudem wird verhindert, dass mehrfach Aufwände für die selbe Arbeit entstehen, indem einfach auf bereits vorhandenem Quellcode aufgebaut wird. Mehr Wettbewerb auf dem Softwaremarkt wird ebenfalls erreicht und der Monopolbildung vorgebeugt. Damit die zuvor genannten Vorteile innerhalb der Prototypumsetzung zum Tragen kommen können, resultiert daraus die Forderung des Einsatzes von Open Source Produkten, sofern diese als Alternative zu kommerziellen Produkten zur Verfügung stehen. Eine letzte Anforderung richtet sich an die Benutzeroberfläche, die unter anderem intuitiv bedienbar sein sollte.

4.2 Applikationsarchitektur

Aus den Anforderungen des Kapitels 4.1 geht hervor, dass der Prototyp in Form einer Web-Anwendung realisiert werden soll. Grundlegend basieren diese auf Client-Server-Architekturen. Für den zu erstellenden Prototyp fiel die Entscheidung auf die n-Tier-Architektur[6], wodurch die Anwendung in wiederverwendbare Komponenten zerlegt werden kann. Ausschlaggebend ist der Vorteil, dass Aufgaben auf die einzelnen Schichten ausgelagert werden können, wodurch sich die Komplexität einzelner Komponenten enorm verringert und Implementierungsaufgaben verteilt werden können. Systeme dieser Art sind des Weiteren einfach zu warten (keine besondere Client-Software, kein Austausch von Versionen) und skalierbar.
Laut [LIAN2002][7] ist die Komponententechnologie ein akzeptierter und verbreiteter technologischer Ansatz. Für die Realisierung dieser Technologie haben sich nach [GART2002][8] zwei De-facto-Standards durchgesetzt – J2EE (Java 2 Enterprise Edition), ein offener Standard auf Basis der Programmiersprache Java von Sun Microsystems, und .NET[9] , die neue Technologiegeneration von Microsoft. Im Fall von .NET handelt es sich nicht um eine Spezifikation und damit um eine offene Plattform, sondern um das Produkt eines Herstellers. Da J2EE innerhalb der gedas deutschland GmbH bisher einen großen Anteil in der Entwicklung eingenommen hat, wird dieser Standard auch innerhalb der prototypischen Umsetzung zum Einsatz kommen. Ein weiterer Grund für den Einsatz von J2EE ist, dass bei der Autorin Kenntnisse in Java und J2EE bereits vorhanden waren. Der Einsatz von .NET hätte eine zusätzliche Einarbeitungszeit in eine der unterstützten Programmiersprachen und die Architektur von .NET erfordert. Folgend wird die Java 2 Enterprise Edition als Grundlage des umzusetzenden Prototyps näher vorgestellt.

[6] Bei der n-Tier-Architektur wird eine Trennung von Präsentations-, Business-Logik- und Datenhaltungsschicht durchgeführt. Jede Schicht hat eine eigene Aufgabe, wodurch sich die Komplexität einzelner Komponenten enorm verringert und Implementierungsaufgaben verteilt werden können. Systeme dieser Art sind einfach zu warten (keine Client-Software, kein Austausch von Versionen) und sie sind skalierbar.

[7] [LIAN2002] Liantis IT COnsulting: *Moderne Konzepte und IT-Architekturen in der Softwareentwicklung*, Forschungstag der Universität Wuppertal, 21.09.2002, URL: `http://www.liantis.com/Downloads/Moderne_Konzepte_IT-Architekturen_Softwareentwicklung_Uni_Wuppertal.pdf`, Datum des Zugriffs: 16.04.2004.

[8] [GART2002] Gartner: *Gartner: J2EE und .NET bergen Risiken*, in: Tecchannel, 18.11.2002, URL: `http://www.tecchannel.de/news/allgemein/10372/`, Datum des Zugriffs: 16.12.2003.

[9] Nähere Informationen zum .NET Framework: `http://www.microsoft.com/net/`.

4.2.1 Vorstellung von J2EE als Plattform zur Umsetzung der prototypischen Web-Anwendung

1999 wurde von Sun Microsystems ein Java Development Kit (JDK) herausgegeben, das an Entwickler von verteilten, mehrschichtigen Anwendungen gerichtet ist - die Java 2 Plattform, Enterprise Edition (J2EE). Sie ist Teil von Suns umfassender Philosophie für die Entwicklung java-basierter Applikationen auf unterschiedlichsten Endgeräten - dem Sun Open Net Environment (Sun ONE). J2EE stellt lediglich eine Spezifikation dar, die den Einsatz von Java im Server-Umfeld definiert. Abbildung 4.2 verdeutlich diesen Zusammenhang graphisch.

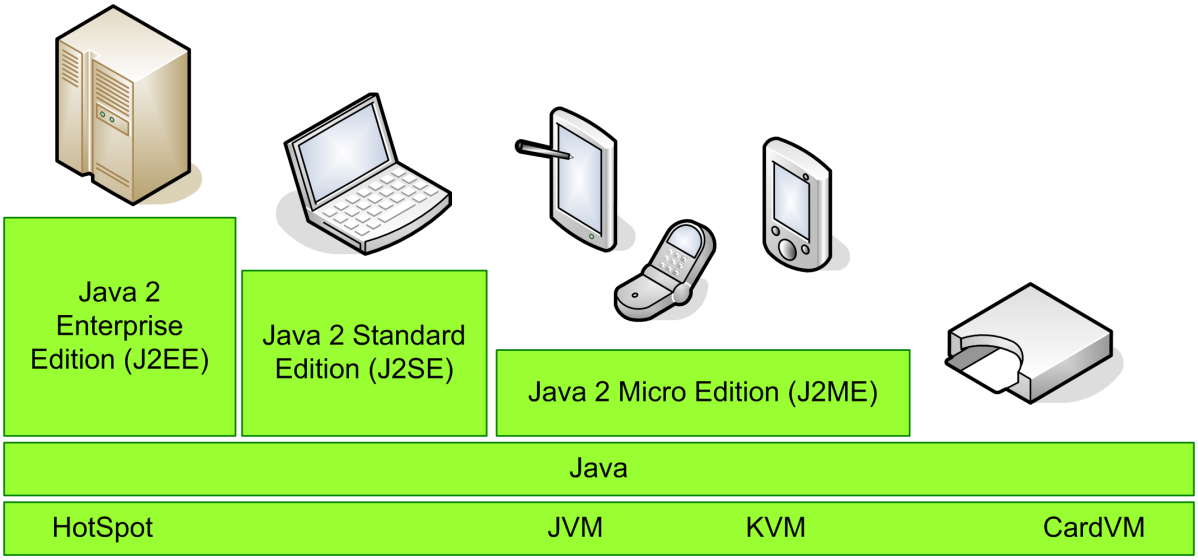

Abbildung 4.2: J2EE (Java 2 Enterprise Edition), nur ein Teil von Sun One

Diese Spezifikation schreibt insbesondere Programmierschnittstellen (APIs), Architekturkonzepte und verbindliche Eigenschaften von Applikationsservern vor. Der jeweilige Softwareanbieter, wie beispielsweise BEA, IBM oder Sun selbst, ist dabei selbst für die konkrete Implementierung verantwortlich. Für die weiteren Betrachtungen werden J2EE und Sun ONE synonym verwendet, da unter dem Begriff J2EE oft die gesamte Java-Entwicklungsplattform zusammengefasst wird.
Mit Hilfe von J2EE ist es also möglich, verteilte, serverseitige und skalierbare Anwendungen auf Basis modularer Komponenten zu erstellen, mit der Option, auch auf bestehende (Alt-) Systeme zuzugreifen. J2EE baut auf einer einzigen Sprache auf – Java. Dabei hebt sich J2EE, wie Java im Allgemeinen, durch seine Plattform- und Herstellerunabhängigkeit

hervor. Es setzt weiterhin auf bereits etablierten Standards wie z.B. JDBC[10] oder CORBA[11] auf und ermöglicht dem Entwickler den Zugriff auf weitere Funktionalitäten wie z.B. Enterprise Java Beans[12], Java Servlets[13], JSP[14] und XML. J2EE basiert auf einer 3-Tier-Architektur und ist netzwerkorientiert, da es auf das Internet als zentrale Infrastruktur zurückgreift. Der Web-Browser wird als bevorzugtes Schnittstelle zum Anwender verwendet. Zur Erweiterung von J2EE sind umfangreiche Java-Bibliotheken verfügbar, sowie vielzählige Entwurfsmuster für die Realisierung der Wiederverwendbarkeit. Suns Technologie rentiert sich längerfristiger durch niedrigere Pflegekosten und bessere Wiederverwendbarkeit, als Microsofts .Net. [15] Für weiterführende Informationen zu J2EE wird an dieser Stelle auf die Web-Seite `http://java.sun.com/j2ee/index.jsp` von Sun Microsystems verwiesen.

4.2.2 J2EE und XML Technologien

Mit J2EE als Basis des zu erstellenden Prototyps steht nun ein Standard zur Verfügung, mit dem n-Tier-Applikationen entwickelt werden können. Gemäß [LANE2001][16] baut bei diesen Applikationen die Präsentationsschicht bisher fast ausschließlich auf HTML auf, wobei die Anwender hauptsächlich den Web-Browser verwenden. Im Hinblick auf zukünftige Applikationen ist es wichtig, sich von HTML als einziges Darstellungsformat zu lösen. Mittlerweile gibt es unterschiedlichste Geräte, die Daten empfangen und senden können (z.B. PC, Handy, PDA usw.). Daher ist es notwendig, den Inhalt von der Präsentationslogik zu entkoppeln, um somit unabhängig vom Gerät des Empfängers zu werden. Eine Lösung lautet

[10] JDBC ist die inoffizielle Abkürzung für Java Database Connectivity und bezeichnet einen Satz von Klassen und Methoden, um relationale Datenbanksysteme von Java zu nutzen.

[11] Die Common Object Request Broker Architecture kurz CORBA ist eine objektorientierte Middleware, die plattformübergreifende Protokolle und Dienste definiert und von der Object Management Group (OMG) entwickelt wird. CORBA ermöglicht das Erstellen verteilter Anwendungen in heterogenen Umgebungen.

[12] Enterprise Java Beans (EJB) sind Teil der zentralen Elemente der J2EE Spezifikation. EJBs dienen der standardisierten vereinfachten Entwicklung komplexer mehrschichtiger verteilter Softwaresysteme mittels Java. EJBs sind Java Beans mit zusätzlicher Funktionalität, besonders Transaktionseigenschaften (ACID), Persistenz und Sicherheit.

[13] Ein Servlet ist ein Java Programm (Applet), welches in einer speziellen Umgebung auf dem Server ausgeführt wird.

[14] Java Server Pages (JSP) sind, wie durch die Bezeichnung bereits impliziert, eine Server-seitige Technologie, die es erlaubt dynamische Web-Anwendungen zu gestalten, indem spezielle Tags in gewöhnliche Markup Sprachen, wie HTML, XHTML, XML und ähnliche, eingebunden werden.

[15] vgl. [NAMI2002] namics ag: *.NET vs. J2EE In welche Plattform investieren?*, namics Whitepaper, August 2002, URL: `http://www2.namics.com/files/kno_whi_net.pdf`, Datum des Zugriffs: 15.04.2004.

[16] [LANE2001] Lane, Eoin: *Add XML to your J2EE applications*, in: JavaWorld, Februar 2001, URL: `http://www.javaworld.com/javaworld/jw-02-2001/jw-0209-xmlj2ee.html`, Datum des Zugriffs: 15.04.2004.

XML, wobei XML-Dateien den Inhalt repräsentieren und die Transformationssprache XSLT die Präsentationsschicht definiert. Mittels XSLT können XML-Dokumente in verschiedene Formate transformiert werden, wie z.B. XHTML, WML, SVG, MathML usw. Somit können aus einer Datenbasis mit unterschiedlichen XSLT Stylesheets verschiedene Ausgabeformate für verschiedene Endgeräte produziert werden.
Diese Technik wird innerhalb des Prototyps eingesetzt, um in XML gehaltene Bordnetzdaten (der Inhalt) mittels XSLT in die Ausgabeformate SVG und HTML zu transformieren. Eine Möglichkeit, dies umzusetzen besteht darin, einen Open Source Application Server XML-fähig zu machen. Dadurch ergibt sich einerseits ein gewisser Freiraum für die Entwicklung von Applikationen, andererseits ist der Entwicklungsaufwand höher als bei einem vorgegebenem Framework[17]. Bei einem Framework kann auf fertige Dienste und Bausteine zurückgegriffen werden, höhere Produktivität schneller erreicht und Aufgaben innerhalb eines Projektes besser verteilt werden. Aus diesen Gründen wird oft auf vorgefertigte Frameworks von Drittfirmen oder Open-Source-Gemeinde zurückgegriffen, die sich für die entsprechende Anwendung eignen. Eine weitere Motivation für den Einsatz von Frameworks ist der Aspekt, dass die Architektur vieler Web-Frameworks auf J2EE Entwurfsmustern (Patterns) basiert. Diese geben Lösungsansätze für den effizienten Einsatz von J2EE Technologien in der Praxis. Relevant für den zu entwickelnden Prototyp ist hierbei das Model View Controller (MVC) Pattern, dessen vorrangiges Ziel es ist, die Datenhaltung (Model), die Datenrepräsentation (View) und die Datenmanipulation (Controller) voneinander zu trennen. Die Wiederverwendung einzelner Komponenten der Architektur wird gefördert und die Software ist leichter zu warten. Durch die Entkopplung von Benutzerschnittstelle, Geschäftslogik und Daten ist es möglich, Aufgaben spezialisierten Teams zuzuordnen. Weitere Vorteile, sowie nähere Erläuterungen zum MVC Pattern sind unter `http://java.sun.com/blueprints/patterns/MVC-detailed.html` zu finden.
Die Entscheidung, ob eine eigene Applikation entwickelt, oder ein „fertiges“ Framework eingesetzt wird, fällt nach Abwägung der Vor- und Nachteile zugunsten des Framework-Einsatzes. Zwar sind Framworks nur schwerlich austauschbar und erfordern eine gewisse Einarbeitungszeit, dennoch überwiegt der positive Nutzen eines Framework-Einsatzes.

[17] Ein Framework ist ein System, welches auf Basis eines Grundgerüsts verschiedene Komponenten für ein bestimmtes Anwendungsgebiet bereitstellt, die zur individuellen Anpassung in das System eingebunden werden können. Die Einbindung der vorgefertigten Komponenten soll für den Nutzer ohne Spezialwissen möglich sein.

4.2.3 XML Publishing Frameworks

Java ist eine mittlerweile oft verwendete und populäre Programmiersprache, die eine einfache Schnittstelle zu XML-Werkzeugen bietet. Mit dem Java Servlet API ist weiterhin ein einfaches Mittel vorhanden, um Web-Anfragen und -Antworten zu bearbeiten.[18] Um ein geeignetes und stabiles Framework für die Umsetzung des Prototyps auswählen zu können, wurden folgende Auswahlkriterien aufgestellt:

Plattformunabhängigkeit Von Plattform-spezifischen Technologien sollte Abstand genommen werden. Ein Publishing Framework muss Clients auf den verschiedensten Plattformen „bedienen".

Unterstützung und Austauschbarkeit verschiedener XML Parser und Prozessoren Es sollte sichergestellt sein, dass das Framework eine Vielfalt an XML Parsern und Prozessoren unterstützt und diese auch nach Belieben ausgetauscht werden können.

Entwickler überwachen XML Technologien Die Entwickler eines Frameworks sollten sich idealerweise mit den Spezifikationen der verschiedenen XML Technologien befassen. Dies lässt schlussfolgern, dass bei Änderungen des Frameworks die Aktualität der XML Technologien bedacht wird – ein bedeutender Faktor für Langlebigkeit eines Frameworks.

Weitere Produkte des Anbieters Die Stabilität eines Produktes kann oft an der Stabilität anderer Produkte desselben Anbieters gemessen werden. Oftmals wird eine ganze Reihe von Produkten von einem Anbieter freigegeben. Wenn diese Produkte keine Unterstützung für SAX 2.0 oder DOM Level 2 bieten, oder erst in Version 1.0 oder 1.1 vorliegen, sollte abgewartet werden, bis dieses Framework ausgereifter ist.

Referenzapplikationen Einer der wichtigsten Punkte ist die Frage nach einer Referenzapplikation des Frameworks. Es sollten Applikationen existieren, die belegen, dass das Framework produktiv eingesetzt werden kann und wird.

Kosten Wie im Kapitel 4.1 ausgeführt, dürfen für die Prototypentwicklung keine zusätzlichen Kosten entstehen. Daher sollte für den Einsatz des XML Publishing Frameworks ein kostenloses Open Source Produkt verwendet werden.

[18] vgl. [MCLA2000] McLaughlin, Brett: *Java and XML*, O'Reilly, Juni 2000, Kapitel 9.

Ebenfalls wichtig ist die Möglichkeit, aus einer Datenbasis verschiedene Ausgabeformate zu erzeugen, da auf dieser Eigenschaft der zu entwickelnden Applikation das Hauptaugenmerk liegt. Dies wird auch Multichanneling genannt.
Folgend werden zwei Frameworks näher betrachtet, die für die Umsetzung der Prototyp-Web-Anwendung in Frage kommen. Dabei handelt es sich um Jakarta Struts und Apache Cocoon. Jakarta Struts[19] ist ein Framework der Apache Software Foundation für die Erstellung von Web-Anwendungen mit Servlets und Java Server Pages. Bei Cocoon[20] handelt es sich um ein Web Publishing Framework der Apache Software Foundation, das intensiv XML-Standards und Technologien einsetzt. Es eignet sich besonders für Web-Anwendungen, die XML als Datenquelle verwenden, auf XML-Technologien wie XSLT, XPath, XQuery oder XLink basieren und verschiedene Ausgabeformate wie HTML, WML, PDF oder SVG liefern. In Tabelle 4.1 werden diese zwei Frameworks anhand der zuvor besprochenen Kriterien miteinander verglichen.

Kriterien	**Jakarta Struts**	**Apache Cocoon**
Versionsnummer	Version 1.1	Version 2.1.x
Weitere Produkte des Herstellers	Avalon, Maven, HTTP Server, Ant uvm.	siehe Jakarta Struts
Unterstützte Plattformen	Windows und Unix-Plattformen	Windows und Unix-Plattformen
Unterstützte XML Parser und Prozessoren	Parser: Xerces; Stylesheet Prozessor: Xalan 1.2	Parser: Xerces; Stylesheet Prozessor: Xalan 1.2
Austauschbarkeit von XML Parsern und Prozessoren	Parser: JAXP konforme Parser beliebig austauschbar; Prozessor: keine Angaben	Parser und Prozessor: aufgrund starker Komponentenorientierung Austauschbarkeit einfach möglich
Entwickler beobachten XML Technologieentwicklungen	ja und nein (letzte Version ist vom Juni 2003)	ja
Referenzapplikationen	siehe `http://www.simonpeter.com/techie/java/struts/sites.html`	siehe `http://cocoon.apache.org/link/livesites-2.1.html`
Kosten	kostenloses Open Source Produkt	kostenloses Open Source Produkt

Tabelle 4.1: Gegenüberstellung der Frameworks Jakarta Struts und Apache Cocoon

[19]Weitere Informationen zu Jakarta Struts: `http://jakarta.apache.org/struts/`.
[20]Weitere Informationen zu Apache Cocoon: `http://cocoon.apache.org/`.

Aus der Tabelle 4.1 wird ersichtlich, dass beide Frameworks verschiedene Plattformen unterstützen und somit nicht auf eine Technologie spezialisiert sind. Struts ist nicht von Haus aus multichannelfähig, jedoch ist es nach [MEBO2002][21] möglich, durch eigene Erweiterungen Transformationen von XML mittels XSLT durchzuführen. Eine andere Variante Transformationen von XML innerhalb des Frameworks Struts durchzuführen, ist stxx[22], das eine Erweiterung von Struts darstellt. Mit diesem können XML und XML-Transformationstechnologien unterstützt werden, ohne dass die Funktionalitäten des Frameworks verändert werden müssen. Bei Cocoon hingegen wird die Möglichkeit, XML in andere Formate zu transformieren, bereits mitgeliefert. Durch eine extrem komponentenorientierte Architektur dieses Systems können Erweiterungen und Anpassungen jederzeit durchgeführt werden. Des Weiteren ist Cocoon ein Paradebeispiel für XML-basierte Web-Frameworks, bei denen XML für die Trennung von Layout, Logik und Daten verwendet wird.
Für die Umsetzung der Transformationen der prototypischen Web-Anwendung fiel die Wahl auf das Framework Cocoon, da bei diesem keine zusätzlichen Installationen oder Erweiterungen durchgeführt werden mussten und unmittelbar mit der Realisierung des Prototyps begonnen werden konnte. Dabei wurde Cocoon in der Version 2.1.2 in Form eines Servlets angewendet, da sich dies durch eine problemlose Installation ohne aufwändige Konfigurationen auszeichnet. Zur Ausführung des Servlets ist ein Servlet Container (Engine) erforderlich. Apache Tomcat[23] ist die offizielle Referenzimplementierung der Java Servlet- und Java Server Pages–Spezifikationen, wobei der Servlet Container in der zur Zeit des Diplomprojekts aktuellen Version 4.1.27 zum Einsatz kam.

4.2.4 Zusammenfassung Applikationsarchitektur

Innerhalb dieses Kapitels wurde die Plattform J2EE als Grundlage der Umsetzung der prototypischen Web-Anwendung vorgestellt. Daraufhin wurde J2EE in Verbindung mit XML-Technologien besprochen. Dabei wurde ausgeführt, weshalb bezüglich der Präsentationsschicht auch andere Ausgabeformate neben HTML bedacht werden sollten. Die Umsetzung dieses Ansatzes kann einerseits durch eine eigene J2EE-Anwendung erfolgen, andererseits durch den

[21] [MEBO2002] Mercay, Julien und Bouzeid, Gilbert: *Boost Struts with XSLT and XML*, Java World, Februar 2002, URL: `http://www.javaworld.com/javaworld/jw-02-2002/jw-0201-strutsxslt.html`, Datum des Zugriffs: 03.02.2004.
[22] Weiter Informationen zu stxx (Struts for Transforming XML with XSL): `http://stxx.sourceforge.net/`.
[23] Weitere Informationen zu Apache Tomcat: `http://jakarta.apache.org/tomcat`.

Einsatz eines bestehenden Frameworks. Da unter anderem aus zeitlichen Gründen eine eigene Anwendung ausgeschlossen werden musste, wurden für den Einsatz geeignete einzusetzende Frameworks untersucht. Diese wurden anhand definierter Kriterien verglichen. Das Framework, das nun zum Einsatz kommt, heißt Cocoon und ist ein Open Source Web-Publishing Framework der Apache Software Foundation.

4.3 Datenhaltung

Dieser dritte Teil der Applikationsanalyse beschäftigt sich mit der Datenhaltung. In Kapitel 4.1 wurde beschlossen, dass die XML-Bordnetzdaten an einer zentralen Stelle gespeichert werden. Dafür werden nun unterschiedliche Möglichkeiten vorgestellt.

4.3.1 Anforderungen an die Speicherung der XML-Bordnetzdaten

Bevor die Wahl auf eine Speichermöglichkeit für die XML-Bordnetzdaten fällt, müssen verschiedene Anforderungen bedacht werden. Dazu gehört, dass die Bordnetzdaten nicht in einer großen XML-Datei gehalten werden, sondern innerhalb mehrerer kleinerer Dateien. Daher müssen eine Vielzahl an XML-Dokumenten persistent abgelegt werden können. Eine hierarchische Speicherung ist ebenso notwendig, um die Projektstruktur der Bordnetzdaten abbilden zu können. Die XML-Dokumente müssen auf Wohlgeformtheit überprüft werden können, um sicherzustellen, dass keine unvollständigen, oder fehlerhaften Daten weiterverwendet werden. Von Vorteil ist ebenso die Unterstützung weiterer Standards aus der XML-Sprachfamilie, mit denen Abfragen auf die XML-Borddaten durchgeführt werden können, wie beispielsweise XQuery und XPath. Die im Datenmodell sichtbare komplexe Bordnetzdaten-Struktur bedingt zudem die Forderung nach kurzen Zugriffszeiten auf XML-Elementteile oder Attributwerte. Idealerweise sollen dafür Indizes eingesetzt werden.

4.3.2 XML-Speichermöglichkeiten

> „Datenbanken haben sich im Laufe der Jahre als ideales Werkzeug zur Verwaltung und Pflege großer Datenvolumen entwickelt. Im Zuge der rasanten Verbreitung von neuen XML-Standards stehen Entwickler sehr schnell vor dem Problem, die Vorteile beider Technologien zu nutzen. Dazu hat heute nahezu jedes Datenbank-Produkt und Datenbank-Tool den Begriff „XML“ in seine Featureliste aufgenommen.“ [24]

Daraus resultiert eine Vielfalt an Möglichkeiten, XML-Daten zu speichern. Folgende Übersicht nach [BOUR2004b] [25] soll einen Einblick geben :

[24] [MART2001] Martin, Lars: *Entwicklung neuer Standards für XML-Datenbanken - Teamgeist!*, Linux-Magazin, April 2001, URL: `http://www.linux-magazin.de/Artikel/ausgabe/2001/04/xmldb/xmldb.html`, Datum des Zugriffs: 27.09.2003.

[25] vgl. [BOUR2004b] Bourret, Ronald: *XML and Databases*, URL: `http://www.rpbourret.com/xml/XMLAndDatabases.htm`, Datum des Zugriffs: 02.04.2004.

Dateisystem Das Dateisystem ist die einfachste Möglichkeit, XML-Dokumente zu speichern. Dabei werden diese als Datei abgelegt. Diese Variante ist gut geeignet, wenn eine große Menge an XML-Dokumenten erzeugt wird, die später gar nicht nur mehr selten verändert werden müssen. Jedoch sind hierbei weder die direkte Bearbeitung auf Elementebene, noch Versionskontrollen oder modulares Sperren möglich. Dies obliegt dem Verwalter des Dateisystems.

Flatstream Bei dieser Methode wird das XML-Dokument als Ganzes in einer relationalen Datenbank in Form eines BLOB (Binary Large Object) oder CLOB (Character Large Object) gespeichert. Das XML-Dokument entspricht in diesem Fall einem Spalteneintrag (Zelle) in einer Tabelle. Diese Variante eignet sich gut für Anwendungen, die beliebige Anzahl von XML-Dokumenten umfassen, diese jedoch nur selten laden bzw. speichern und nur als Einheit behandeln. Auch hier ist keine Bearbeitung auf Elementebene möglich, jedoch können die Vorzüge, die eine Datenbank mit sich bringt, zum Einsatz kommen, darunter Transaktionskontrolle und Mehrbenutzerbetrieb.

Mapping Beim so genannten Mapping werden die Struktur und der Inhalt eines XML-Dokumentes mit Hilfe einer zusätzlichen Software auf vorhandene Strukturen, beispielsweise eine Tabelle in einer relationalen Datenbank, abgebildet. Somit kann ein XML-Dokument mehreren Zeilen in verschiedenen Tabellen in einer Datenbank entsprechen. Für XML-Dokumente mit flacher Hierarchie und regelmäßiger Struktur eignet sich diese Speichervariante bestens. Eine Bearbeitung auf Elementebene ist möglich.

Native XML-Datenbanken Bei der Speicherung von XML-Dokumenten in einer nativen XML-Datenbank wird die Struktur eines XML-Dokumentes in ein für das XML-Format ausgelegtes Speicherformat übertragen. Eine Bearbeitung auf Elementebene ist bei dieser Variante möglich, und XML-Standards werden indirekt unterstützt. Weitere Eigenschaften sind wie bei den relationalen Datenbanken (Unterstützung abhängig je nach Produkt) Transaktionskontrolle, Mehrbenutzerbetrieb, verschiedene Abfragesprachen (z.B. XPath, XQL), APIs usw. Ein weiterer wichtiger Punkt ist das „Round-Tripping“. Dies bedeutet, dass ein Dokument sogar exakt in der Form wiederhergestellt werden kann, in der es in die Datenbank eingepflegt wurde.

Persistent DOM Persistent DOM ist eine Speicherungslösung, die XML-Dokumente mit einer Transaktionsverwaltung – auch im Mehrbenutzerbetrieb – sichert. Dabei werden XML-Dokumente in einem kompakten Binärformat gespeichert, auf dem mit den

Methoden der W3C-normierten DOM-API Operationen umgesetzt werden können. Dadurch werden Schreibzugriffe auf beliebig große XML-Dokumente möglich und es wird ohne Pufferung auf einem Standard-PC ein hoher Durchsatz erreicht.[26]. Zusätzlich bringt „Persistent DOM“ einen Zugang zur PDOM-API mit, welche die wichtigen Funktionen zur Versubstantivierung bereitstellt.

Die Speicherung der XML-Bordnetzdaten in einem Dateisystem eignet sich theoretisch, da der Export der Bordnetzdaten aus LCable nicht innerhalb von kurzen Abständen durchgeführt wird. Weiterhin können die XML-Daten hierarchisch in einer Dateistruktur abgelegt werden. Jedoch fehlt eine Bearbeitungsmöglichkeit auf Element- oder Attributebene, sowie die automatische Prüfung auf Wohlgeformtheit der XML-Dokumente. XML-Bordnetzdaten des Weiteren als Flatstream in einer relationalen Datenbank zu speichern, scheidet ebenfalls aus: Unter anderem werden bei der zu entwickelnden Plattform dynamische Sichten auf die Bordnetzdaten erstellt, wodurch es zu häufigen Zugriffen auf die Daten kommt. Wie zuvor schon erwähnt, ist die Struktur eines Bordnetzes sehr komplex. Beim Mapping müsste die Struktur zur Übertragung in Tabellenzellen und -spalten aufgespalten werden. Diese Variante hat den Vorteil, dass dadurch eine Berarbeitung auf Elementebene möglich würde. Jedoch ist fraglich, ob die komplexe Struktur des Bordnetztes komplett wiederhergestellt werden kann. Werden XML-Bordnetzdaten hingegen in einem für XML ausgelegten Format gespeichert, ist beispielsweise die Bearbeitung auf Elementebene möglich und verschiedene XML relevante Standards können verwendet werden. Mittels der XML-Daten-Speicherung mit dem Persistent DOM sind beliebig große XML-Dokumente bearbeitbar, wobei XML-Datenbanken je nach Produkt große Dokumente unterschiedlich handhaben. Weiter zu beachten ist, dass die Umsetzung der Architektur des Prototyps bisher auf XML für die Abbildung der Bordnetzdaten beruht, andererseits auf XSLT zur Durchführung der Transformationen und auf SVG und HTML zur Darstellung der Bordnetzdaten. Somit werden bisher durchgängig Standards der XML-Sprachfamilie eingesetzt. Ein weiterer Vorteil des Einsatzes einer XML-Datenbank wäre daher, dass mit denselben Standards weitergearbeitet werden kann und ein Technologiebruch vermieden wird. Da für die Prototyperstellung keine Lizenzgebühren oder Anschaffungskosten anfallen dürfen, entfällt der Einsatz einer relationalen Datenbank. Nach [BOUR2004b] gibt es derzeit keine kostenfreie relationale Datenbank, die um Funktionalitäten zur Speicherung von XML-Dokumenten erweitert ist. Im Bereich der XML-Datenbanken existieren jedoch

[26] vgl. [FRAU2004] Fraunhofer Institut für integrierte Publikations- und Informationssysteme: *PDOM und XQL-Prozessor, der XML-Datenserver*, `http://www.ipsi.fraunhofer.de/oasys/projects/pdom/index_d.html`, Datum des Zugriffs: 02.04.2004.

verschiedene kostenfreie Produkte. Aufgrund der verschiedenen Vorteile für die Speicherung der Bordnetzdaten in XML ist eine relativ klare Tendenz zum Einsatz einer XML-Datenbank innerhalb des Prototyps zu erkennen.

4.3.3 Vergleich der XML-Datenbanken Xindice und eXist

Unter [BOUR2004a][27] ist eine Übersicht der momentan verfügbaren XML-Datenbanken zu finden. Nach näheren Betrachtungen bezüglich der Eignung für die Prototypumsetzung, der Kosten und der Aktualität der verfügbaren Distributionen, konnten Apache's Xindice und Wolfgang Meier's eXist XML-Datenbank in die nähere Auswahl gezogen werden.

Produktname	**Xindice**	**eXist**
Homepage	`http://xml.apache.org/xindice`	`http://exist.sourceforge.net/`
Version	1.0 (stabile Version), 1.1b3 (Entwicklungsversion Dezember 2003)	0.9.2 (stabile Version November 2003)
Lizenz	Open Source	Open Source
Unterstützte Plattformen	in Java entwickelt - plattformunabhängig	in Java entwickelt - plattformunabhängig
Benötigte Software vor der Installation	Sun Java SDK Version 1.3 oder höher, Tomcat Servlet Container Version 4.1.12 oder höher	Sun Java SDK Version 1.4
Dokumentation		
Verfügbare Formate	HTML, PDF	HTML
Qualität	teilweise nicht aktuell und irreführend	kurz gehalten, aber sehr hilfreich

Tabelle 4.2: Vorstellung der XML-Datenbanken Xindice und eXist

In den Tabellen 4.2, 4.3, 4.4, 4.5 und 4.6 werden die Fähigkeiten beider Produkte miteinander verglichen. [28]

[27] vgl. [BOUR2004a] Bourret, Ronald: *XML Database Products*, URL: `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`, Datum des Zugriffs: 02.04.2004.

[28] vgl. [APAC2003a] Apache Software Foundation: *Apache Xindice*, URL: `http://xml.apache.org/xindice/`, Datum des Zugriffs: 05.10.2003; [MEIE2003b] Meier, Wolfgang: *eXist Open Sorce Database*, URL: `http://exist.sourceforge.net/`, Datum des Zugriffs: 01.10.2003; [FRIE2003] Frieb Werner: *XML Databases compared*, Institute for Software Technology and Interactive Systems, Vienna University of Technology, URL: `http://www.studierstube.org/world/xml_databases_compared.html`, Datum des Zugriffs: 20.12.2003.

	Xindice	**eXist**
SAX und DOM	ja, in Java implementiert	ja, implementiert in Java als Teil der XML:DB API
XML:DB API	ja, Core Level 1 Implementierung, + XUpdateQueryService, + CollectionManagementService	ja, Core Level 1 Implementierung , + UserManagementService, + DatabaseInstanceManager, + IndexQueryService
JAXP:	nein	nein
JDOM:	nein	nein

Tabelle 4.3: Vergleich der API-Unterstützung der XML-Datenbanken Xindice und eXist

Unterstützte Netzwerkprotokolle	**Xindice**	**eXist**
HTTP	nein	ja, im Standalone-Modus
XML-RPC	ja	ja
WebDav	ja	ja
SOAP	nein	ja, im Servlet-Modus

Tabelle 4.4: Vergleich der unterstützten Netzwerkprotokolle der XML-Datenbanken Xindice und eXist

Günstig ist, dass die Systemkonfiguration und Administration bei Xindice mittels einer graphischen Benutzeroberfläche durchgeführt wird und dadurch die Bedienung erleichtert wird. Bei eXist geschieht dies ebenfalls mittels einer integrierten und kombinierten graphischen Kommandozeilen- und Benutzer-Applikation, die auf der XMLDB:API[29] basiert. Daten werden bei eXist über diese Applikation importiert und exportiert, wobei bei Xindice ein zusätzlicher Kommandozeilen-Client verwendet werden muss. In Abbildung 4.3 werden eXist, Xindice anhand der Verarbeitungsgeschwindigkeit von XPath-Anfragen verglichen.[30] Es ist zu erkennen, dass eXist, mit und ohne Verwendung der XPath-Erweiterungen, in jedem Fall deutlich schneller Anfrageergebnisse liefert, als Xindice. Somit kommt eXist der Forderung nach, schnell auf Elemente und Attribute zugreifen zu können. Vorteilhaft bei beiden Datenbanken ist die umfassende Unterstützung der standardisierten APIs DOM, SAX und XML:DB

[29] Die API XML:DB wurde entwickelt, um einen allgemeinen Zugang für native XML-Datenbanken zu ermöglichen. Mit der API können Anwendungen zum Speichern, Lesen, Ändern und Abfragen der in einer XML-Datenbank gespeicherten Daten erstellt werden.

[30] vgl. [MEIE2003c] Meier, Wolfgang: *eXist: An Open Source Database*, Darmstadt University of Technology, URL: `http://exist-db.org/webdb.pdf`, Datum des Zugriffs: 15.10.2003.

Unterstütze XML Standards	**Xindice**	**eXist**
XPath	ja, implementiert durch Xalan	ja, in erweiterter Form
XQuery	ja	ja, aber noch unvollständig
XSD	nein	ja
XUpdate	ja, mit Hilfe der Lexus Implementation	ja
SiXDML (Simple XML Data Manipulation Language)	ja, durch sixdml von sourceforge	nein
XPointer	nein	nein
XLink	nein	nein
Xinclude	ja, teilweise	ja

Tabelle 4.5: Vergleich der unterstützten Formate der XML-Datenbanken Xindice und eXist

API. Darüber hinaus bietet eXist Applikationen für die Administration, automatische Indexgenerierung und die Möglichkeit, in eine Java-Applikation eingebettet zu werden. Nachteilig wirkt sich bei Xindice die Tatsache aus, dass lediglich kleine bis mittelgrosse Dokumente (bis 5 MB) sinnvoll bearbeitet werden können. Die Dokumente der XML-Bordnetzdaten können weitaus größer werden. Weiterhin unterstützt Xindice im Gegensatz zu eXist keine Anfragen über gesamte bzw. untergeordnete Collections. Trotz der fehlenden Unterstützung manueller Indizierung und einer nicht komplett fertiggestellten XPath-Unterstützung wird für die Speicherung der XML-Bordnetzdaten die XML-Datenbank eXist von Wolfgang Meier zum Einsatz kommen.

Eigenschaften der Datenbank	**Xindice**	**eXist**
Schemunterstützung	nein	ja, aber nicht zwingend
Unterstützung von Collections	ja, aber keine Unterstützung für XPath-Anfragen über Collections	ja
Indizes	ja, aber nur Benutzerindizes	ja, Volltextindizes
Unterstützung von Transaktionen	nein	nein
Mechanismus zur Authorisierung	nein	ja, Unix-ähnliche Zugangsrechte
Mulituser-Zugang, Nebenläufigkeit	unklar	ja
Granularität der Updates	logisch auf Knotenebene, physisch auf Dokumentebene	auf Knotenebene
Sicherungs- und Wiederherstellungsstrategie	nein (DB herunterfahren und komplette DB kopieren und anderweitig speichern)	ja, über Kommandozeile oder graphische Benutzeroberfläche

Tabelle 4.6: Vergleich der Datenbankeigenschaften der XML-Datenbanken Xindice und eXist

XPath Query	**eXist**	**eXist + extensions**	**Xindice**
/movie[.//genre='Drama']//credit[@role='directors']	3.44	1.14	10.62
/movie[genres/genre='Western']/title	0.79	0.23	1.39
/movie[languages/language='English']/title	1.45	0.97	34.18
/movie[.//credit/@charactername='Receptionist']	3.12	0.21	27.04
/movie[contains(.//comment, 'predictable')]	2.79	0.20	25.75
/movie[.//credit='Gable, Clark']	4.47	0.35	0.38
/movie[.//languages/language='English']/title[starts-with(.,'42nd Street')]	1.63	0.32	17.47
/movie[languages/language='English' and credits/credit='Sinatra, Frank']	5.16	0.58	0.11

Abbildung 4.3: Vergleich der Ausführungszeiten (in Sekunden) von XPath-Anfragen bei eXist und Xindice

4.4 Sicherheitsaspekte und Performanz

Bisher wurden innerhalb dieses Kapitels die grundlegenden Anforderungen an den Prototypen besprochen, Möglichkeiten für die Applikationsarchitektur und Datenhaltung vorgestellt, und jeweils Entscheidungen für die Umsetzung auf der prototypischen Plattform getroffen. Dieses Kapitel befasst sich nun mit möglichen Sicherheitsrisiken und der Performanz auf Basis der zuvor getroffenen Entscheidungen. Zunächst werden diese in der Abbildung 4.4 zusammenhängend dargestellt.

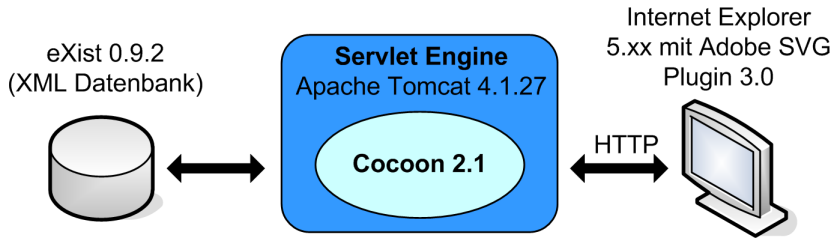

Abbildung 4.4: Überblick des Prototypkonzepts

Der Web-Browser Internet Explorer ab der Version 5.xx, sowie das Adobe SVG Plugin 3.0 stellen den Client dar, der auf die Servlet Engine Apache Tomcat zugreift. Diese enthält das Open Source XML Publishing Framework Apache Cocoon 2.1 in Form eines Servlets. Cocoon verwaltet HTTP-Anfragen des Clients und übernimmt die Durchführung der Transformationen von XML nach SVG oder HTML mittels XSLT. Die XML-Bordnetzdaten, die den Transformationen zugrunde liegen, werden in der XML-Datenbank eXist 0.9.2 gespeichert.

4.4.1 Mögliche Sicherheitsprobleme

Innerhalb dieser Zusammenstellung können Probleme auftreten, die die Sicherheit der Applikation betreffen. Folgende Ausprägungen sind denkbar:[31]

1. Die Daten vom Client zum Server werden in Klartext übertragen. Der Übertragungsweg ist nicht gesichert und die Daten sind einsehbar. Somit können Eingabedaten, Pfade und Betriebssystembefehle manipuliert werden.

[31] vgl. [SCHM2003b] Schmiderer, Johannes: *Webapplikationen mit Unterstützung von XML*, Kapitel 3.5 Sicherheitsrisiken bei Webapplikationen, URL: `http://www.schmiderer.cc/xmlwebapp/chapter_3_5.php`, Datum des Zugriffs: 12.11.2003.

2. Die beteiligten Client-Web-Browser sind nicht auf dem gleichen Stand bezüglich der Aktualität von Sicherheitsupdates.

3. Durch fehlende Benutzerverwaltung ist der Zugriff auf die Daten nicht beschränkt und somit auch Unbefugten gestattet.

4. Alte Demos, oder Testkonten auf der Datenbank oder dem Servlet können ebenfalls Unbefugten freien Zutritt verschaffen.

5. Kommentare im generierten und statischen HTML- oder SVG-Code können Implementierungsrückschlüsse geben.

Der unter 1. angesprochene Übertragungsweg kann durch SSL (Secure Socket Layer)[32] verschlüsselt und somit gegen den Zugriff Dritter gesichert werden. Regelmäßige Aktualisierungen der Systemsoftware bewirken ebenfalls einen Zugriffschutz gegen Unbefugte. Zusätzlich kann durch die Einführung eines Benutzer- und Rollenkonzepts mit entsprechender Rechtevergabe der Anwenderkreis ein- und beschränkt werden. Unbefugten wird es dadurch erschwert, auf Daten zuzugreifen, die nur für bestimmte Nutzerkreise vorgesehen sind. Das Framework Cocoon bietet die Funktionalität des Loggens von Zugriffen und Vorgängen. Dadurch können Fehlfunktionen oder fehlerhafte Vorgänge erkannt und entsprechend korrigiert werden. Innerhalb eines Firmennetzwerkes können die unter 3. angesprochenen notwendigen Sicherheitsupdates von Web-Browsern beispielsweise einheitlich über das Netzwerk realisiert werden. Dadurch wird gewährleistet, dass eine gemeinsame Basis bezüglich der Web-Browser existiert.

Beachtung der Sicherheitsaspekte bei der Umsetzung des Prototyps

Innerhalb der Prototypumsetzung wird das Logging des XML Frameworks Cocoon für die Entwicklung eingesetzt, wodurch fehlerhafte Vorgänge erkannt und behoben werden können. Trotz der Möglichkeit, die Daten nur bestimmten Anwendern zugänglich zu machen, wird kein Benutzer- und Rollenkonzept eingeführt. Wie in Kapitel 4.1 erwähnt, ist dies für den Prototypen nicht vorgesehen, bei einer späteren Umsetzung in einem größeren Rahmen jedoch wünschens- und empfehlenswert. Für die Übertragung der Daten vom Client zum Server wird des Weiteren keine SSL-Verschlüsselung vorgenommen, wobei jedoch alte Demo- oder Testkonten zum Schutz gegen unbefugte Dritte entfernt werden. Kommentare in HTML

[32] Secure Socket Layer (SSL)

oder SVG, durch die Rückschlüsse auf Implementierungen gezogen werden könnten, werden vermieden. Lediglich die Transformationsdateien werden Kommentare bezüglich der zu generierenden SVG- oder HTML-Dateien enthalten.

4.4.2 Performanz

Neben dem Thema der Sicherheit einer Web-Anwendung, spielt auch die Performanz eine wichtige Rolle. Unter Umständen kann eine Applikation die Anfragen von vielen verschiedenen Nutzern funktional richtig bearbeiten, jedoch bringt dies keinen Mehrwert, wenn die Antworten zeitlich stark verzögert beim Nutzer eintreffen. Es besteht somit eine Abhängigkeit von einem „positiven Besuchererlebnis“. Dieses definiert sich nach [DIA2004][33] über eine Antwortzeit, die weniger als 8 Sekunden beträgt und über die Verfügbarkeit von Web-Services. Um dieses so genannte „positive Besucherlebnis“ zu gewährleisten, müssen unterschiedliche Faktoren bedacht werden, die die Korrektheit von Mehrbenutzerzugriffen, die Stabilität, Verfügbarkeit und Performanz einer Anwendung beeinflussen. Diese basieren auf der komplexen Anordnung interagierender Technologien und der verteilten Struktur von J2EE-Anwendungen. Beispielhafte Faktoren sind das Datenbankdesign, die Architektur der Applikation, die Konfigurationseinstellungen der Infrastruktur und physische Gerätebeschränkungen. Nach [HOEF2001][34] sollten neben Funktionstests auch so genannte Lasttests durchgeführt werden. Diese geben unter anderem Aufschluss über Performanzengpässe, die somit frühzeitig entdeckt und behoben werden können. Gegebenfalls können Mechanismen wie Spiegelungen (Mirroring)[35], Caching[36] oder Content Delivery, eine Verbindung von Spiegelung und Caching, für die Lastverteilung eingesetzt werden.

[33] [DIA2004] Deutsche Informatik Akademie: *Web-Performance: Metriken, Modelle und Methoden für Planung und Test von E-Business-Websites* , URL: `http://www.dia-bonn.de/web2_2004/web.html`, Datum des Zugriffs: 05.04.2004.

[34] [HOEF2001] Höfling, Jürgen: *Performance ist geschäftskritisch*, in: Informationweek, Ausgabe 8, 5. April 2001, URL: `http://www.informationweek.de/index.php3?/channels/channel20/010834.htm`, Datum des Zugriffs: 02.04.2004.

[35] Die Spiegelung ist ein aufwändiger Push-Mechanismus. Es muss immer der gesamte Server repliziert werden, auch wenn nur auf einen kleinen Teil des Server-Inhalts zugegriffen werden soll.

[36] Durch Caching wird eine Kopie der Seiten, die man im Internet/Intranet durchblättert hat, auf der lokalen Festplatte zurückbehalten, damit man im Falle eines Wiederabrufens nicht abwarten muss, bis die Informationen wieder neu geladen werden.

Prototypspezifische Erwartungen

Im Folgenden werden die Anforderungen an den Prototypen bezüglich der Performanz aufgelistet und daraufhin Umsetzungsmöglichkeiten beschrieben:

- Die Umsetzung des Prototyps soll einen Mehrbenutzerzugriff von ca. 40 Personen ermöglichen.
- Die Ladezeiten statischer HTML-Seiten sollen kleiner als 4 Sekunden sein.
- Dynamische SVG- oder HTML-Seiten sollen je nach Komplexität der zugrunde liegenden Transformationen eine akzeptable Ladezeit beanspruchen. Beispielsweise sollte die Transformation einer 500 KB großen SVG-Datei nicht länger als 20 Sekunden dauern.
- Bei den Transformationen sollen lediglich die dafür notwendige Ressourcen verwendet werden, um die Bearbeitungszeiten so gering wie möglich zu halten. Dies bedeutet z.B. so wenig Datenbankzugriffe wie möglich durchzuführen und die Zahl der Zwischentransformationen auf ein Minimum zu beschränken.

Um diesen Anforderungen gerecht zu werden, können das Design und die Konfiguration der Datenbank, die Applikationsarchitektur, sowie die Konfigurationseinstellungen der Infrastruktur beispielsweise wie folgt bearbeitet werden: Die XML Datenbank eXist kann durch eine Konfigurationsdatei an die speziellen Anforderungen des Prototyps angepasst werden. Innerhalb des XML Frameworks Cocoon stehen so genannte Caching- und Pooling-Mechanismen zur Verfügung. Durch das Cachen generierter HTML- oder SVG-Seiten können die Antwortzeiten bei nochmaligem Aufruf dieser Seiten verringert werden. Unter einem Pooling-Mechanismus ist die effiziente Verwendung von Komponenten in Cocoon zu verstehen. Dabei wird in Cocoon eine bestimmte Anzahl von Komponenten für die Anfragebearbeitung angelegt und nach Ende der Bearbeitung nicht zerstört. Somit kann bei einer erneuten Anfrage auf einen vorhandenen Pool von Komponenten zugegriffen werden, wodurch die Zeit für die Erzeugung und Zerstörung der Komponenten entfällt und Anfragen schneller bearbeitet werden können.
Innerhalb der Umsetzung der prototypischen Anwendung sollen diese Konfigurationseinstellungen erst dann vorgenommen werden, wenn sich abzeichnet, dass durch die Default-Einstellungen der beteiligten Produkte die zuvor genannten Anforderungen nicht erfüllt werden können und Performanzengpässe auftreten.

4.5 Verfügbarkeit

Unter der Verfügbarkeit einer Anwendung ist die Anforderung zu verstehen, 24 Stunden am Tag und 7 Tage in der Woche auf eine Anwendung zugreifen zu können. Eine hohe Verfügbarkeit kann durch die Benutzung von „Clustering“ gewährleistet werden. Dabei kann es sich um mehrere Computer handeln, die zur Verbesserung der Zuverlässigkeit, Verfügbarkeit, Servicefreundlichkeit und Leistung mittels Load-Balancing [37] miteinander verbunden sind. Solche Cluster nutzen gemeinsame Ressourcen und eine spezielle Software, mit der die Aktivitäten der einzelnen Komponenten koordiniert werden. Es können aber auch Cluster für die Ausfall-Überwachung eingesetzt werden. Dabei besteht der Cluster aus zwei Knoten, wobei ein passiver Knoten die Funktion des aktiven Knotes überwacht und im Fehlerfall seine Position einnimmt.
Bei der prototypischen Umsetzung wird der Aspekt der Verfügbarkeit weiterhin nicht genauer betrachtet. Zwar ist eine Verfügbarkeit des Prototyps für 24 Stunden am Tag und 7 Tage die Woche vorgesehen, es werden jedoch keine speziellen Vorkehrungen getroffen, um dies zu gewährleisten.

[37] Load-Balancing wird die Lastverteilung für Übertragungsstrecken und Hardware-Einrichtungen genannt. Neben der Maximierung des Durchsatzes bei gleichzeitiger Minimierung der Server-Last, dient das Load Balancing der standortübergreifenden Fehlertoleranz, der einfachen Konfigurierbarkeit der Server-Cluster und der Skalierbarkeit der verfügbaren Ressourcen, sowie der Verbesserung der Server-Server-Kommunikation.

5 Umsetzung des Prototyps

Im folgenden Kapitel wird die Realisierung des Prototyps anhand des zuvor erstellten Konzeptes beschrieben. Ziel ist es, den der Diplomarbeit zugrunde liegenden Fragestellungen aus Kapitel 2 nachzugehen, sowie die Funktionsweise und die verwendeten Technologien des Prototyps zu erläutern. Für die Entwicklung des Prototyps wurde ein IBM Thinkpad A31p mit einem 2 GHz Pentium 4 – Prozessor und 1 GB RAM zur Verfügung gestellt. Weiterhin wurde als Betriebsystem Microsoft Windows XP Professional mit Service Pack 1 verwendet.

5.1 Funktionalität und Abgrenzungen des Prototyps

Die Funktionalitäten des zu erstellenden Prototyps werden in drei Bereiche gegliedert. Der erste Bereich sieht die Generierung verschiedener SVG-Sichten auf Bordnetzschaltpläne vor. Dabei werden die Gesamt- und Einzelansichten von Schaltplänen in SVG dargestellt. Gesamtansichten zeigen einen vollständigen Schaltplan, während Einzelansichten ausgewählte Teile eines Schaltplanes darstellen. Eine weitere Funktionalität besteht in der Analyse von Bordnetzschaltplänen, wie z.B. der Suche nach allen verwendeten Querschnitten in einem bestimmten Schaltplan. Diese Ergebnisse werden in HTML oder SVG dargestellt. Berichte im HTML-Format zu erzeugen, ist die dritte Funktionalität des Prototyps. Es soll z.B. eine Liste generiert werden, die alle Verbindungen/Netze eines Schaltplanes enthält und für jedes Netz zusätzliche Informationen zur Verfügung stellt.
Im vorhergehenden Kapitel wurde das XML-Publishing-Framework Cocoon für die Umsetzung des Prototyps und die Abarbeitung der Transformationen gewählt, sowie die XML-Datenbank eXist für die Speicherung der Bordnetzdaten. Abbildung 5.1 zeigt, wie über Formularlisten einer Web-Seite Anfragen für die oben genannten Transformationen an die Servlet Engine Apache Tomcat geschickt werden. Die Engine beinhaltet das Framework Cocoon, das die Anfragen entgegen nimmt und bearbeitet. Dabei werden Bordnetzdaten aus der XML-Datenbank eXist verwendet, die zuvor über die Export-Schnittstelle des Schaltplaneditors

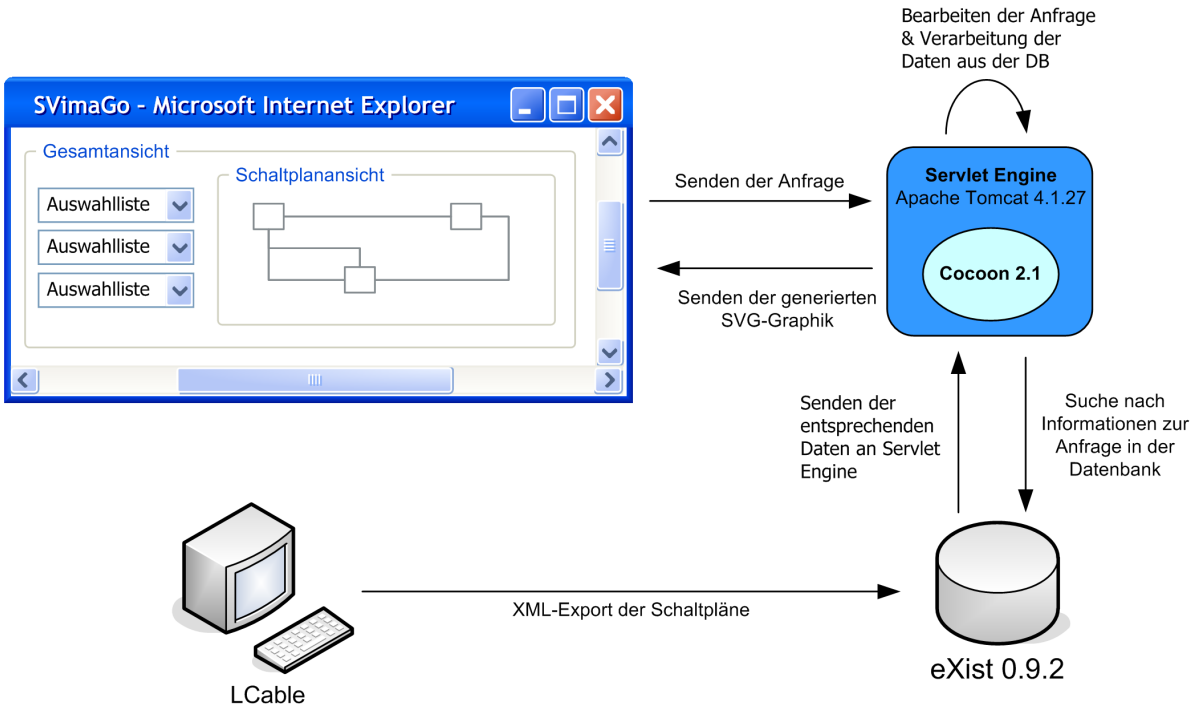

Abbildung 5.1: Übersicht der verwendeten Technologien

LCable generiert und in die Datenbank eingepflegt wurden.
Bedingt durch den begrenzten Bearbeitungszeitraum der Diplomarbeit ist es notwendig, Beschränkungen bei der Umsetzung des Prototyps vorzunehmen. Im Rahmen der Diplomarbeit werden die Transformationen zunächst mit den Bordnetzdaten aus LCable durchgeführt. Da die Darstellung dreidimensionaler Daten mithilfe des Vektorformates SVG aufwändig ist und im Schnittstellenmodell 3D-Angaben bisher nicht vorgesehen sind, werden 3D-Koordinaten aus CATIA nicht berücksichtigt. Bei Volkswagen wird, wie im Kapitel 1.2 erläutert, in systemorientierte und kabelorientierte Schaltpläne unterschieden. Im Prototyp wird bei der Auswahl der Schaltpläne für die Transformationen keine Unterscheidung vorgenommen. Je nach Datenbestand wird entweder ein KAB oder SYS für die Transformationen der Schaltpläne nach SVG oder HTML verwendet. Eine Benutzerauthentisierung wird an dieser Stelle nicht implementiert, wie zuvor in Kapitel 4.1 festgelegt. Ähnlich verhält es sich mit einem Versionierungssystem der in der Datenbank eingepflegten Bordnetzdaten. Werden Veränderungen an einem Schaltplan vorgenommen, muss ein erneuter Export in LCable gestartet werden. Die somit erzeugten XML-Bordnetzdaten werden in die XML-Datenbank eingepflegt, wobei die vorherige Version überschrieben wird. Letztlich werden die möglichen Ausgabeformate

auf HTML und SVG beschränkt, sowie die dynamische Generierung der Schaltpläne nach SVG auf die Gesamtansicht eines Schaltplanes.

5.2 Schnittstellenmodell für die Abbildung von Schaltplandaten in XML

Wie im vorherigen Kapitel erwähnt, basieren die Funktionalitäten des Prototyps auf Bordnetz-Schaltplan-Daten im XML-Format. Während eines Exportvorganges des Schaltplaneditors LCable werden alle in LCable vorhandenen Elemente, sowie die komplexe Bordnetz-Schaltplan-Struktur in XML abgebildet. Diesem Vorgang liegt ein Schnittstellenmodell zugrunde, anhand dessen verschiedene XML-Dokumente pro Schaltplan erzeugt werden. Für dieses Schnittstellenmodell existierten zum Zeitpunkt der Erstellung der Diplomarbeit keine Modellierungen in UML oder XML Schema Definitionen. Für weitere Entwicklungen ist es jedoch sinnvoll, an dieser Stelle auf XML Schemata oder UML-Modellierungen zurückgreifen zu können, die die Struktur der Bordnetz-Schaltplan-Daten genau beschreiben.

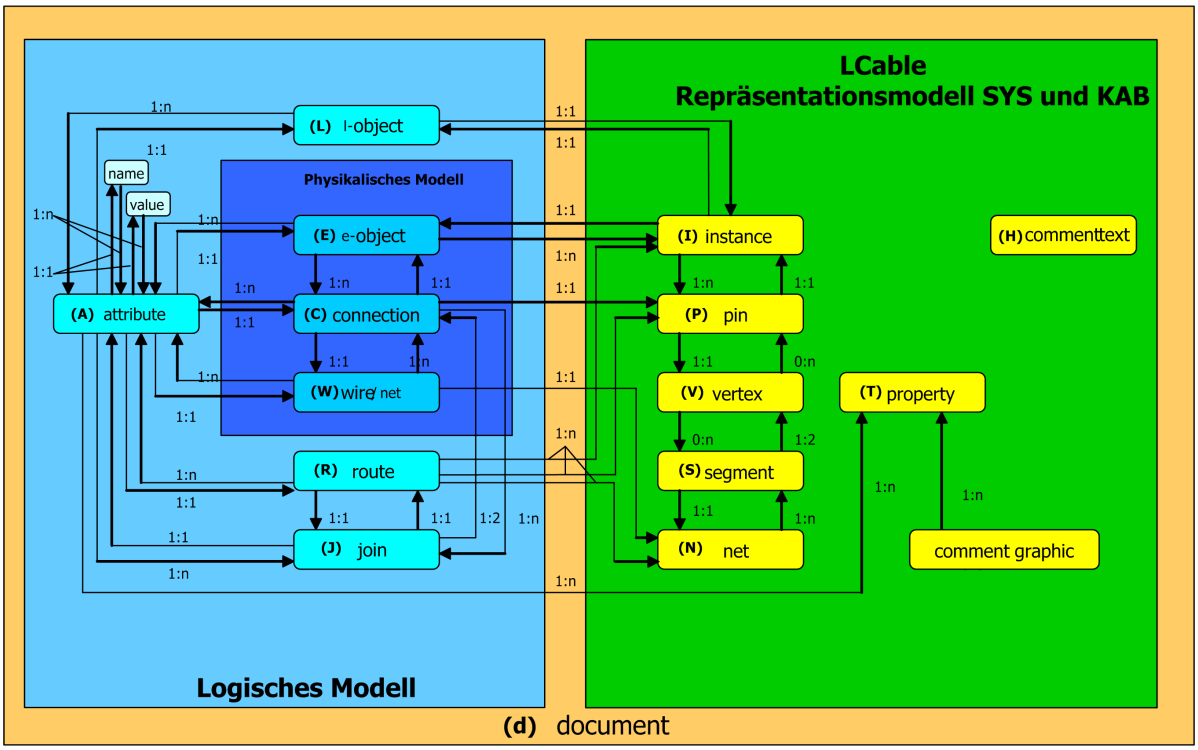

Abbildung 5.2: Objektsicht des Bordnetz-Schnittstellenmodells

Zur Erstellung des Prototyps stand die in Abbildung 5.2 dargestellte Übersicht zur Verfügung, die die Objekte des LCable-Bordnetz-Schnittstellenmodells und ihre Beziehungen untereinander zeigt. Dieses Modell war Grundlage folgender Betrachtungen.

5.2.1 Beschreibung des Schnittstellenmodells

Das LCable-Bordnetz-Schnittstellenmodell (siehe Abbildung 5.2) ist in drei Bereiche aufgeteilt – in das Repräsentationsmodell, das logische Modell und das physikalische Modell, dargestellt in Abbildung 5.2. Die verschiedenen Elemente eines Bordnetzes-Schaltplanes sind einem dieser Bereiche zugeordnet und in Abbildung 5.3 zum Teil beispielhaft veranschaulicht. Der

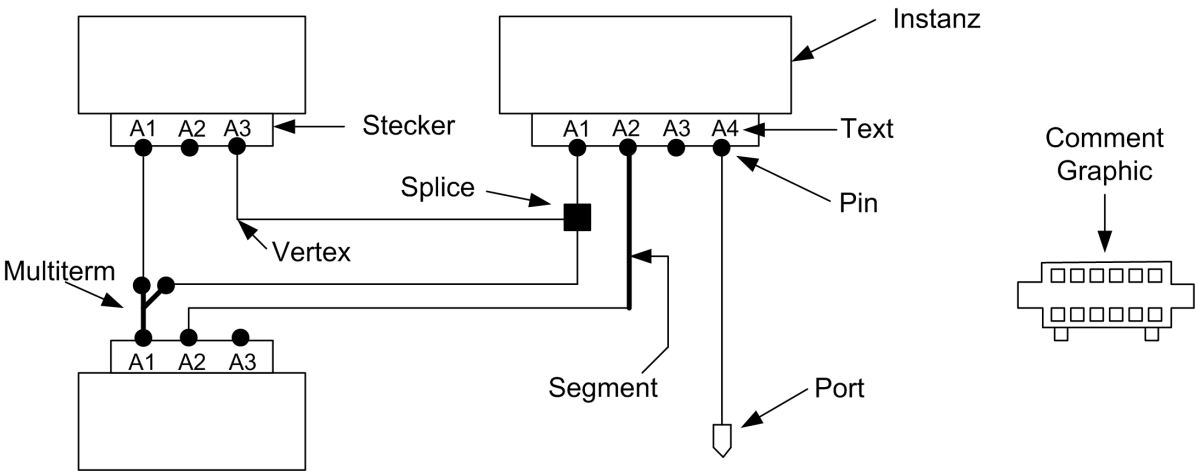

Abbildung 5.3: Beispielhafte Übersicht der Elemente eines Bordnetz-Schaltplanes

Bereich des Repräsentationsmodelles beinhaltet alle Informationen der Bordnetz-Schaltplan-Elemente, die tatsächlich auf einem Schaltplan dargestellt werden können:

Instanz Eine Instanz ist die graphische Repräsentation eines elektrischen Objektes. Dabei handelt es sich um Bauteile wie Schalter oder Steuergeräte.

Pin Ein Pin ist die Anschlussmöglichkeit einer Komponente.

Vertex Ein Vertex ist der Aufhänge- oder Eckpunkt eines Netzes.

Segment Ein Segment ist ein Teilabschnitt eines Netzes.

Net Ein Net ist die graphische Repräsentation der Verbindungen verschiedener Anschlüsse.

Text Der Text steht für die Properties in der graphischen Repräsentation, diese Texte gehören zu den Objekten.

Comment Graphic Eine Kommentargraphik ist kein elektrisches Objekt, trägt keine Pins und besitzt nur informativen Charakter.

Comment Text Kommentartexte sind losgelöst, sie haben keinen Besitzer.

Eine Instanz kann dabei einen bis unendlich viele Pins besitzen, wobei ein Pin nur einer Instanz zugeordnet sein darf. Anhand eines Aufhängepunktes wird ein Pin im Schaltplan platziert. Ein Segment hingegen setzt sich aus einem, maximal zwei Aufhängepunkten zusammen, wobei ein Aufhängepunkt keinem oder unendlich vielen Segmenten zugeordnet sein darf. Ein Segment darf nur einem Netz angehören und ein Netz kann sich aus einem bis unendlich vielen Segmenten zusammensetzen. Eine Kommentargraphik kann weiterhin eine bis unendlich viele Texte enthalten.
Die Zusammenhänge und Struktur eines Bordnetz-Schaltplanes sind an andere Objekte gebunden und im logischen Modell des LCable-Bordnetz-Schnittstellenmodells abgebildet.

Attribute Ein Attribut ist der Überbegriff der Eigenschaften und Merkmale eines E-Objektes, einer Connection, eines Wires, einer Route oder eines Joins.

Name Name bezeichnet den Namen eines Attributes.

Value Value bezeichnet den Wert eines Attributes.

L-Object Ein L-Object bezeichnet ein logisches Objekt eines Schaltplanes, wie z.B. einen Multiterm[1] oder einen Port.

Route Eine Route ist der Weg einer Verbindung, repräsentiert durch 1 bis n Leitungen.

Join Ein Join ist die Verbindung zweier Pins, wie z.B. in der Abbildung 5.3 vom Pin A2 zum Pin A2.

Im physikalischen Modell werden die Elemente abgebildet, die tatsächlich in einem Leitungsstrang existieren.

E-Object Ein E-Objekt ist ein elektrisches Objekt wie z.B. ein Aktor, ein Stecker oder ein Port.

[1] Ein Multiterm wird bei einem Mehrfachanschlag von Leitungen auf einen Pin verwendet.

Connection Eine Connection ist ein elektrischer Anschluss, die graphische Repräsentation einer Connection ist der Pin.

Wire/Net Ein Wire bezeichnet eine Leitung.

Ein E-Objekt kann ein bis unendlich viele Attribute besitzen, ebenso wie eine Connection, ein Wire, eine Route oder ein Join. Ein Attribut besitzt des Weiteren einen Namen und einen Wert, wohingegen ein Name und Wert einem oder mehreren Attributen zugeordnet sein kann. Ein elektrisches Objekte kann auf einem Schaltplan durch eine oder mehrere Instanzen dargestellt werden. Dies wird dann notwendig, wenn ein E-Objekt gebrochen, d.h. auf zwei Schaltplänen dargestellt werden muss. In diesem Fall erscheint für ein E-Objekt mit 50 Pins auf einem Schaltplan eine Instanz mit den ersten 25 Pins und auf einem zweiten Schaltplan die zweite Instanz mit den restlichen 25 Pins. Weiterhin kann ein E-Objekt mehr als eine Connection besitzen. Eine solche kann auf dem Schaltplan nur durch einen Pin dargestellt werden. Im vorherigen Beispiel wird dadurch die Darstellung aller 50 Pins auf dem ersten und zweiten Plan vermieden. Weitere Abhängigkeiten der Objekte des LCable-Schnittstellenmodells können der Abbildung 5.2 entnommen werden.
Informationen des Schaltplandokumentes selbst, wie beispielsweise das Erstelldatum, werden innerhalb des document-Elementes gespeichert. Vorteilhaft an dieser Aufteilung ist, dass die Daten, die beispielsweise für die Darstellung eines Schaltplanes notwendig sind, schnell erreicht werden können. Je nach gefordertem Ergebnis ist es möglich, über verschiedene Startpunkte in das Modell einzusteigen und die notwendigen Informationen zu sammeln.

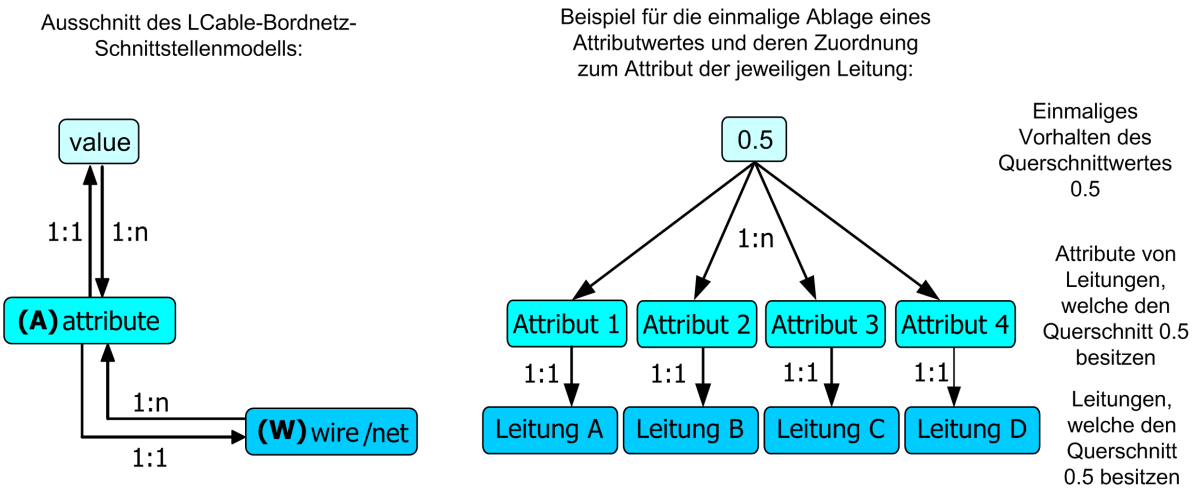

Abbildung 5.4: Beispiel für die Referenzierung innerhalb des LCable-Schnittstellenmodells

Weiterhin ist dieses Modell darauf ausgelegt, eine redundante Datenhaltung zu vermeiden. Mit Hilfe von Referenzen auf die jeweiligen Klassen müssen die Informationen lediglich einmal an einer Stelle existieren, auf die dann mehrfach verwiesen werden kann. Angenommen, mehrere Leitungen eines Schaltplanes besitzen denselben Querschnitt. Der Wert des Querschnittes wird nach dem obigen Modell ein einziges Mal in einer Instanz der Value-Klasse gehalten (siehe Abbildung 5.4). Diese verweist auf mehrere Attributinstanzen, die wiederum auf all jene Leitungen referenzieren, die den besagten Querschnitt besitzen. Somit werden lediglich Referenzen auf bestimmte, häufig auftretende Merkmale gehalten, anstatt diese explizit jedes Mal aufzuführen. Des Weiteren wurde eine geringe Typisierung der Klassen eines Bordnetzes durchgeführt. Dies ist daran zu erkennen, dass keine Schaltplan-spezifischen Klassen wie Stecker oder Massepunkte definiert wurden. Die Unterscheidung, ob es sich bei einer elektrischen Komponente um einen Stecker oder Massepunkt handelt, wird hierbei ebenfalls über die Attribute durchgeführt.

5.2.2 Definition der verschiedenen Dokumente des Schnittstellenmodells

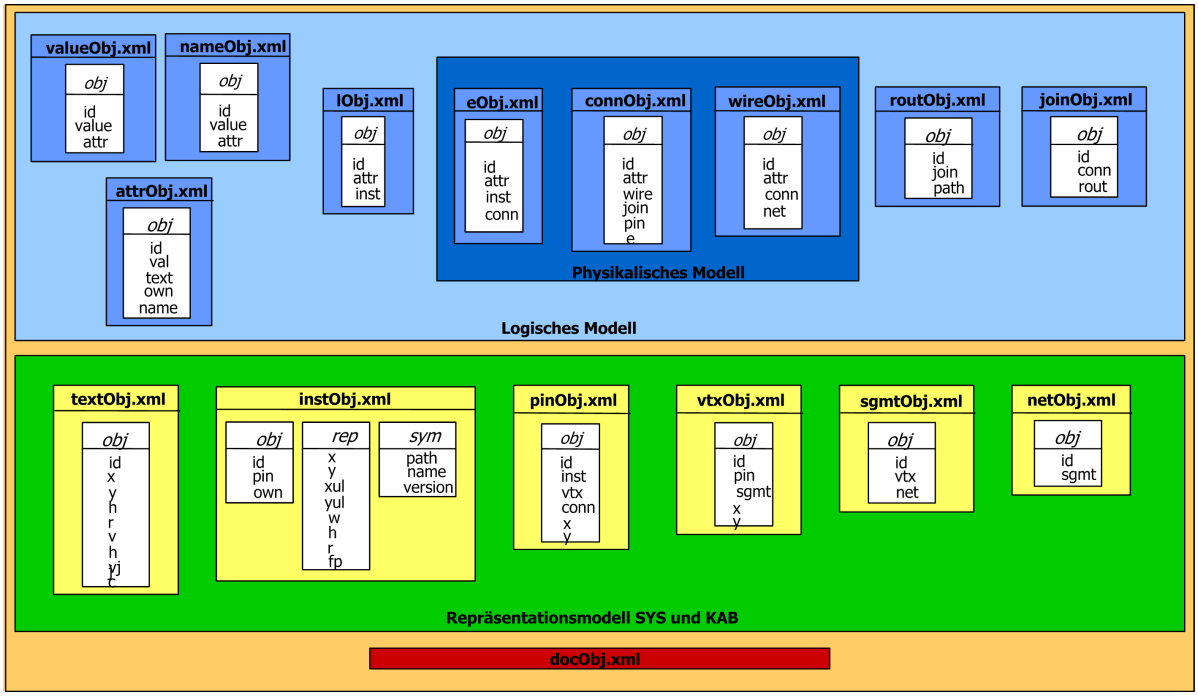

Abbildung 5.5: Dokumentsicht des Bordnetz-Schnittstellenmodells

Wird der Export eines Schaltplanes in LCable gestartet, werden anhand des zuvor beschriebenen Modells, mehrere XML-Dokumente generiert. Es entsteht pro Klasse des Schnittstellenmodells eine XML-Datei mit mehreren Instanzen der Klasse, die durch jeweils ein `obj`-Element dargestellt werden. Beispielsweise enthält die Datei `pinObj.xml` die Repräsentationsdaten aller Pins eines Schaltplanes, die Datei `eObj.xml` die Informationen über alle elektrischen Objekte usw. Insgesamt entstehen bei einem LCable-Export pro Schaltplan 18 XML-Dokumente (siehe Abbildung 5.5). Dabei wird innerhalb der Instanzen eines Objektelementes durch Positionsangaben auf weitere XML-Dokumente verwiesen. Um die Dateigrößen der XML-Dokumente so gering wie möglich zu halten, wurden zur Umsetzung dieser Verweise nicht die W3C-Spezifikationen XLink oder XPointer verwendet. Für spätere Entwicklungen sollte dieser Aspekt jedoch erneut überdacht werden, da dadurch die Verweise innerhalb der XML-Dokumente ebenfalls standardisiert umgesetzt werden können.
Die Darstellung 5.6 zeigt die Datei `instObj.xml` eines Schaltplanes mit je einem elektrischen und logischen Objekt. Pro Instanz der Klasse `instance` wird innerhalb der Datei

```
<instObj>
 <obj id="I$2733815" pin="1 2 3 4 5 6 7 8" own="1 0" type="vobes_component_instance">
  <rep x="-130" y="120" xul="-170" yul="163.75" w="45" h="43.75" r="0" fp="@false"/>
  <sym path="$CLS_SYM_PATH/aktor/MO_FENST_T_RE" name="MO_FENST_T_RE" version="4"/>
 </obj>
 <obj id="I$2734226" pin="0" own="6 0" type="vobes_mergeport_out_instance">
  <rep x="-165" y="-27.5" xul="-166.25" yul="-26.875" w="2.5" h="6.88" r="270" fp="@false"/>
  <sym path="$VW_MISC_PATH/symbols/merge_out" name="merge_out" version="3"/>
 </obj>
</instObj>
```

Abbildung 5.6: Beispiel der Datei instObj.xml

`instObj.xml` ein XML-Element `obj` mit Attributen angelegt. Da der Schaltplan zwei Objekte besitzt, die durch die Klasse `instance` repräsentiert werden, enthält diese Datei zwei `obj`-Elemente. Innerhalb dieser Elemente wird durch die Attributwerte des Attributes `pin` auf Positionen von `obj`-Elementen in der Datei `pinObj.xml` dieses Schaltplanes verwiesen (siehe Abbildung 5.7). Treten mehrere Verweise innerhalb eines Attributes auf, werden diese durch ein Leerzeichen voneinander getrennt dargestellt. Da die Nummerierung der `obj`-Elemente bei 0 beginnt, verweist somit `pin="1 2 3 4 5 6 7 8"` des ersten `obj`-Elementes auf das zweite bis inklusive neunte `obj`-Element der in Abbildung 5.8 dargestellten Datei `pinObj.xml`. Somit besitzt das erste `obj`-Element der Klasse `instance` acht und das zweite nur einen Pin. Durch diese Methodik werden die Zuordnungen innerhalb eines Schaltplanes durchgeführt und die Zusammenhänge der Elemente umgesetzt. Eine vollständige Übersicht der existierenden XML-Dokumente und ihrer jeweiligen Attribute ist in Abbildung 5.5 dargestellt.

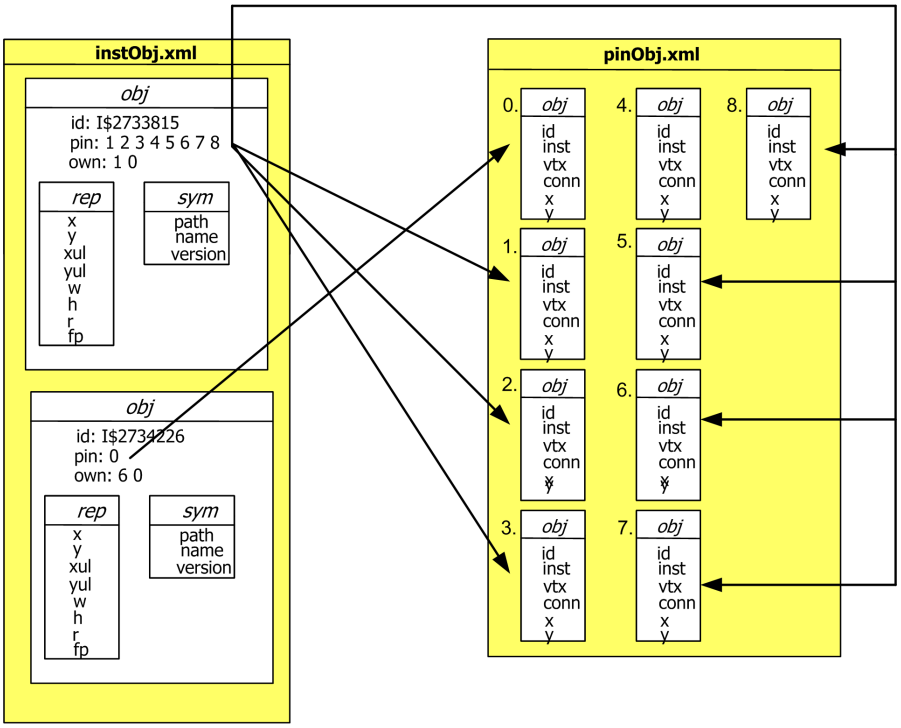

Abbildung 5.7: Beispiel für die Referenzierung in den XML-Dokumenten des LCable-Bordnetz-Schnitstellenmodells

```
<pinObj>
 <obj id="P$1152" inst="1" vtx="0" x="-165" y="-27.5"/>
 <obj id="P$924" inst="0" vtx="1" x="-130" y="120"/>
 <obj id="P$932" inst="0" vtx="2" x="-135" y="120"/>
 <obj id="P$939" inst="0" vtx="3" x="-140" y="120"/>
 <obj id="P$946" inst="0" vtx="4" x="-145" y="120"/>
 <obj id="P$953" inst="0" vtx="5" x="-150" y="120"/>
 <obj id="P$965" inst="0" vtx="6" x="-155" y="120"/>
 <obj id="P$972" inst="0" vtx="7" x="-160" y="120"/>
 <obj id="P$979" inst="0" vtx="8" x="-165" y="120"/>
</pinObj>
```

Abbildung 5.8: Beispiel der Datei pinObj.xml

5.3 Speicherung der XML-Bordnetzdaten in der XML-Datenbank eXist

Für die Speicherung der aus LCable erzeugten XML-Dokumente, wurde in Kapitel 4.3 die Entscheidung für den Einsatz der XML-Datenbank eXist getroffen. Neben einer Vorstellung der Eigenschaften dieser nativen XML-Datenbank beschreibt dieses Kapitel, wie die XML-Dokumente abgelegt werden und welche Eigenschaften der XML-Datenbank in welchem Zusammenhang für die Umsetzung des Prototyps verwendet wurden.

Allgemeines zur XML-Datenbank eXist

Die XML-Datenbank eXist wurde im Rahmen der prototypischen Applikation in der zu dem Zeitpunkt aktuellen, stabilen Version 0.9.2 eingesetzt. eXist selbst ist in Java implementiert, wodurch die Datenbank auf jeder von Java unterstützten Plattform verfügbar ist. Nach [SEFF2003][2] werden die XML-Dokumente in Collections gespeichert. Dabei liest der XML-Parser SAX die Dokumente ein und erzeugt daraufhin einen DOM-Baum. Mittels einer B+-Baumstruktur wird diesem eine ID zugewiesen. Hierauf wird dieser Baum gespeichert. In einer Collection können mehrere XML-Dokumente und weitere Collections enthalten sein, wobei diese Struktur einem Dateisystem gleicht. Durch den Einsatz von Indizes können Anfragen an bestimmte Dokumente oder den gesamten Inhalt von Collections beschleunigt werden. Im einzelnen existieren Indizes für Elemente, Namensräume und Texte. Laut [MEIE2003a][3] besitzt eXist Erweiterungen für XPath, um solche indexbasierten Anfragen zu unterstützen. Diese Erweiterungen beinhalten zusätzliche Operatoren für eine Stichwortsuche, eine Reihe von Funktionen für Abfragen von ähnlichen Suchbegriffen, einfache Platzhalter und reguläre Ausdrücke. Wie in Abbildung 5.9[4] dargestellt, steigen die Antwortzeiten ohne Verwendung der Erweiterungen mit dem Datenumfang der jeweiligen Quelle. Werden jedoch die Erweiterungen eingesetzt, ist ein deutlicher Unterschied zu erkennen.

Weiterhin kann eXist entweder als eigenständiger Server eingesetzt werden, eingebettet in eine bestehende Applikation, oder als Servlet innerhalb eines Servlet Containers. Des Weiteren unterstützt eXist grundlegende Authentisierungs- und Zugangskontrollen mit Unix-

[2] [SEFF2003] Hendrik Seffler: *XML-Datenbank Exist*, HU Berlin, URL: `http://www.informatik.hu-berlin.de/~seffler/2-ausarbeitung.pdf`, Datum des Zugriffs: 15.10.2003.

[3] [MEIE2003a] Meier, Wolfgang: *eXist – Feature Overview*, URL: `http://exist.sourceforge.net/features.html`, Datum des Zugriffs: 01.10.2003.

[4] [MEIE2003c] Meier, Wolfgang: *eXist: An Open Source Database*, Darmstadt University of Technology, URL: `http://exist-db.org/webdb.pdf`, Datum des Zugriffs: 15.10.2003.

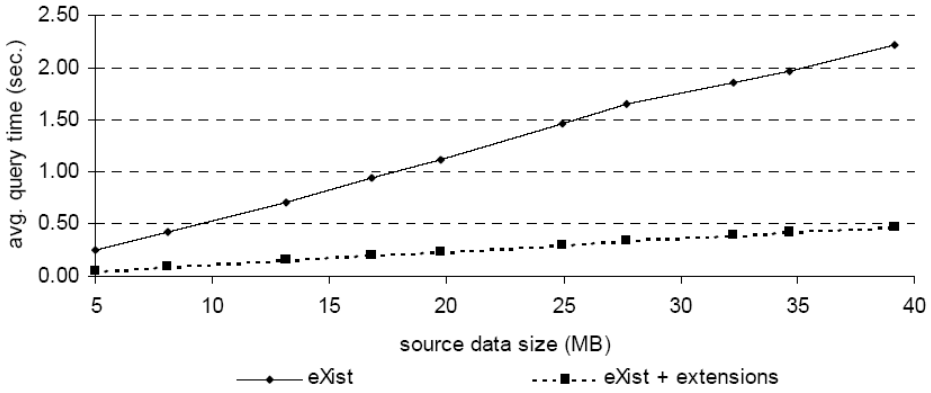

Abbildung 5.9: Durchschnittliche Antwortzeiten bezüglich der Quelldatengröße ([MEIE2003c])

ähnlichen Zugangsrechten für Benutzer und Gruppen. Weitere Informationen zu eXist sind unter `http://exist.sourceforge.net` zu finden.

Datenspeicherung mit eXist

Für die Speicherung der XML-Bordnetzdaten wurde eXist als eigenständiger Server eingesetzt. Somit konnten die Daten entweder über den mitgelieferten Java-Client oder über einen HTTP-Client in die Datenbank eingepflegt werden. Das Hinzufügen von großen XML-Dokumenten in die Datenbank auf einem entfernten Rechner über den GUI-Client ist problematisch. Denn dabei werden die Dokumente mittels einer XML-RPC Schnittstelle in einzelne Stücke aufgeteilt und über das Netzwerk versendet. Die Teilstücke werden in einem temporären Dokument auf dem Server zwischengespeichert, das anschließend in der Datenbank gespeichert wird. Dieser zeitintensive Prozess beansprucht das Speichervolumen des Servers stark. Beim Hinzufügen der XML-Bordnetzdaten traten des öfteren Out-Of-Memory-Fehler auf, die Übertragung wurde nach einer gewissen Zeit beendet, oder der GUI-Client stürzte ab. In solchen Fällen ist es demnach ratsam, einen Client zu verwenden, der eine direkte Verbindung zur Datenbank besitzt. Denkbar ist auch der Einsatz des in eXist enthaltenen HTTP-basierten Admin-Interfaces, das ähnliche Funktionalitäten bietet wie der GUI-Client und eine lokale Verbindung zur Datenbank besitzt. Mit diesem HTTP-Client konnten große Dokumente nahezu problemlos der Datenbank hinzugefügt werden, wie sich während der Entwicklung des Prototyps herausstellte.

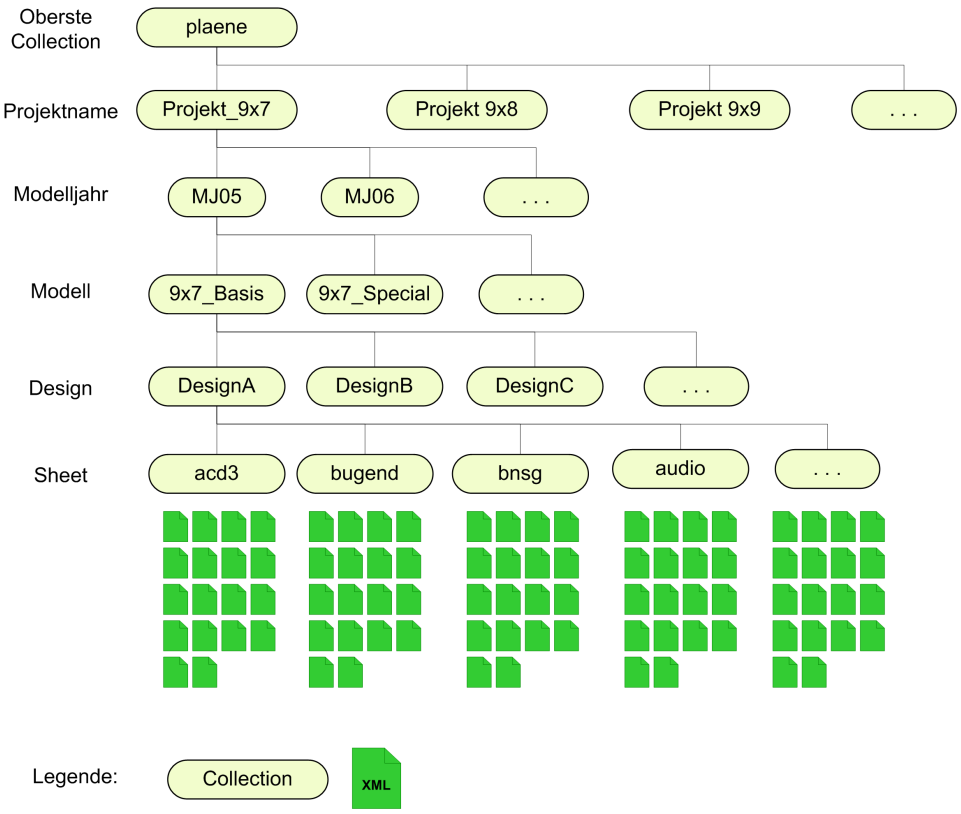

Abbildung 5.10: Struktur der Collections in eXist

Als Grundlage der durchzuführenden Transformationen wurden Testdaten der Dr. Ing. h.c. F. Porsche AG zur Verfügung gestellt. Die Struktur der Bordnetzentwicklung ist wie folgt aufgebaut. Beginnend mit dem Projektnamen wird in verschiedene Modelljahre unterschieden. Innerhalb eines Modelljahres können mehrere Modelle enthalten sein. Ein Modell kann verschiedene Designs besitzen, die sich wiederum aus mehreren Sheets zusammensetzen können. Ein Sheet stellt dabei einen Teil es gesamten Bordnetzes eines Designs dar.
Diese Zusammenhänge spiegeln sich in der Struktur der Datenbank wieder, wie in Abbildung 5.10 dargestellt ist. Daher wurde eine Collection „Projekte“angelegt, unter der alle Projekte gespeichert werden können. Innerhalb dieser Collection wird wiederum eine weitere Collection erstellt, die die verschiedenen Modelle beinhaltet, usw. Letztendlich werden unterhalb jeder „Sheet“-Collection die 18 XML-Dokumente abgelegt. Durch diese Struktur sind somit beispielsweise Abfragen über einen einzelnen Schaltplan oder auch alle in einem Projekt vorhandenen Schaltpläne möglich.

5.4 Umsetzung des Prototyps mit Hilfe des XML-Publishing-Frameworks Cocoon

Für die Generierung verschiedener Sichten auf XML-Bordnetze wird das XML-Publishing Framework Cocoon verwendet. Dieses Kapitel befasst sich zunächst mit einer allgemeinen Beschreibung dieses Frameworks. Danach folgt eine Schilderung der Entwicklung der Benutzeroberfläche, sowie des internen Aufbaus und der Anfragenbehandlung des Prototyps. Innerhalb eines weiteren Unterkapitels wird die Verwendung der XML-Datenbank eXist in Cocoon erläutert, da Cocoon auf diese zugreift um Teile der XML-Bordnetzdaten in verschiedene Ausgabeformate zu transformieren. Abschließend werden Transformationen von XML nach SVG und HTML präsentiert.

5.4.1 Allgemeines zum XML-Publishing Framework Cocoon

Architekturübersicht

Nach [ZILA2002b][5] ist Cocoon ein Framework mit einer offenen, komponentenbasierten Architektur. Grundlage dieser Architektur ist das Apache Projekt Avalon[6], ein Java-Ansatz für die komponentenorientierte Softwareentwicklung. Zugleich bedient sich Cocoon laut Langham und Ziegeler aus dem Fundus anderer Apache Projekte. Beispielsweise werden Apache Xerces als XML-Parser, Apache Xalan als XSLT-Prozessor und Apache FOP[7] zur PDF-Generierung eingesetzt. Wie fast jede Web-Anwendung ist auch Cocoon in einen festen Anfrage/Antwort-Zyklus eingebettet. Dabei wird, wie in 5.11 dargestellt, die Anfrage von der aufrufenden Umgebung, die entweder eine Servlet Engine oder ein Kommandozeilen-Interface ist, an Cocoon weitergeleitet. Als Servlet Engine kann, wie im vorliegenden Fall, Apache Tomcat verwendet werden. Die vom Client ausgehende Anfrage wird durch Cocoon bearbeitet und ein generiertes Antwortdokument an den Client zurückgeschickt. Für die Funktionsweise von Cocoon ist es unbedeutend, in welcher Umgebung Cocoon eingesetzt wird. Jede Anfrage an Cocoon liefert eine Antwort, bzw. ein Dokument. Dieses erhält immer einen eindeutigen Namen, der im Browser als Teil der URI[8] angegeben werden kann. Das zentrale

[5] [ZILA2002b] Ziegeler, Carsten und Langham, Matthew: *Cocoon – aber sicher! Eine Einführung in personalisiertes XML Publishing mit Apache Cocoon*, in: XML Magazin und Webservices, 2002, Ausgabe 3, Seite 20ff.

[6] Nähere Informationen zum Apache Avalon Project: `http://avalon.apache.org/`.

[7] Nähere Informationen zum Apache Formatting Objects Processor: `http://xml.apache.org/fop/`.

[8] URI – Uniform Resource Identifier.

Bindeglied zwischen der generierten Antwort und dem Dokumentnamen ist die Sitemap.

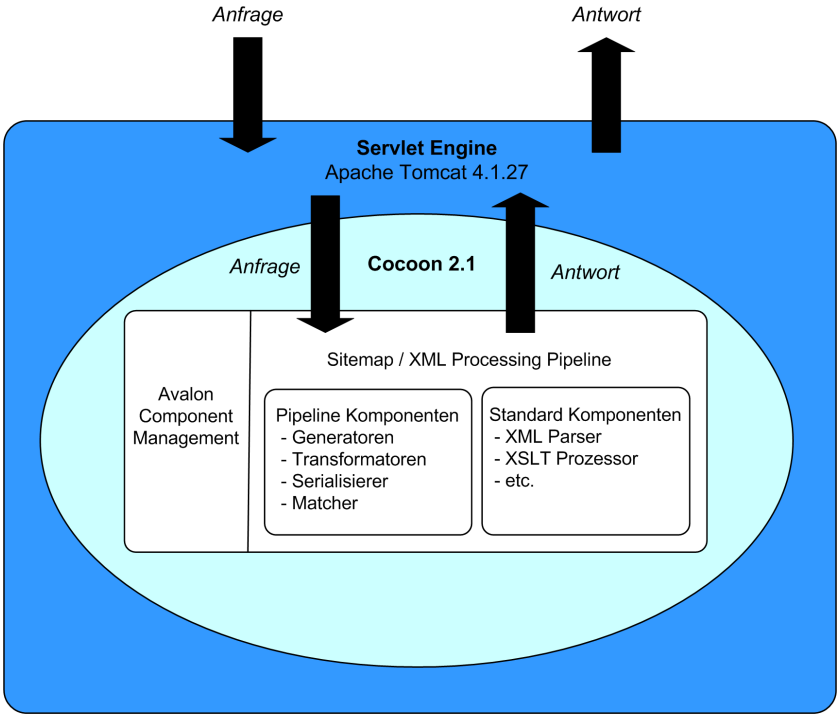

Abbildung 5.11: Architektur des Cocoon Framework

Sitemap und Pipelines

Die Sitemap ist laut [ZILA2002b] ein XML-Dokument (`sitemap.xmap`) in dem festgelegt wird, welche Arbeitsschritte beim Eintreffen einer Anfrage mit einer bestimmten URI durchgeführt werden müssen. Ebenso ist sie die zentrale Konfigurationsdatei des Frameworks Cocoon, welche die Web-Anwendung durch die Deklaration und Konfiguration von Komponenten verwaltet. Sie enthält gleichermaßen einen Satz an sofort einsetzbaren Komponenten, wobei es dennoch möglich ist, eigene Komponenten zu schreiben und einzubinden. Damit ein Ausgabedokument für den Client erzeugt werden kann, muss eine so genannte Pipeline aufgebaut werden. Die Pipeline ist ein Teil der Sitemap und kann als funktionale Einheit verstanden werden, die sich aus einer Kombination verschiedener Komponenten zusammensetzt. Abbildung 5.12 stellt exemplarisch eine solche Pipeline dar. Am Anfang steht ein Generator, der aus einem beliebigen Quelldokument eine XML-Repräsentation erzeugt, deren Struktur in Form von SAX-Events abgebildet wird. Der Vollständigkeit halber sei an dieser Stelle erwähnt, dass mit Sax – nicht wie bei DOM – der komplette Baum erzeugt werden muss,

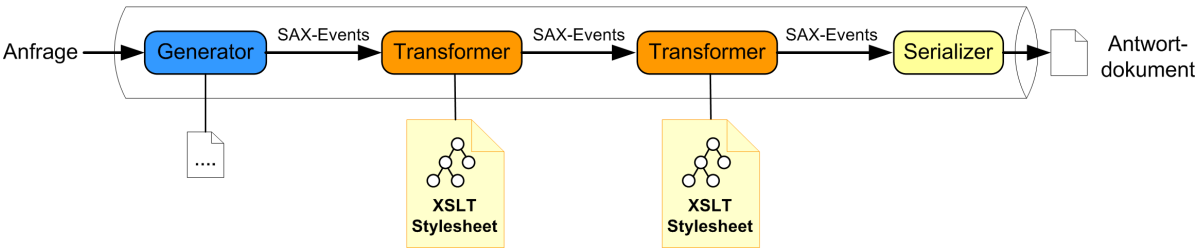

Abbildung 5.12: Cocoon XML-Pipeline

sondern die Daten seriell eingelesen und verarbeitet werden können. Dadurch wird die Verarbeitung eines XML-Dokumentes wesentlich schneller durchgeführt. Dem Generator können in der Pipeline so genannte Transformer folgen. Dies sind optionale Komponenten einer Pipeline, die den XML-Datenstrom aus der vorgelagerten Komponente erhalten. Ein Transformer kann den Datenstrom ändern, indem neue XML-Elemente eingefügt, bzw. bestehende herausgenommen oder direkt verändert werden. Mit Hilfe eines Stylesheet Transformers, der ein XSLT-Stylesheet verwendet, wird beispielsweise der XML-Datenstrom umgewandelt. Die letzte Komponente einer Pipeline ist der Serializer, der den XML-Datenstrom in das entsprechende Ausgabeformat wandelt, den Mime-Type für den Client setzt und letztendlich das eigentliche Dokument erzeugt.
Eine Sitemap kann mehrere Pipelines enthalten, in denen jeweils eine bestimmte Reihenfolge von Arbeitsschritten festgelegt ist. Beispielhaft zeigt Abbildung 5.13 eine Sitemap mit nur einer Pipeline. Zunächst muss eine Verbindung zwischen dem bei der Anfrage übergebenen Dokumentnamen und der Pipeline hergestellt werden. Dies wird über eine Cocoon-Komponente,

```
1  <map:sitemap>
2   <...>
3   <map:pipelines>
4    <map:pipeline>
5     <map:match pattern="schaltplan">
6      <map:generate src="schaltplan.xml" type="file"/>
7      <map:transform src="schaltplan-svg-style.xsl" type="xslt"/>
8      <map:serialize type="svgxml"/>
9     </map:match>
10   </map:pipeline>
11   <...>
12  </map:pipelines>
13 </map:sitemap>
```

Abbildung 5.13: Sitemap mit einer Pipeline

einen Matcher (im Beispiel durch `<map:match pattern="schaltplan">` erkennbar), erreicht. Der vom Client geschickte Dokumentname wird mit dem vorgegebenen Muster „schaltplan“ verglichen. Ist der Vergleich erfolgreich, werden alle Anweisungen ausgeführt, die in dem entsprechenden XML-Element eingeschachtelt sind. Im vorliegenden Beispiel der Abbildung 5.13 wird die Datei `schaltplan.xml` durch einen Generator geladen und mittels des Stylesheet Transformers und des XSLT-Stylesheets `schaltplan-svg-style.xsl` verändert. Mit Hilfe des SVG-Serializers wird abschließend das fertige SVG-Dokument erzeugt.

Grundlegende Konzepte

Grundlegende Konzepte von Cocoon sind die Trennung von Verwaltung (Management), Programmlogik (Logic), Inhalt (Content) und Darstellung (Style). Wie in [VOCK2003][9] aufgezeigt, werden in vielen Systemen für die Darstellung dynamischer Inhalte Programmlogik, Teile des Inhalts und die Darstellung miteinander verbunden. Dadurch sind die Kompetenzen von Progammierern und Graphikern nicht eindeutig trennbar, was sich bei großen und komplexen Web-Anwendungen negativ bemerkbar macht. Nun gibt es verschiedene Ansätze, die Kompetenzen voneinander abzugrenzen. Eine davon ist das in Kapitel 4.2.2 angesprochene Model-View-Controller Konzept. Systeme, die dieses Konzept umsetzen, sind nach [VOCK2003] nur unter Schwierigkeiten erweiterbar.
Cocoon versucht mit dem Konzept der Trennung von Verwaltung, Programmlogik, Inhalt und Darstellung dieses Problem zu lösen (siehe Abbildung 5.14). Aufgabe der Managementebene

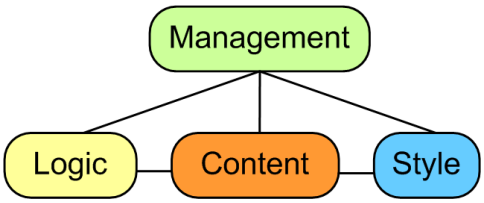

Abbildung 5.14: Konzept Cocoons für die Kompetenztrennung

ist es, die drei anderen Elemente anzuordnen und zu verknüpfen. Der Content stellt für Cocoon das Ausgangsmaterial dar, auf dessen Grundlage durch die Programmlogik Änderungen ausgeführt werden. Erst am Ende eines solchen Prozesses stehen Änderungen durch die

[9] [VOCK2003] Vockeroth, Johannes: *Das XML Publishing Framework „Apache Cocoon“*, Ausarbeitung im Rahmen des Proseminars Multimediatechnik, Fakultät Informatik TU Dresden, Januar 2003, URL: `http://www-mmt.inf.tu-dresden.de/lehre/Wintersemester_02_03/Proseminar/JohannesVockeroth/ausarbeitung.pdf`, Datum des Zugriffs: 24.10.2003.

Style-Komponente an. Eine wesentliche Konsequenz aus dieser Trennung ist die komfortable Anpassung der gesamten Anwendung an neue Ausgabeformate. Ein neues Endgerät bedeutet lediglich die Entwicklung eines entsprechenden Stylesheets und eventuell einer neuen Komponente für die Umwandlung der XML-Daten in das Datenformat des Endgerätes. Der Manager kann dann die Style-Transformierung und die entsprechende Komponente in seine Sitemap einsetzen, wobei vorhandene Styles unberührt bleiben.
Durch die extrem komponentenorientierte Architektur von Cocoon sind Programmkomponenten, die an der Ausführung des Generierungs- und Transformationsprozesses beteiligt sind, beliebig kombinier- und auswechselbar. Das Framework bietet einfache Schnittstellen, um eigene Komponenten hinzuzufügen und begegnet damit der Anforderung nach einfacher Integration bestehender Formate und Anwendungen.
Überdies lässt sich Cocoon mühelos in bestehende Anwendungen integrieren, da es unter anderem komplett auf Java und XML-Technologien wie XSLT und SAX basiert und über eine einfache Kommandozeilen-Schnittstelle aus praktisch allen Umgebungen aufgerufen werden kann.

Für nähere Informationen zu weiteren Eigenschaften von Cocoon wird auf die Web-Seite `http://cocoon.apache.org/` verwiesen.

5.4.2 Umsetzung der Benutzeroberfläche des Prototyps

Da sich, wie in Kapitel 5.1 erwähnt, die Funktionalität des Prototyps in die drei Bereiche Gesamt-/Einzelansicht, Export und Analyse aufteilt, ist diese Unterscheidung auch in der Benutzeroberfläche des Prototyps wiederzufinden (siehe Abbildung 5.15). Beim erstmaligen Aufruf des Prototyps über den Web-Browser erscheint die Startseite (Abbildung 5.16), welche auf die Seite SVimaGo[10] verweist. Dabei werden die Bereiche „SVimaGo“, „Export“ und „Analysen“ auf der Benutzeroberfläche in Form von Tabularfeldern dargestellt, wie in der Graphik 5.17 zu erkennen ist. Jeder Bereich enthält dabei ein Formular mit Auswahllisten, über welche entweder sofort eine Transformation gestartet, oder zunächst die Seite eines Teilbereiches (z.B. Gesamtansicht) geöffnet werden kann. Die Auswahllisten, die die zuvor erläuterte Projektstruktur widerspiegeln und bis auf die Startseite in jeder Seite enthalten sind,

[10] SVimaGo: Dieser Begriff setzt sich aus den Buchstaben SV und dem lateinischen Wort imago zusammen. SV steht hierbei für „Skalierbare Vektor-“ und imaGo für die „Ansicht“ oder „Bild“. SVimaGo bedeutet somit „Skalierbare Vektor-Ansichten oder -Bilder“. Durch die Großschreibung der Buchstaben S, V und G wird der Hinweis gegeben, dass SVG als Graphikformat verwendet wird.

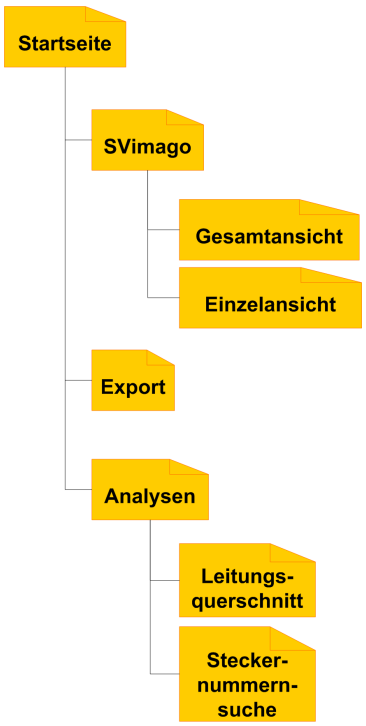

Abbildung 5.15: Übersicht der Bereiche Benutzeroberfläche

wurden mit Hilfe des Formular-Frameworks Woody[11] umgesetzt, das in Cocoon integriert ist. Abbildung 5.18 stellt den Ablauf dar, wie mit Woody die Formularseiten erzeugt wurden.

Beispielsweise trifft eine Anfrage für die Export-Seite ein, wird von der Sitemap eine Javascript-Datei aufgerufen, die in der Abbildung durch den Controller dargestellt ist. Innerhalb dieser Javascript-Datei sind die weiteren Arbeitsschritte im so genanntem Flowscript[12] definiert. Zunächst wird anhand einer Datei eine Formularinstanz erzeugt. In dieser Datei, dem Form Definition File, sind die grundlegenden Elemente und Funktionen des späteren Formulars, sowie die Vorgaben für die Pflichtfelder festgelegt. Im nächsten Schritt wird die Anfrage aus dem Formular ausgeführt und die Angaben werden durch das Flowscript auf Korrektheit überprüft. Wird eine Seite das erste Mal aufgerufen, werden die Auswahllisten zunächst mit Default-Werten gefüllt. Daraufhin ruft das Flowscript eine Pipeline der Sitemap auf, in der über den Generator vorab das Form-Template geladen wird, im vorliegenden Beispiel für die Exportseite. Dieses Template enthält den Teil der späteren HTML-Seite, in

[11] Nähere Informationen zu Woody: `http://wiki.cocoondev.org/Wiki.jsp?page=Woody`.
[12] Nähere Informationen zum Flowscript: `http://wiki.cocoondev.org/Wiki.jsp?page=Woody&version=19`.

Abbildung 5.16: Startseite der prototypischen Web-Anwendung

dem sich das Formular befinden wird. An den Stellen, an denen die Auswahllisten des Formulars stehen sollen, werden Platzhalter eingefügt. Im Woody Template Transformer werden diese Platzhalter mit den SAX-Fragmenten der widgets[13] ersetzt, die zuvor in der Formularinstanz erzeugt wurden. Das Formular ist somit fertiggestellt. Für die Darstellung im Browser wird anschließend mittels eines für den jeweiligen Bereich spezifischen XSLT-Stylesheets die HTML-Seite zusammengesetzt. Für den Export-Bereich entsteht somit eine HTML-Seite mit hervorgehobenem Tabularfeld, das neben zusätzlichen Informationen das Auswahlformular für die dynamische Generierung von Netzlisten enthält. Der HTML Serializer erzeugt aus den SAX-Events das endgültige HTML-Dokument, das an den Web-Browser geschickt wird.
Über das Formular der Exportseite kann nun ein Schaltpan ausgewählt werden, anhand dessen eine Netzliste generiert wird. Dieses Formular, das stellvertretend für die weiteren Formulare der Benutzeroberfläche steht, enthält voneinander hierarchisch abhängige Drop-Down-Listen.

[13] engl. widget = dt. Trickfenster, graphisches Interaktionsobjekt

Abbildung 5.17: Export-Seite der prototypischen Web-Anwendung

Angelehnt an das mit der Cocoon-Distribution gelieferte „carselector“-Beispiel, wird in einer XML-Datei die aktuelle Struktur der Collections der XML-Datenbank abgelegt. Folgender Ausschnitt verdeutlicht den grundlegenden Aufbau dieser Datei. Beim erstmaligen Aufruf einer Formularseite werden die Werte des Attributes `name` aller vorhandenen Projekte aus der XML-Datei ausgelesen und in der ersten Auswahlliste angezeigt, wie beispielhaft in Abbildung 5.20 zu sehen ist. Nach dem Selektieren eines Projektes werden automatisch mit Hilfe einer Pipeline die Namen aller Modelljahre des ausgewählten Projektes in die zweite Auswahlliste geschrieben. Wird daraufhin ein Modelljahr ausgewählt, erfolgt erneut eine Abfrage. Dabei wird nach den Modellen dieses Modelljahres in der XML-Datei gesucht, die daraufhin in die dritte Auswahlliste eingefügt werden, von dort wiederum ausgewählt werden können, usw. Dies wird so oft wiederholt, bis in jeder Auswahlliste ein Wert selektiert wurde. Erst dann kann das Formular abgeschickt werden. Dabei wird eine HTTP-Anfrage an das Cocoon Servlet gesendet, wobei die ausgewählten Parameter übergeben und daraufhin die Transformationen

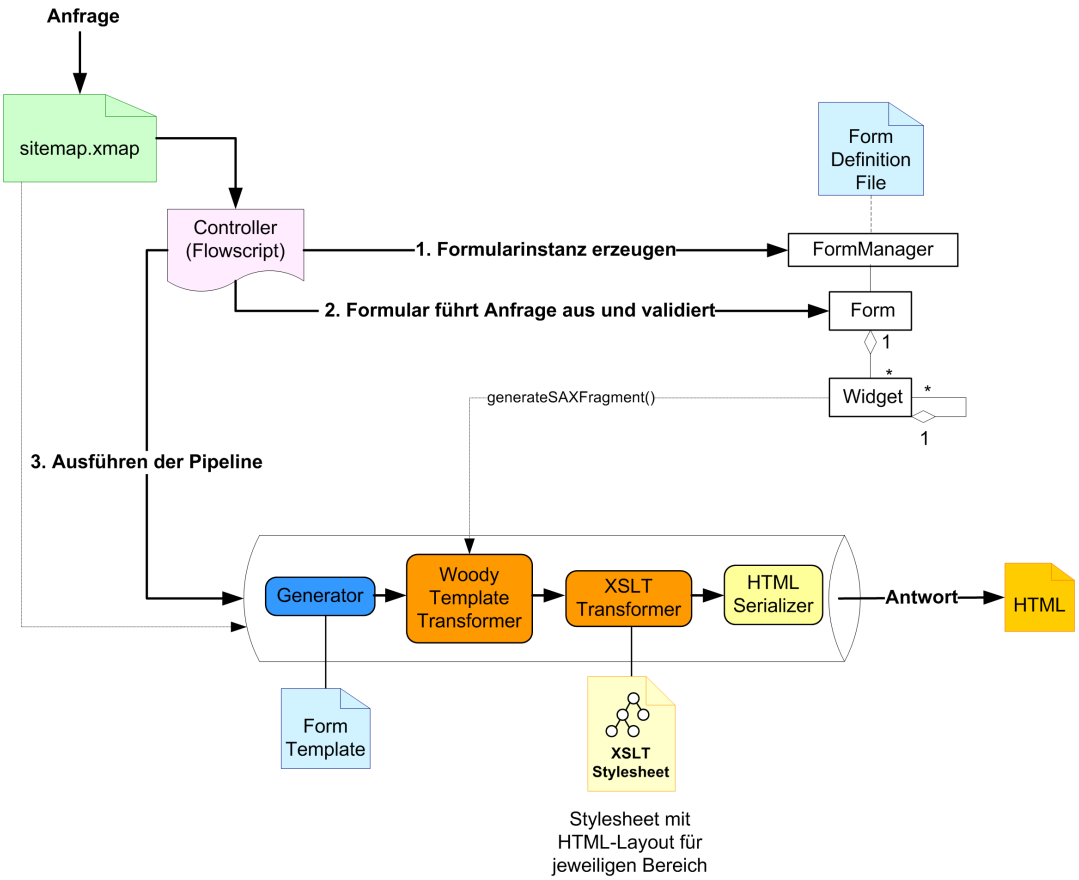

Abbildung 5.18: Ablauf der Formulargenerierung mit Woody

gestartet werden. Wie die Struktur des Prototyps bezüglich der Bearbeitung solcher Anfragen aufgebaut ist, wird nun im folgenden Kapitel näher erläutert.

```
...
<project name=" ">
 <year name=" ">
  <model name=" ">
   <design name=" ">
    <sheet name=" "/>
    <sheet name=" "/>
    <sheet name=" "/>
    <sheet name=" "/>
    ...
   </design>
   <design name=" ">
    <sheet name=" "/>
    <sheet name=" "/>
   </design>
   ...
  </model>
  ...
 </year>
</project>
<project name=" ">
 <year name=" ">
 ...
 <year/>
</project>
...
```

Abbildung 5.19: Aufbau der XML-Datei, in der die Struktur der XML-Datenbank abgelegt wird

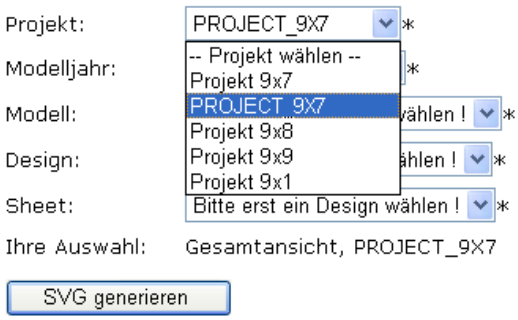

Abbildung 5.20: Auswahlliste der Benutzeroberfläche

5.4.3 Aufbau und Anfragenbehandlung des Prototyps

Wie anfänglich erwähnt, ist die Sitemap die zentrale Schaltstelle von Cocoon. Darin wird festgelegt, welche Arbeitsschritte für einen URI ausgeführt werden. Nach [ZILA2002b] empfiehlt es sich, bei größeren Web-Applikationen die Sitemap in verschiedene Sub-Sitemaps aufzuteilen, um das Editieren und Verwalten zu vereinfachen. Eine Sub-Sitemap stellt dabei eine eigenständige Sitemap dar, die in die Haupt-Sitemap integriert wird. Dies erfolgt mittels einer speziellen Anweisung in der Haupt-Sitemap, bei der ein Präfix für die Sub-Sitemap definiert wird. Anfragen, die mit einem solchen Präfix beginnen, werden für die weitere Bearbeitung an die Sub-Sitemap übergeben. Dieses Verfahren erhöht die Wartbarkeit der Sitemap und erlaubt es, die Haupt-Sitemap übersichtlicher und modularer zu gestalten. Daher wurden innerhalb der prototypischen Web-Anwendung neben zwei Haupt-Sitemaps mehrere Sub-Sitemaps erstellt. Wie die Abbildung 5.21 verdeutlicht, wurde somit jedem Funktionsbereich des Prototyps eine eigene Sitemap zugewiesen.

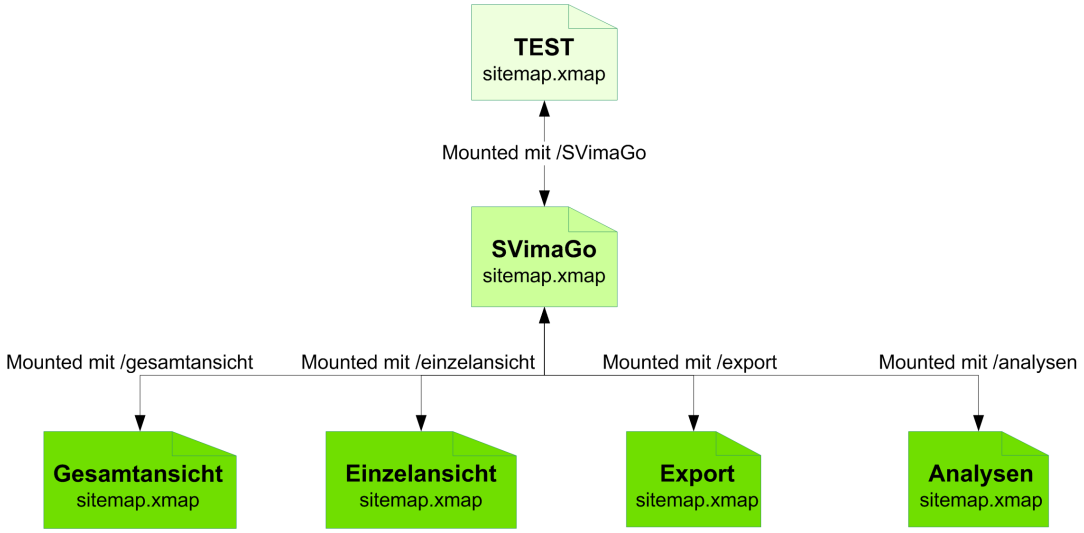

Abbildung 5.21: Sitemapübersicht des Prototyps

Die Sitemap „TEST“ besitzt hierbei lediglich deklarativen Charakter. In ihr werden die Komponenten von Cocoon definiert, die innerhalb dieses Servlets zum Einsatz kommen sollen und können. Über einen Verweis, bzw. ein Mount, wird die „SVimaGo Sitemap“ eingebunden. Anfragen, die sich hauptsächlich mit der Benutzeroberfläche des Prototyps befassen, werden in dieser Sitemap behandelt. Beispielsweise befinden sich hier die Pipelines für die in Kapitel 5.4.2 erläuterten Formulare. Die Sitemaps „Analysen“, „Einzelansicht“, „Gesamtansicht“

und „Export“ enthalten Verarbeitungs-Pipelines. Dies bedeutet, dass alle für die Transformationen benötigten Pipelines in die Sub-Sitemaps der verschiedenen Bereiche ausgelagert sind. Zum Beispiel befinden sich in der Sitemap „Gesamtansicht“ all die Pipelines, die die XML-Bordnetzdaten in eine SVG-Schaltplan-Darstellung transformieren. Ähnlich verhält es sich

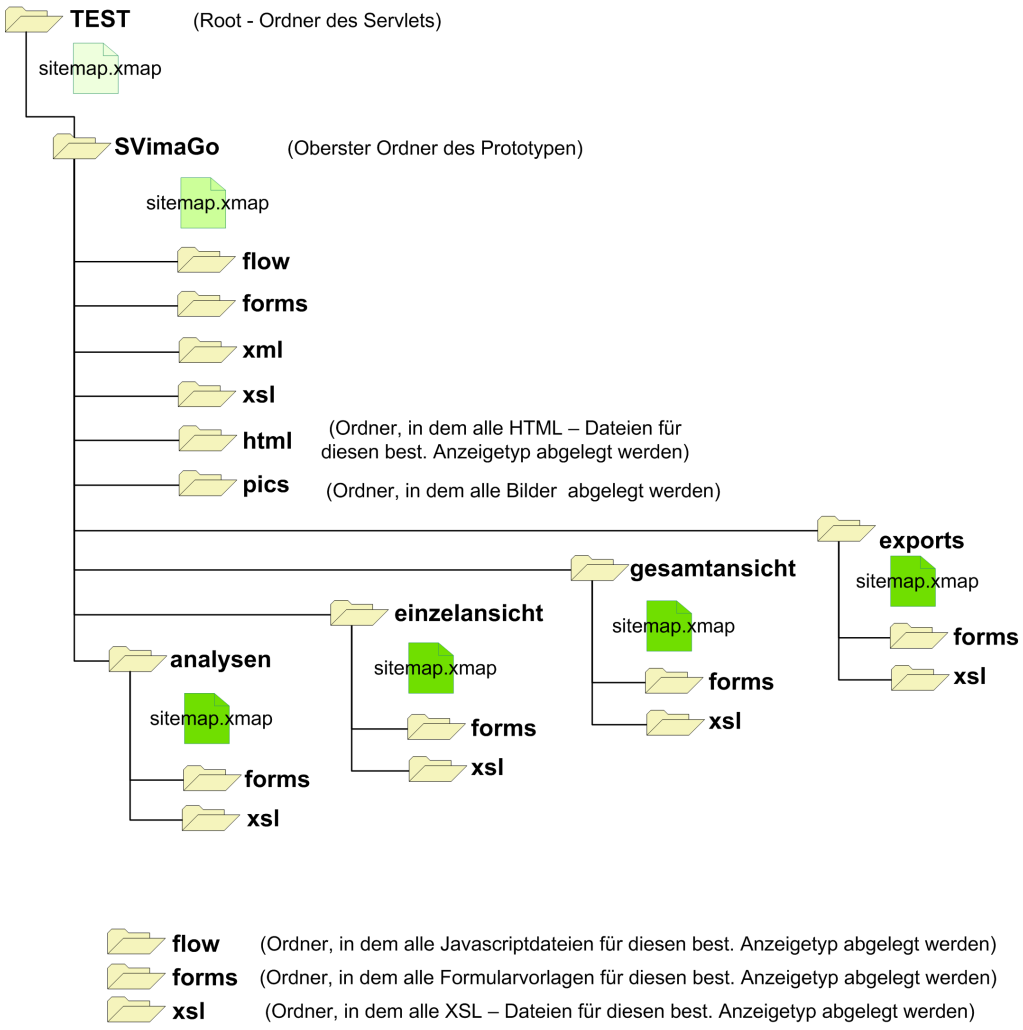

Abbildung 5.22: Anordnung der Prototypdokumente im Servlet

mit der Ordner- und Dokumentstruktur des Prototyps. Aus Gründen der Übersichtlichkeit und der Erweiterbarkeit werden die jeweiligen Dokumente eines Funktionsbereiches separat in einem eigenen Order im Servlet gespeichert. Bei den Dokumenten handelt es sich einerseits um die jeweilige Sitemap, andererseits um XSL-Stylesheets, Formulartemplates oder statische HTML-Seiten. Abbildung 5.22 zeigt eine Übersicht der Anordnung der Ordner und Dokumente. Dank dieser Struktur können dem Prototyp auf einfache Weise Erweiterungen

hinzugefügt werden. So muss lediglich ein neuer Ordner inklusive einer Sitemap erstellt werden, die in die oberere Sitemap eingebunden wird. Daraufhin können eigene Dokumente und Pipelines innerhalb dieses Bereiches erstellt werden.

5.4.4 Einbinden der XML-Datenbank eXist in Cocoon

Grundlage aller Transformationen des Prototyps sind in XML gehaltene Bordnetzdaten. In Kapitel 5.3 wurde bereits vorgestellt, wie diese in der XML-Datenbank eXist eingefügt und gespeichert werden. Nun gilt es, innerhalb von Cocoon auf diese Daten zuzugreifen. Dafür bestehen laut [MEIE2003d][14] drei verschiedene Möglichkeiten. Mittels dem XMLDBTransformer, den Extensible Server Pages (XSP)[15] oder dem XMLDB Pseudo Protokoll ist es möglich, aus Cocoon-basierten Anwendungen auf eXist zuzugreifen.
Der XMLDB-Transformer bietet nach [MEIE2003d][16] eine einfache Möglichkeit, Anfragen an die Datenbank zu richten. Er geht dabei wie die bisherigen Transformer von Cocoon vor. Grundlegend werden einkommende SAX-Events der vorherigen Pipeline nach bekannten Tags durchsucht, Aktionen ausgeführt und die ermittelten Tags mit den Ergebnissen der Aktionen ersetzt. Mit den XSPs steht ein Mechanismus zur Verfügung, auf XML basierend dynamische Web-Seiten zu entwickeln. Dabei wird Javacode in XML eingebettet. Um zu verhindern, dass größere Mengen an Javacode in die XML-Seiten eingefügt werden, kann wiederverwendbarer Code in so genannte Logic-Sheets ausgelagert werden. Dadurch werden Inhalt und Programmierlogik voneinander getrennt. Mit der Version 0.9.2 von eXist wird ein solches Logic-Sheet mitgeliefert. Dieses enthält grundlegende Tags, um von eXist auf Cocoon zuzugreifen. Weiterführende Informationen zur genauen Einsatzweise sind unter [BEMI2004] zu finden. Die dritte Möglichkeit, von Cocoon auf eXist zuzugreifen, basiert auf Cocoons Unterstützung von Pseudo-Protokollen ([MEIE2003e]). Durch diese ist es möglich, zur Einbingung von Ressourcen anstelle eines bekannten Protokolls wie http:// oder file:// ein selbstdefiniertes Protokoll in der Sidemap zu verwenden. Die aktuelle Version von Cocoon definiert ein Pseudo-Protokoll,

[14] [MEIE2003c] Meier, Wolfgang: *eXist: An Open Source Database*, Darmstadt University of Technology, URL: `http://exist-db.org/webdb.pdf`, Datum des Zugriffs: 15.10.2003.

[15] Mit Hilfe von Extensible Server Pages ist es möglich, XML-basierte dynamische Web-Seiten zu erstellen. Ähnlich zu den Java Server Pages wird auch bei den XSPs Java-Code in XML-Seiten eingebettet. XSP-Seiten sind im Gegensatz zu normalen Server Pages XML Dokumente. In der XSP Seite kann weiterhin anstatt von HTML Markup, eine Abstraktion in Form von selbstdefinierten Tags verwendet werden. Diese Tags können später an zentraler Stelle in HTML, WML oder PDF umgewandelt werden. Eine XSP Seite ist durch diese Trennung übersichtlicher und leichter zu warten.

[16] [MEIE2003d] Meier, Wolfgang: *eXist – Developers Guide*, URL: `http://exist.sourceforge.net/devguide.html`, Datum des Zugriffs: 03.10.2003.

um XML:DB API-fähige Datenbanken einzubinden. Es ist lediglich notwendig, in der Konfigurationsdatei von Cocoon die Treiberklasse der entsprechenden Datenbank zu spezifizieren. Diese drei Möglichkeiten wurden für die Umsetzung innerhalb der prototypischen Anwendung getestet. Dabei stellte sich heraus, dass die Dokumentation zur Einbindung des XMLDB-Transformers und der Extensible Serverpages unzureichend war. Trotz erheblicher Aufwände blieb ein befriedigendes Ergebnis aus, daher konnte ein Einsatz in der prototypischen Anwendung ausgeschlossen werden. Zwar war bezüglich des Pseudo-Protokolls für den Zugriff von Cocoon auf eXist ebenfalls keine ausführliche Dokumentation vorhanden, doch enthält die eingesetzte Version von Cocoon wie zuvor erwähnt ein vordefiniertes, sofort einsatzfähiges Protokoll, das lediglich geringer Veränderungen bedarf. In der Cocoon-Hauptkonfigurationsdatei `cocoon.xconf` wurde die Treiberklasse für eXist eingetragen. Dadurch war es möglich, eine XML:DB URI an jeder Stelle der Sitemap zu verwenden, an der eine URI als Eingabe erwartet wird. Da diese Möglichkeit ohne großem Aufwand sofort einsetzbar war und zufriedenstellende Ergebnisse erbrachte, werden innerhalb des Prototyps die Daten aus eXist in Cocoon über das Pseudo-Protokoll bezogen. Der folgende Ausschnitt einer Sitemap aus Cocoon stellt eine Pipeline dar, mit der auf diese Weise im Generator auf die XML-Datenbank eXist zugegriffen und der Inhalt des XML-Elementes `pinObj` abgefragt wird.

```
1 <map:pipeline>
2  <map:match pattern="Pin">
3  <map:generate src="xmldb:exist://localhost:8080/exist/xmlrpc/db/
4     #xcollection('plaene/PROJECT_9X7/MJ05/Basis9x7/DesignA/ACD3')//pinObj"/>
5  <map:serialize type="xml"/>
6  </map:match>
7 </map:pipeline>
```

Abbildung 5.23: Pipeline in der auf die XML-Datenbank eXist zugegriffen wird

Dazu wird im Attribut `src` des Elementes `<map:generate>` das Pseudo-Protokoll für den Zugriff auf eXist und die Serveradresse angegeben, sowie mittels der XPath-Erweiterung `xcollection()` der Pfad der Collection in der Datenbank, an die die Anfrage gestellt werden soll, bestimmt. Nach dieser Pfadangabe wird über einen XPath-Ausdruck das XML-Dokument, bzw. der Teil des XML-Dokuments, der adressiert werden soll, angefragt. Im vorliegenden Beispiel wird in der Collection `ACD3` des Projektes `PROJECT_9X7` im Modelljahr `MJ05` des Modelles `Basis9x7` und dem Design `DesignA` nach dem XML-Element `pinObj` gesucht. Das Ergebnis dieser Datenbankanfrage wird an den Serializer übergeben, der daraus ein XML-Dokument erstellt.

5.4.5 Transformationen der XML-Bordnetzdaten in verschiedene Ausgabeformate

Zuvor wurde die Umsetzung einzelner Teile der prototypischen Web-Anwendung vorgestellt, darunter die die Speicherung der XML-Bordnetzdaten in eXist, der Zugriff auf die gespeicherten Daten aus Cocoon oder die Verteilung eintreffender Anfragen auf verschiedene Sitemaps. Nun soll einerseits das Zusammenwirken dieser Bestandteile dargestellt, und andererseits, wie in Kapitel 2 gefordert, die Generierung verschiedener Sichten auf ein Bordnetz erläutert werden. Zusammenfassend wurden für die drei Bereiche „Gesamtansicht/Einzelansicht (SVimaGo)“, „Export“ und „Analysen“ der Web-basierten Benutzeroberfläche unterschiedliche Ansätze umgesetzt, XML-Bordnetzdaten in verschiedene Formate zu transformieren. Weiterführend werden die Bereiche „Gesamtansicht“ und „Export“ herangezogen, um die Funktionsweise der Transformationen des Prototyps zu erläutern.

Transformation des Bereiches „Gesamtansicht“

Im Bereich „Gesamtansicht“ wird eine Transformation der Bordnetzdaten aus XML nach SVG durchgeführt. Dabei stellt die generierte SVG-Graphik einen Schaltplan mit all seinen Komponenten, Beschriftungen Verbindungen und zusätzlichen Kommentargraphiken dar. Über ein Formular der Benutzeroberfläche, dargestellt in Abbildung 5.24 auf Seite 81, wird ein Schaltplan ausgewählt, von dem eine graphische Repräsentation in SVG generiert werden soll. Im vorliegenden Beispiel wurden die folgenden Quelldaten ausgewählt: Projekt `PROJECT 9X7`, Modelljahr `MJ05`, Modell `9x7Basis`, `DesignA`, Sheet `ACD3`. Beim Abschicken des Formulars werden diese Eingaben an die URI angehängt, um so der Sitemap mitzuteilen, welcher Schaltplan transformiert werden soll. Im vorliegenden Fall wird folgende URI an das Cocoon-Servlet gesendet: `http://Servername:Serverport/TEST/SVimaGo/gesamtansicht/PROJECT9X7/MJ05/9x7_Basis/DesignA/ACD3`. Wie in Kapitel 5.4.3 erläutert, werden Anfragen für Verarbeitungspipelines direkt an die Sitemap des jeweiligen Bereiches weitergeleitet. Die oben genannte URI verweist somit auf die Sitemap des Bereiches “Gesamtansicht“. In dieser wird nun die Pipeline „getSVG“ aufgerufen, an die die Formularwerte übergeben werden. Bezogen auf das Schnittstellenmodell, das den XML-Bordnetzdaten zugrunde liegt, sind für die reine Darstellung eines Schaltplanes in SVG lediglich die Dokumente des Repräsentationsmodells notwendig (siehe Abbildung 5.2). Die Abbildung ?? auf Seite ?? enthält eine Übersicht der Vorgänge, durch die XML-Dokumente aus der Datenbank bezogen und verarbeitet werden. Da durch den Einsatz des Pseudo-Protokolls lediglich ein Dokument über

Abbildung 5.24: Benutzeroberfläche des Prototyps, Bereich Gesamtansicht

den Generator aus der Datenbank bezogen werden kann, wird die so genannte Content Aggregation Technologie eingesetzt. Dadurch ist es möglich, die Resultate mehrerer Pipelines zusammenzufügen und zu bearbeiten. Bei der vorliegenden Transformation wurde für die Darstellung eines Schaltplanes in SVG für jedes benötigte Dokument eine Pipeline angelegt. Der Sitemap-Ausschnitt der Pipeline für das Dokument `pinObj.xml` ist in Abbildung 5.25 dargestellt. Mit den beiden Sternsymbolen `**` im `pattern`-Attributwert werden die Formularwerte übergeben. In der nächsten Zeile ist zu erkennen, wie mittels des Generators über das Pseudo-Protokoll auf die XML-Datenbank eXist zugegriffen wird. In der XML:DB URI `xmldb:exist://localhost:8080/exist/xmlrpc/db/#xcollection('plaene/{1}')//pinObj` werden anstelle des Platzhalters `{1}` die übergebenen Werte des Formulars eingesetzt. Dadurch kann in der gewünschten Collection nach dem Element `pinObj` gesucht werden. Bei dieser Abfrage wird eine Erweiterung von XPath eingesetzt (`xcollection`), wodurch nur in der angegebenen Collection und keiner

```
<map:pipeline>
 <map:match pattern="Pin/**">
  <map:generate src="xmldb:exist://localhost:8080/exist/xmlrpc/db/
    #xcollection('plaene/{1}')//pinObj"/>
  <map:transform src="xsl/pinObj.xsl"/>
  <map:serialize type="xml"/>
 </map:match>
</map:pipeline>
```

Abbildung 5.25: Beispiel-Pipeline der Gesamtansicht-Sitemap, welche auf die XML-Datenbank eXist zugreift

Unter-Collection nach diesem Element gesucht wird. Nach der erfolgreichen Übernahme des Dokumentes aus eXist werden aus diesem durch das XSL-Stylesheet `pinObj.xsl` alle für die Darstellung eines Pins in SVG notwendigen Informationen ausgelesen. Diese Informationen werden für die weitere Verarbeitung aufbereitet und in Form von XML an die übergeordnete Pipeline geschickt. Innerhalb der übrigen aggregierten Pipelines werden auf diese Weise die Informationen zur Darstellung des Zeichnungsrahmens, aller Texte, Instanzen und Segmente ausgelesen und aufbereitet, sowie die Informationen für das Verschieben des Schaltplanes in den sichtbaren Bereich des SVG-Koordinatensystems. Mit dem XSL-Stylesheet `all2svg.xsl` werden die Ergebnisse der Pipelines in einem SVG-Dokument platziert. Dieses wird zunächst als XML-Dokument an die SVimaGo Sitemap zurückgeschickt, die daraus eine SVG-Datei erzeugt und an den Client-Browser zurücksendet. Die generierte SVG-Graphik dieser Transformation ist in der Abbildung 5.26 dargestellt.

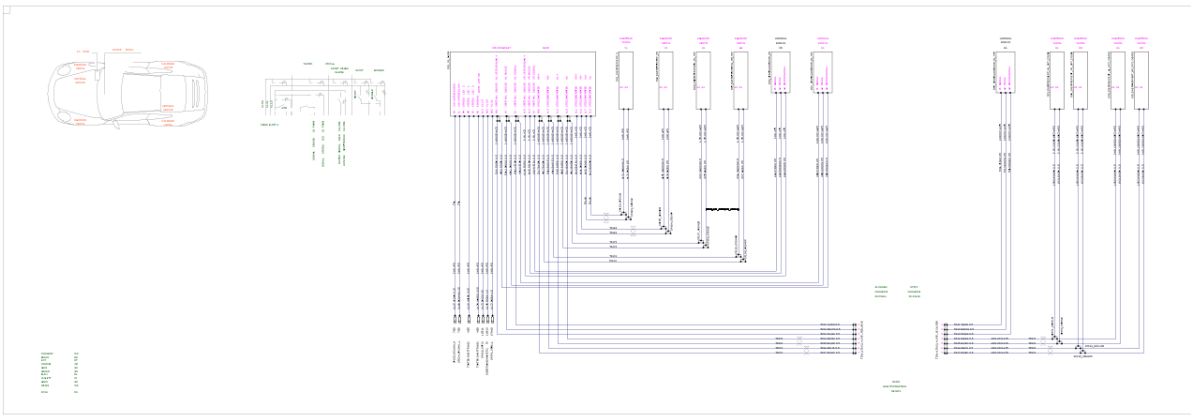

Abbildung 5.26: Generierte Darstellung des Schaltplanes ACD3 in SVG

Bei dieser Transformation mussten lediglich die Koordinaten und weitere Darstellungsda-

ten aus den XML-Dokumenten ausgelesen und in einer SVG-Datei platziert werden, wofür sich XSLT bestens eignet. Eine ca. 550 KB große SVG-Graphik wurde innerhalb von 12 Sekunden generiert, inklusive der Zugriffe auf die Datenbank, der Transformationen und des Sendens der Graphik an den Client. Die SVG-Graphiken, die auf diese Weise generiert wurden, enthalten allerdings keinerlei Interaktivität. Soll diese hinzugefügt werden, muss bei der Generierung der Graphik die Logik des Schaltplanes miteinbezogen werden. Dies bedeutet, dass nicht nur die Daten des Repräsentationsmodells, sondern auch die des logischen und physikalischen Modells bei den Transformationen berücksichtigt werden müssen. Dadurch wachsen die erforderlichen Transformationen und die zu bearbeitende Datenmenge sprunghaft an, was mit einer Vervielfachung der Transformationszeiten einhergeht. Verschiedene Versuche, diese Daten mit einzubeziehen, zeigten, dass XSLT in diesem Fall nur beschränkt einsetzbar ist. Beispielsweise sind die Realisierung von Schleifen mittels rekursiver Abfragen, bzw. die dynamische Belegung von Variablen ist nicht möglich. Dadurch entstehen „umständliche“ Stylesheets, die die Übersichtlichkeit und Wartbarkeit der Anwendung beeinträchtigen.

Transformation des Bereiches „Export“

Die zweite Transformation, deren Funktionsweise erläutert werden soll, ist die des „Exports“. Über ein ähnliches Formular wie dem der Gesamtansicht wird ein Schaltplan ausgewählt, von dem eine Netzliste generiert werden soll. Ein Netz bezeichnet hierbei die Verbindung einer elektrischen Komponente zu einer anderen. In einer Netzliste wird an dieser Stelle eine Übersicht aller Verbindungen eines Schaltplanes in einer HTML-Tabelle dargestellt. Dabei kann ein Netz eine Vielzahl unterschiedlicher Attribute besitzen, die in dieser Tabelle aufgeführt werden können. In diesem Fall wird für jedes Netz der Netzname, der Querschnitt und die Farbe angegeben, ebenso die Inhalte der Attribute `comment_1` und `alias_ref` beider beteiligter Komponenten. Wie beim Abschicken des Formulars der Gesamtansicht, werden die Formularwerte der URI beigefügt und der Sitemap übergeben, die für die Verarbeitung des Bereiches „Export“ zuständig ist. In diesem Fall handelt es sich um die „Export Sitemap“. Die Anfrage wird der Pipeline `export_wire` übergeben, deren Ablauf in der Abbildung 5.27 dargestellt ist. Innerhalb dieser Pipeline werden, wie schon zuvor, XML-Daten aus der XML-Datenbank über das Pseudo-Protokoll bezogen und mit Hilfe von XSLT-Stylesheets bearbeitet. Anders als im vorangegangenen Beispiel werden dabei für die Netzlistengenerierung nur die logischen und physikalischen Daten des Schnittstellenmodelles verwendet. Dies gründet darauf, dass nun lediglich die Informationen relevant sind, die die Zusammenhänge und die

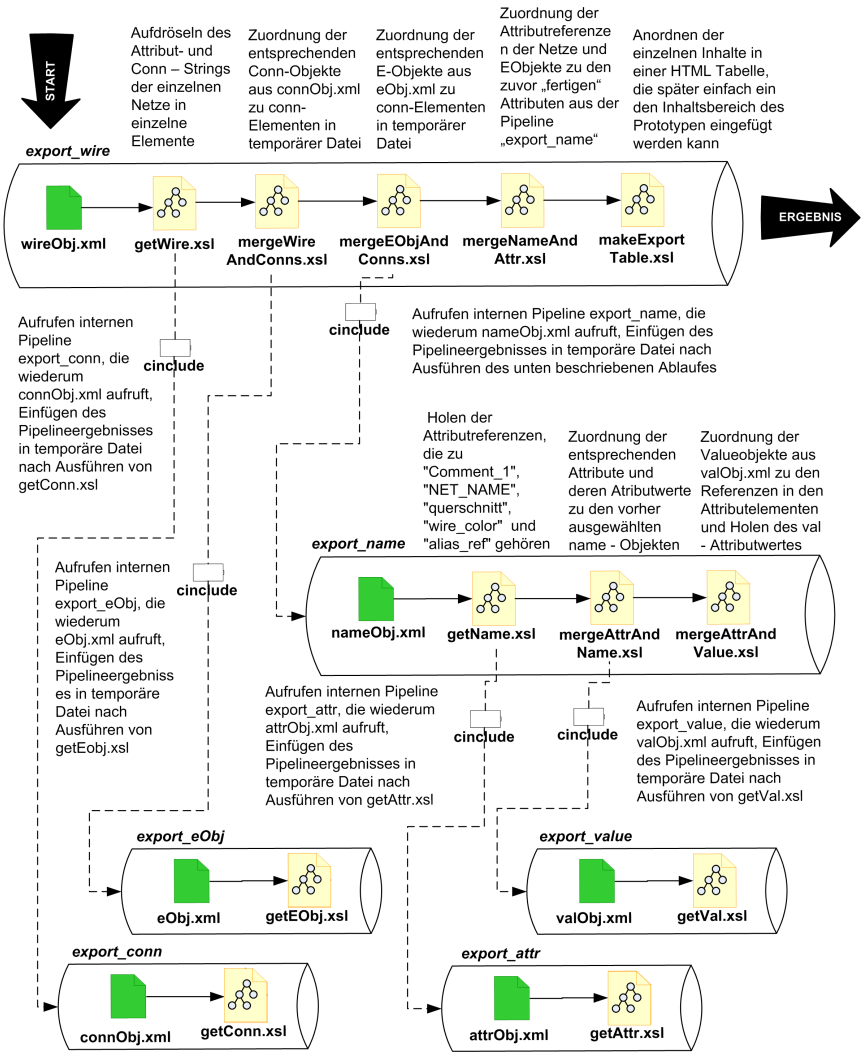

Abbildung 5.27: Transformationspipeline des Exports

Logik eines Bordnetzes beinhalten. Wie in Abbildung 5.27 erkennbar, ist die Zusammenstellung der erforderlichen Informationen für eine Netzliste mittels XSLT weitaus aufwändiger, als die Darstellung eines Schaltplans in SVG. Es werden verschiedene Stylesheets benötigt, die die Informationen aus der Datenbank aufbereiten und Zuordnungen von Referenzen zu Objekten durchführen. Zusätzlich zur Content Aggregation wird hierbei CInclude[17] eingesetzt. CInclude besitzt dieselben Funktionalitäten wie XInclude[18], jedoch ist es auf den Einsatz in

[17] Nähere Informationen zu Cinclude: `http://wiki.cocoondev.org/Wiki.jsp?page=CInclude`.

[18] XML Inclusions ist eine W3C Candidate Recommendation, mit deren Hilfe große XML-Dokumente aus vielen wohlgeformten XML-Dokumenten gebildet werden können. Jeder Teil kann ein komplettes XML-Dokument sein, ein Fragment eines XML-Dokumentes oder ein normales Text-Dokument, wie ein

Cocoon abgestimmt, vielseitiger und performanter. Dadurch ist es möglich, innerhalb eines XSLT-Stylesheets das Ergebnis einer weiteren Pipeline mit einzubeziehen und zu verarbeiten. Komplexe Pipelines können somit in kleinere Pipelines aufgeteilt und erst zum gewünschten Zeitpunkt verwendet werden. Weiterhin können immer wieder auftretende und ähnliche Verarbeitungen in einer eigenständigen Pipeline ausgelagert und so wiederverwendbar gemacht werden. Allerdings ist bei dieser Variante zu beachten, dass nicht zu viele kleinere Pipelines eingebunden werden, da darunter die Performanz leidet. Als Einstiegspunkt der Transformation wird die Datei `wireObj.xml` gewählt, da in dieser alle Netze eines Schaltplanes enthalten sind. In der Pipeline `export_wire` werden somit aus der Datei `wireObj.xml` alle in einem Schaltplan vorhandenen Netze bezogen. Mittels einer weiteren Pipeline `export_conn`, deren Ergebnis über CInclude in die Pipeline `export_wire` eingebunden wird, werden alle Verbindungen der jeweiligen Netze aus der Datenbank abgefragt. Die Zuordnung der Verbindungen zu den Netzen erfolgt in einem XSLT-Stylesheet. Mit Hilfe einer weiteren Pipeline, die über das CInclude eingebunden wird, werden die zu den Verbindungen gehörigen E-Objekte erfasst. Diese sind notwendig, um die Inhalte der Attribute `comment_1` und `alias_ref` der an den Verbindungen beteiligten E-Objekte darstellen zu können. Die Zuordnung der E-Objekte zu den Verbindungen wird ebenfalls wieder in einem XSLT-Stylesheet durchgeführt. Nun stehen folgende Informationen zur Verfügung: die Liste aller Netze eines Schaltplanes, denen Verbindungen zugeordnet sind, sowie die zu den Verbindungen gehörigen E-Objekte. Für die Darstellung der zuvor beschriebenen Netzliste mit Netznamen, Querschnitt usw. müssen verschiedene Attributwerte der Netze und E-Objekte aus der Datenbank bezogen werden. In einer weiteren Pipeline `export_name` erfolgt die Abfrage dieser Attributwerte. Mittels verschiedener XSLT-Stylesheets werden die Attributwerte den Netzen und E-Objekten zugeordnet. Nachdem alle Informationen für eine Netzliste vorhanden sind, werden diese mit Hilfe des XSLT-Stylesheets `makeExportTable.xsl` in einer HTML-Tabelle angeordnet und der übergeordneten Sitemap übergeben. In der Pipeline `getNetList` wird diese Tabelle durch zwei Layout-Stylesheets im Export-Tab platziert und daraufhin an den Client gesendet. Die Transformationszeit variiert dabei in Abhängigkeit von der Größe des Schaltplanes und den Attributwerten, die in der Netzliste angezeigt werden sollen. Für das vorherige Beispiel des Sheets ACD3 dauert der Vorgang der kompletten Netzlistentransformation ca. 14 Sekunden. Das Ergebnis ist in Darstellung 5.28 abgebildet.

Java-Programm oder eine Email. Nähere Informationen zu Xinclude: `http://www.w3.org/TR/2004/CR-xinclude-20040413/`.

SVimaGo

SVimaGo | Export | Analysen

Netzliste von: PROJECT_9X7/MJ05/9x7_Basis/DesignA/ACD3

Netz-Nr.	Netzinformationen			von Komponente		nach Komponente	
	Netzname	Querschnitt	Farbe	Kommentar	Alias-Referenz	Kommentar	Alias-Referenz
0	5211 YE/BN 0.5	0.5	YE/BN-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
1	5212 BN/OG 0.5	0.5	BN/OG-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
2	5213 RD/BK 0.5	0.5	RD/BK-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
3	5214 BN/BK 0.5	0.5	BN/BK-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
4	5215 BK/RD 0.5	0.5	BK/RD-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
5	5216 BN/YE 0.5	0.5	BN/YE-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
6	5217 BK/BK 0.5	0.5	BK-FLRY/A	-	TS013_TSX2_4_997_FGR_HECK	-	-
7	4025 BK/BK 0.5	0.5	BK-FLRY/A	-	SP186_SPDHLP	DAEMPFER-	508_DAEMPFERVENTIL_HL_987
8	4024 WH/WH 0.5	0.5	WH-FLRY/A	-	SP187_SPDHLM	DAEMPFER-	508_DAEMPFERVENTIL_HL_987
9	4027 BK/BK 0.5	0.5	BK-FLRY/A	-	SP188_SPDHRP	DAEMPFER-	509_DAEMPFERVENTIL_HR_987
10	4026 WH/WH 0.5	0.5	WH-FLRY/A	-	SP189_SPDHRM	DAEMPFER-	509_DAEMPFERVENTIL_HR_987
11	6167 YE/WH 0.35	0.35	YE/WH-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
12	6168 BK/WH 0.35	0.35	BK/WH-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
13	6169 GN/BK 0.35	0.35	GN/BK-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
14	6174 WH/RD 0.35	0.35	WH/RD-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
15	6175 BK/OG 0.5	0.5	BK/OG-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
16	6176 RD/BK 1.5	1.5	RD/BK-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
17	6177 BN/BN 1.5	1.5	BN-FLRY/A	-	-	STEUERGERAET	500_SG_PASM
25	6170 BK/BK 0.5	0.5	BK-FLRY/A	DAEMPFER-	502_DAEMPFERVENT_VR	-	SP184_SPDVRP

Abbildung 5.28: Teil der nach HTML generierten Netzliste des Schaltplanes ACD3

Da die Transformationszeiten für diesen Vorgang, bzw. die Erstellung der Gesamtansicht zufriedenstellen sind und den Anforderungen genügen, werden keine weiteren Veränderungen an der Konfiguration von Cocoon oder eXist vorgenommen. Zusammenfassend wird weiterhin festgestellt, dass die Transformation der Netzlisteninformationen eines Schaltplanes aufgrund der Verarbeitung von logischen Daten des Schnittstellenmodelles mehr Zeit beansprucht. Dies wird unter anderem durch komplexere Stylesheets bedingt, die die Zusammenhänge der Daten zunächst verarbeiten und letztendlich in HTML darstellen.

6 Fazit und Ausblick

6.1 Fazit

Mit der Umsetzung des Prototyps wurde die Zielsetzung erreicht, dynamische Transformationen der Bordnetzdaten von XML nach SVG oder HTML durchzuführen. Dafür wurden applikationsneutrale XML-Bordnetz-Schaltplandaten von LCable in der XML-Datenbank eXist abgelegt. Mit Hilfe des Web Publishing Frameworks Cocoon wurde eine Browser-basierte Benutzeroberfläche entwickelt, über die die Generierung verschiedener Sichten auf die Bordnetzdaten gestartet und das Resultat angezeigt werden kann. Für die Transformation der XML-Bordnetzdaten wurde XSLT eingesetzt. Aus der Umsetzung des Prototyps konnten, bezogen auf die Fragestellungen des Kapitels 2, folgende Erkenntnisse gezogen werden:

Applikationsneutrale Abbildung der Bordnetzdaten in XML

Anhand des Schnittstellenmodells, das in Kapitel 5.2 vorgestellt wurde, können Bordnetz-Schaltplandaten applikationsneutral in XML abgebildet werden. Diese XML-Daten enhalten keine LCable-spezifischen Informationen, wodurch es möglich ist, ohne Kenntnisse über den Schaltplaneditor LCable Schaltpläne in SVG „nachzuzeichnen“.

Ableitbarkeit verschiedener Sichten auf ein Bordnetz

Mit der entwickelten Applikation wurde bewiesen, dass aus den in XML gehaltenen Bordnetz-Schaltplandaten mittels XSLT verschiedene Sichten dynamisch generiert werden können. Das den XML-Bordnetzdaten zugrunde liegende Schnittstellenmodell stellt die Basis für eine Vielzahl von Transformationsmöglichkeiten dar, wobei innerhalb des Prototyps zunächst vier verschiedene Transformationen umgesetzt wurden.

Eignung von XML für die Speicherung von Bordnetzinformationen

Die Transformationen des Prototyps basieren auf Bordnetz-Schaltplandaten im XML-Format. Die klare Trennung von Daten, Programmierlogik und Darstellung, sowie die Möglichkeit, die komplexe Struktur eines Bordnetzschaltplanes in XML abzubilden, sind Fürsprecher für den Einsatz von XML in der Bordnetzentwicklung. Weitere Vorzüge dieser Strategie wurden in Kapitel 3.2.3 bereits aufgezeigt, wodurch abschließend die Eignung von XML für die Speicherung von Bordnetzinformationen festgestellt werden kann.

Eignung von XSLT für die Aufbereitung von Bordnetzdaten

Wie zuvor in Kapitel 5.4.5 angedeutet, eignet sich XSLT für die Aufbereitung von Bordnetz-Schaltplandaten nur bedingt. Für einfachere Transformationen von XML-Bordnetzdaten, wie beispielsweise die realisierte Gesamtansicht des Prototyps, ist XSLT gut geeignet. Sobald jedoch versucht wird, mittels XSLT Anwendungslogik umzusetzen (z.B. den logischen Teil des Schnittstellenmodells), wird die Erstellung der XSLT-Stylesheets durch die begrenzten Möglichkeiten XSLTs (z.B. keine Schleifen möglich, keine Variablen) umständlich. Zusätzlich wird die Auswertung der umfangreichen Stylesheets erschwert und die Fehlersuche gestaltet sich aufwändig. Zugleich ist XSLT sehr CPU- und Speicher-intensiv, wodurch die Transformationszeiten negativ beeinflusst werden. Daher ist es sinnvoll, bei komplexeren Transformationen die Anwendungslogik mittels DOM oder SAX umzusetzen, wofür Java, C oder C++ verwendet werden können.[1]

Eignung von XML als alleiniges Datenformat in der Bordnetzentwicklung

Da sich diese Diplomarbeit nur mit einem Teilbereich der Bordnetzentwicklung beschäftigt, kann eine generelle Aussage über die Eignung von XML als alleiniges Datenformat in der Bordnetzentwicklung nicht getroffen werden. Innerhalb des behandelten Teilbereiches ist XML als alleiniges Datenformat sehr gut geeignet. Dabei handelt es sich zunächst um die Bordnetz-Schaltplandaten des Schaltplaneditors LCable, die in XML abgebildet wurden. Die Darstellung verschiedener Sichten auf diese Bordnetz-Schaltplandaten konnte allein mit den in XML vorliegenden Daten realisiert werden – es wurden keine weiteren Informationen aus anderen Quellen benötigt. Alle notwendigen Daten konnten in XML abgebildet werden.

[1] vgl. [BURK2001] Burke, Eric M.: *Java and XSLT*, 1. Auflage, September 2001, Kapitel 5.

6.2 Ausblick

6.2.1 Allgemein

Nunmehr gilt es, den möglichen Produktivbetrieb des prototypischen Applikations-Konzepts zu bewerten. Aufgrund der Erkenntnisse aus der Entwicklung des Prototyps ist folgender Ansatz denkbar: Basierend auf dem Konzept des Prototyps wird eine Plattform geschaffen, die nicht nur entwicklungsbezogene, sondern alle die Bordnetzentwicklung betreffenden Daten zentral zur Verfügung stellt. Dazu gehören beispielsweise Daten des Kundendienstes, sowie die der Schaltplaneditoren CATIA und LCable. Die Informationen dieser verschiedenen Datenquellen werden anhand eines Modelles nach XML überführt und an einer zentralen Stelle persistent gespeichert. Daraufhin können Transformationen auf Basis dieser Daten über Auswahllisten gestartet und im Browser angezeigt werden. Denkbar ist zusätzlich eine Authentisierung des Nutzers, wodurch je nach Anwendergruppe und Benutzer eine individuelle Benutzeroberfläche mit eingeschränkten Auswahlformularen erscheint. Dadurch kann unter anderem der Zugriff Unbefugter beschränkt werden. Aufgrund der geringen Anforderungen, die an den Client gestellt werden, kann nahezu applikations- und plattformunabhängig auf den gesamten, die Bordnetzentwicklung betreffenden, Datenbestand zugegriffen werden. Die verschiedenen Sichten, die aus den nach XML überführten Daten generiert werden können, sind vielfältig. Möglich sind beispielsweise Analysen, Suchfunktionen, Schaltplandarstellungen uvm. in den Ausgabeformaten HTML, XHTML, XML, PDF, SVG, MS Excel. Der Phantasie sind somit kaum Grenzen gesetzt.

6.2.2 Prototyp

Um aus dem Prototyp ein funktionstüchtiges System zu entwickeln, müssen verschiedene Bereiche weiterentwickelt oder verändert werden.

Schnittstellenmodell

Das Schnittstellenmodell, das den XML-Schaltplandaten des Prototyps zugrunde liegt, berücksichtigt zunächst nur die Bordnetzdaten des Schaltplaneditors LCable. Um im lauffähigen System auf alle, die Bordnetzentwicklung betreffenden Daten zugreifen zu können, muss dieses erweitert werden. Dadurch können beispielsweise dreidimensionale Daten aus CATIA in die Sichtgenerierungen von XML nach SVG einbezogen werden.

Da während der Diplomarbeit keine XML Schema Definitionen zum Schnittstellenmodell existierten, sollte bei der Integration der „übrigen“ die Bordnetzentwicklung betreffenden Daten ein solches erstellt werden. Diesem kann eine Modellierung der Daten, z.B. mittels UML, vorausgehen, in der die Abläufe und Informationsflüsse beschrieben sind, sowie die Strukturen und Bedeutungen der Informationen in einem Dokument. Dem folgt der Entwurf der Dokumente, wobei die gewonnenen Kenntnisse nach Regeln innerhalb eines XML Schemas oder einer DTD umgesetzt werden.[2]

Performanz

Im Folgenden sollen mögliche Strategien zur Verbesserung der Transformations-Performanz und der Antwortzeiten im Hinblick auf ein einsetzbares System kurz vorgestellt werden. Wie zuvor festgestellt, ist XSLT für die Transformation von XML-Bordnetz-Schaltplandaten nur bedingt geeignet. Eine mögliche Alternative zur Performanzverbesserung ist eine zweistufige Bearbeitung der XML-Dokumente. Zunächst kann durch eine Applikation, die mit Java, C oder C++ umgesetzt ist und die entweder die DOM oder SAX API verwendet, eine Restrukturierung der Daten vorgenommen werden. Im nächsten Schritt erfolgt dann die Transformation dieser XML-Daten mittels XSLT in das gewünschte Ausgabeformat. Weiterhin bestehen verschiedene Strategien, XSLT-Stylesheets zu verkürzen und somit übersichtlicher zu gestalten. Dazu gehören z.B. die Verwendung von Entities oder die Auslagerung sich wiederholender Abläufen in separaten Stylesheets, die mit `<xsl:include>` eingebunden werden können.[3] Darüber hinaus sollte die Möglichkeit von Vorab-Generierungen bedacht werden. Ist die Anzahl der zu generierenden Sichten nicht allzu groß und die XML-Bordnetzdaten ändern s ich nicht in unregelmäßigen, zeitlich nahen Abständen, empfiehlt sich die Vorab-Generierung dieser Sichten durch Batchprozesse. Dadurch können Transformationszeiten verkürzt und die generierten Sichten statisch zur Verfügung gestellt werden, wodurch sich die Antwortzeiten wesentlich reduzieren. Zudem sollten zusätzlich Cachingmechanismen eingesetzt werden, um einerseits Ressourcen zu sparen und andererseits durch das Zwischenspeichern von Anfrageantworten die Antwortzeiten ebenfalls zu verkürzen. Um dies zu erreichen, kann z.B. ein transparenter Proxy[4] eingesetzt werden. Zum anderen bietet das XML-Framework Cocoon

[2] vgl. [ABM2000b] Anderson R., Birbeck M., Kay M. u.a.: *XML Professional, Kapitel 4: Datenmodellierung in XML*, 1.Auflage, MITP Verlag GmbH Bonn, 2000, Seite 123ff.

[3] vgl. [XMLM2003] XML Magazin und Web Services: *Kurzschreibweise, Tipps zur Verkürzung von XSLT-Skripten*, in: XML Magazin und Web Services, 2003, Heft-Nr. 03, Seite 39 - 41.

[4] Die Verwendung eines Proxy-Servers muss meist dem Client explizit mitgeteilt werden. Ein transparenter Proxy muss hingegen nicht explizit angegeben werden. Ein Router erkennt die Verwendung des vom Proxy

einen eigenen Cachingmechanismus, der steuert, ob ein Dokument aus dem Cache geholt werden kann, oder neu generiert werden muss. Ähnlich verhält es sich mit dem Component-Pooling von Cocoon, bei dem ein bestimmter „Pool“ an Komponenten für die Bearbeitung von Anfragen einmalig angelegt wird. Bei einer Anfrage werden die entsprechend benötigten Komponenten des Pools gesperrt und nach der Bearbeitung wieder „entlassen“. Somit müssen diese bei einer nochmaligen Anfrage nicht wieder von Neuem erstellt, und anschließend wieder zerstört werden, was zeitlich sehr aufwändig ist. Wie in Kapitel 4.1 dargestellt, kann die Bearbeitung von Transformationen über mehrere Rechner verteilt werden. Auch dies sollte in Betracht gezogen werden, um die Transformationen so schnell wie möglich durchzuführen und eine Ausfallsicherheit durch Redundanz zu gewährleisten.

Datenhaltung

Anhand verschiedener Anforderungen wurde mittels eines Auswahlverfahrens die XML-Datenbank eXist für die Datenspeicherung der XML-Bordnetzdaten gewählt. Laut Mittermeier[5] sind die Standards und Produkte von XML-Datenbanken noch relativ neu und befinden sich teilweise in einem intensiven Entwicklungsprozess. Dadurch können bei der Verwendung einer nativen XML-Datenbank mitunter Schwierigkeiten auftreten, die beim Einsatz einer relationalen Datenbank nicht anzutreffen sind.Dennoch empfiehlt sich auch weiterhin der Einsatz einer XML-Datenbank, da die Abbildung bzw. Erhaltung der komplexen Struktur des Schnittstellenmodells, die den XML-Bordnetzdaten zugrunde liegt, sowie die Forderung nach einem schnellen Zugriff auf die Element- und Attributebenen der XML-Dokumente von XML-Datenbanken eher erfüllt werden, als von relationalen Datenbanken. Bei letzteren müsste dafür der Inhalt der XML-Dokumente auf verschiedene Tabellen aufgeteilt werden (siehe Kapitel 4.3.2). Dies gestaltet sich je nach Komplexität des zugrunde liegenden Modells schwierig, wobei auch eine hundertprozentige Wiederherstellung der XML-Dokumente nicht gegeben ist. Beim Einsatz einer XML-Datenbank empfiehlt es sich laut Wilcox[6], diese für die schnellere Bearbeitung von großen XML-Dokumenten in der Applikation einzubetten. Neben der Verwendung einer XML-Datenbank für die Speicherung der Bordnetzdaten

verwendeten Protokolls und leitet die Anfragen an den Proxy weiter, ohne dass das Anwendungsprogramm etwas davon bemerkt.

[5] vgl. [MITT2003] Mittermeier, Ludwig: *XML und Datenbanken, naiv nativ*, in: iX, 2003, Heft-Nr. 8, Seite 42 ff.

[6] [WILK2003] Wilcox, Mark: *Using Embedded XML Databases to Process Large Documents*, 22. Oktober 2003, URL:`http://www.xml.com/lpt/a/2003/10/22/embed.html`, Datum des Zugriffs: 06.11.2003.

bestehen noch andere Möglichkeiten, Xml-Dokumente zu halten. Wird das lauffähige System beispielsweise auf Online Recherchen der Bordnetzdaten ausgerichtet, erscheint eine multidimensionale Datenbank[7] als geeignetes Mittel. Dadurch sind Adhoc-Anfragen, sowie großflächige Auswertungen möglich. Wird das System eher für einen schnellen Zugriff, beispielsweise auf SVG-Graphiken, erstellt, die vollständige oder teilweise ausgeführte Bordnetz-Schaltpläne darstellen, kann eine Multimedia Datenbank verwendet werden. In dieser können vorgenerierte Graphiken oder Analysen abgelegt und performant zur Verfügung gestellt werden. Dies hat allerdings den Nachteil, dass keine Flexibilität bezüglich der Anfragen mehr möglich ist und alle Abfragevarianten bedacht werden müssen.

Transaktionskonzept

Bei dem entwickelten Prototyp handelt es sich um eine entwicklungsbegleitende Web-Anwendung. Der Dialog der Benutzeroberfläche des Prototyps ist nicht transaktionsgesteuert, es handelt sich um einen reinen Auskunfts-Dialog. Um jedoch Anwender-bezogen auf Transformationsanfragen reagieren und Unbefugten den Zugriff verweigern zu können, muss ein Transaktionskonzept entwickelt werden. Dabei kann ein Web-Server die Verwaltung der Sessions übernehmen, wobei der Application Server zustandsfrei bleibt. Auf diesem wird lediglich ein Nutzer angelegt und es werden ausschließlich an diesen Nutzer gerichtete Anfragen bearbeitet. Die Erarbeitung eines solchen Transaktions- und Nutzerkonzeptes ist sehr komplex und und geht mit hohem Zeitaufwand einher.

Erweiterung des SVG-Plugins

Ein weiterer Ansatz für die Weiterentwicklung ist die Erweiterung des SVG-Plugins. Basierend auf einem existierenden Nutzerkonzept kann das Kontextenü des Plugins je nach mitgesendeter Rolle oder Nutzer durch Javascript erweitert oder verändert werden.

Verwendung eines Frameworks

Als Basis der prototypischen Applikation diente das XML-Publishing Framework Cocoon. Dies kann durch weitere Frameworks, wie beispielsweise das kommerzielle Produkt OXF[8] ersetzt werden. Es ist des Weiteren ebenfalls denkbar, die Vorzüge verschiedener Frameworks zu

[7] Eine multidimensionale Datenbank ist ein Datenbanktyp, der für die Bearbeitung von Data Warehouse-Applikationen und Online-Analysen optimiert ist.

[8] Nähere Informationen zu OXF: `http://www.orbeon.com/oxf/`.

kombinieren. Somit können beispielsweise Transformationen mit Cocoon, und die Oberfläche mit Hilfe des Struts Frameworks umgesetzt werden.

Historisierung

Es sollte ein Konzept erarbeitet werden, das die Versionierung der in der Datenbank gehaltenen Bordnetzdaten beschreibt. Darin muss enthalten sein, wann ein Schaltplan den Status einer Version erlangt und der XML-Export in LCable (und CATIA) gestartet wird, wann diese Version in die Datenbank eingepflegt und wie die Versionierung allgemein in der Datenbank umgesetzt wird.

Zeitlicher Aufwand

An dieser Stelle eine Abschätzung des zeitlichen Aufwands zu geben, ist nahezu unmöglich. Einerseits fehlen dazu praktische Erfahrungen, die eine Einschätzung des erforderlichen Zeitaufwands für bestimmte Aufgaben erlauben würden. Andererseits treffen hierbei zu viele Unwägbarkeiten aufeinander, wie beispielsweise die Erarbeitung eines Benutzerkonzeptes, eines Versionierungsystems, sowie die Erweiterung des Schnittstellenmodells um zusätzliche Bordnetzdaten etc. Der Umfang und die Anforderungen an diese verschiedenen Bereiche müssen zunächst genauer spezifiziert werden, um eine zeitliche Abschätzung des Aufwandes abgeben zu können.

7 Dokumentation der Diplomarbeit

Dieser Abschnitt ist dem Web-Tagebuch gewidmet, welches zu Beginn der Diplomarbeit mit Hilfe des Content Management Systems ZOPE[1] umgesetzt wurde. Dieses gliedert sich in folgende Bereiche:

Diplomarbeit Unter diesem Punkt ist der aktuelle Stand der Diplomarbeit zu finden, wie beispielsweise Zwischenberichte, der Titel und eine nähere Beschreibung der Diplomarbeit.

Tagebuch Hier sind die einzelnen Arbeitstage aufgelistet, jeweils mit Literaturanhang und einer Beschreibung der Arbeit an diesem Tag (siehe Abbildung 7.1).

Projektplan Über einen Projektplan in Tabellenform, sowie in Excel, werden die Projekstufen aufgeführt, in die die Arbeit der Diplomarbeit thematisch und zeitlich unterteilt ist.

Literatur Das Literaturverzeichnis enthält Links, Artikel, PDFs usw. die im Laufe der Diplomarbeit gefunden wurden.

Vortrag Hier ist eine kleine Übersicht über den Ort und die Zeit meines Diplomvortrages zu finden.

Impressum Im Impressum ist eine Übersicht der an der Diplomarbeit Beteiligten dargestellt.

Über das Web-Tagebuch wurde ohne großen Aufwand täglich ein Eintrag zur Arbeit an der Diplomarbeit hinzugefügt, wodurch die sukzessive Entstehung dieser mit dokumentiert werden konnte. Der aktuelle Stand der Diplomarbeit war dadurch jederzeit für alle Beteiligten und Interessierten über das Intranet der gedas, VW, Audi und Porsche verfügbar. Des Weiteren wurde angeeignetes Wissen, Erkenntnisse und verschiedene Lösungsansätze strukturiert abgelegt, was bei der Erstellung der Reinschrift sehr hilfreich war. Gedanken, Ergebnisse

[1] Nähere Informationen zu ZOPE: `http://www.zope.org/`.

und Informationen, auf die zu einem späteren Zeitpunkt zugegriffen werden sollte, konnten einheitlich festgehalten werden – dadurch gingen keine Informationen verloren.

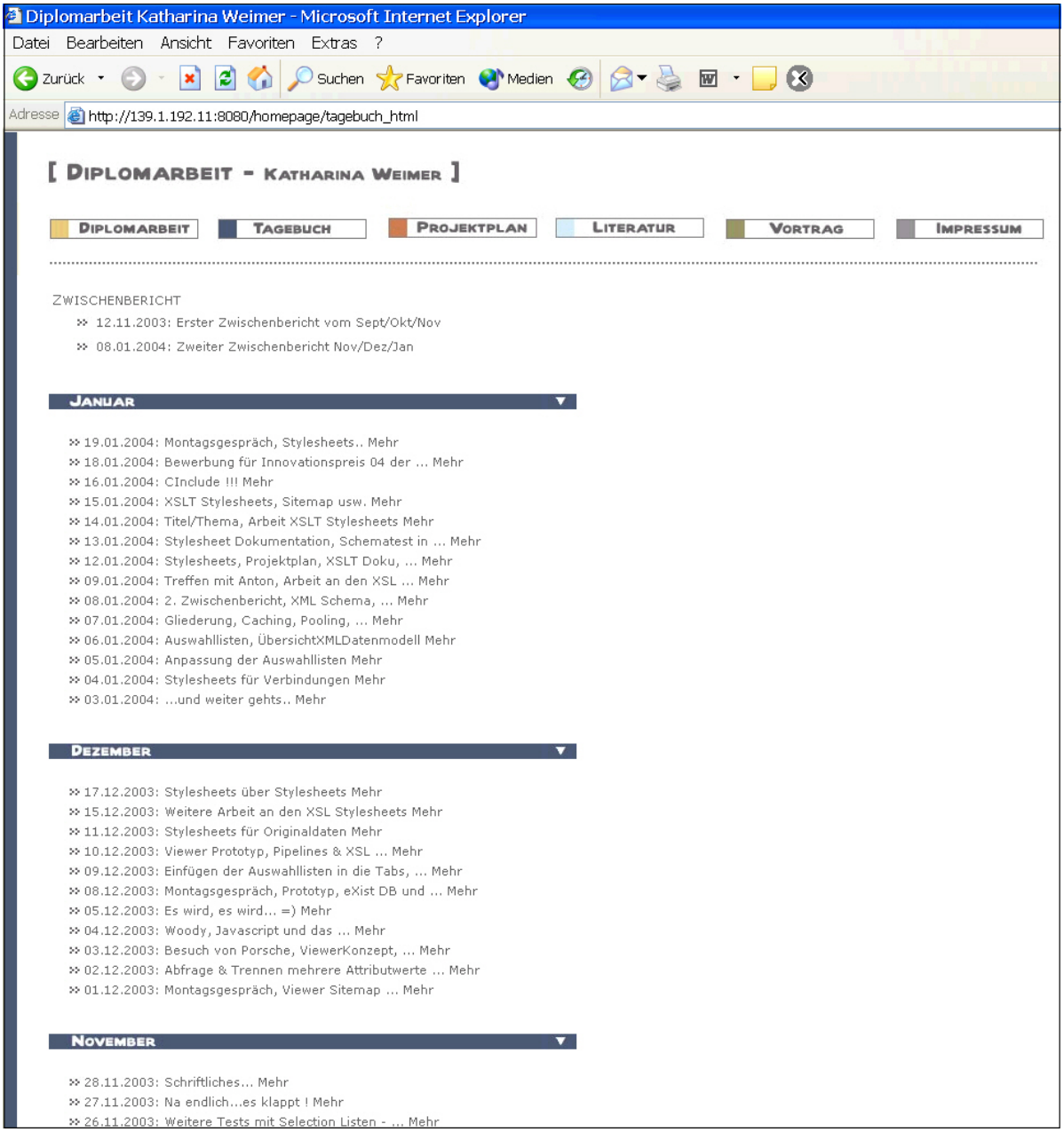

Abbildung 7.1: Tagebuchseite des Diplomarbeit-Web-Tagebuches

8 Glossar

Apache Cocoon Apache Cocoon ist ein Web Development Framework, welches zum Einen das Konzept der Trennung von Inhalt, Gestaltung und Logik und zum Anderen das Konzept der komponentenbasierten Webentwicklung umsetzt.

Apache Tomcat Der Apache Tomcat ist ein Servlet Container, welcher in der offiziellen Referenzimplementatierung für die Java Servlet- und Java Server Pages Technologien verwendet wird.

API Application Programming Interface.

ASCII American Standard Code of Information Interchange. Buchstaben müssen für PCs in Zahlen umgesetzt werden. Diese Aufgabe erfüllt ASCII, indem es 128 Zeichen auf 128 Zahlen kodiert (0-127, sieben Bit). Der erweiterte ASCII-Standard mit acht Bit bietet 256 Zeichen Platz und enthält außer englischen z.B. auch deutsche Sonderzeichen wie Umlaute.

Browser Software, die Web-Seiten visuell darstellt. Bekannte Browser sind der Internet Explorer, der Netscape Navigator, Mozilla, iCab (Macintosh) oder Konqueror (Linux).

Client Clients (Kunden) sind die Benutzer, die Informationen anfordern. Client-Programme sind die Programme, mit denen die Benutzer von ihren eigenen Rechnern (PCs) aus auf die Informationen, die auf den Servern gespeichert sind, zugreifen. WWW-Client-Programme werden gemeinhin auch als Web-Browser bezeichnet.

CSS Casading Style Sheets. CSS ist ein vom W3-Consortium definiertes, einfaches Format für Stylesheets für die Darstellung von HTML- und XML-Dokumenten.

DTD Document Type Definition. Eine DTD beschreibt die Struktur einer Klasse von SGML- oder XML-Dokumenten, also einer SGML- oder XML-Applikation, mit Hilfe eines Text-Files, das alle Syntax-Regeln in einem von SGML vorgeschriebenen Format enthält.

Beispielsweise ist jede HTML-Version durch eine DTD definiert. Eine Alternative dazu ist die Definition mit Hilfe eines Schemas.

DOM Document Object Model. DOM ist ein Objektmodell, es beschreibt die in einem Dokument einer bestimmten XML-Anwendung enthaltenen Elemente als Objekte, für die Verarbeitung mit einer objekt-orientierten Programmiersprache wie z.B. Java. DOM liefert eine komplette Baumstruktur aller Objekte eines XML-Dokuments.

eXist eXist ist eine Open Source native XML Datenbank, ausgestattet mit einem effizienten, Index-basierten XPath Prozessor, Erweiterungen für die Keywortsuche, XUpdate-Unterstützung und enge Integration mit existierenden XML Entwicklungswerkzeugen. Die Datenbank ist komplett in Java geschrieben und kann einfach in verschiedener Art und Weise eingesetzt werden. Einerseits kann sie als eigenständiger Serverprozess innerhalb eines Servlet Containers laufen, oder direkt eingebettet in einer Applikation.

Framework Ein Framework ist eine Menge von kooperierenden Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsbereich implementieren.

GUI Über ein Graphisches User Interface (GUI) wird es einem Anwender ermöglicht, über einen mehr oder weniger „Bildorientierten Weg“ mit einer Technologie zu interagieren.

HTML Hypertext Markup Language. Bei HTML handelt es sich um eine Sprache, die mit Hilfe von SGML (Standard Generalized Markup Language) definiert wird. Mittlerweile gibt es einen Ableger von HTML namens XHTML. In der Sprachversion 1.0 ist XHTML eine Redefinition von HTML mit Hilfe von XML. HTML ist eine so genannte Auszeichnungssprache (Markup Language) und hat die Aufgabe, die logischen Bestandteile eines textorientierten Dokuments zu beschreiben (wie Überschriften, Textabsätze, Listen, Tabellen oder Grafikreferenzen). HTML-Files können mit einfachen Text-Editoren oder mit speziellen Hilfsprogrammen erstellt oder aus bestehenden Dokumenten oder Datenbanken generiert werden.

HTTP Hypertext Transfer Protocol. Http ist das Protokoll, nach dem Informationen zwischen WWW-Servern und WWW-Clients über das Internet übertragen werden.

JAXP Java API for XML Parsing. Einheitliches API, um unter Java auf XML zuzugreifen. Umfasst DOM, SAX und XSLT und kann verschiedene XML-Parser einbinden. JAXP ist in J2SE 1.4 enthalten.

JDOM Java Document Object Model. JDOM ist eine Java-Bibliothek, die eine an Java angepasste Programmierschnittstelle bietet und einen an Java angepassten Objekt-Tree aus dem XML-Dokument erstellt.

J2EE Java 2 Platform, Enterprise Edition, abgekürzt J2EE, ist ein Standard, um mit modularen Komponenten verteilte, mehrschichtige Anwendungen zu entwickeln. J2EE setzt auf bereits etablierte Standards wie z.B. JDBC oder CORBA auf und ermöglicht dem Entwickler den Zugriff auf weitere Funktionalitäten wie z.B. Enterprise Java Beans, Java Servlets, JSP und XML. J2EE ermöglicht die Erstellung von skalierbaren und plattformunabhängigen Anwendungen, mit der Möglichkeit auch auf bestehende (Alt-) Systeme zuzugreifen.

SAX Simple API for XML. SAX ist eine Programm-Schnittstelle (Application Programmers Interface API) für die Verarbeitung einer Klasse von XML-Dokumenten, also einer XML-Applikation, mit Hilfe einer objekt-orientierten Programmiersprache wie z.B. Java. SAX liefert ein XML-Element nach dem anderen in einem Eingabestrom und eignet sich daher auch für die Bearbeitung großer XML-Dateien.

SiXDML Simple XML Data Manipulation Language. Nähere Informationen:`http://sixdml.sourceforge.net/`.

SGML Standard Generalized Markup Language. SMGL ist ein Standard für Auszeichnungssprachen für digitale Dokumente. SGML strukturiert und kennzeichnet Inhaltselemente eines Dokuments in Form von Text. Mit Hilfe von festgelegten Markup-Anweisungen (Markierungen wie zum Beispiel Title, Chapter, Footnote) kann ein Dokument strukturiert und dessen Inhaltselemente (Text, Bilder, Tabellen) bezeichnet werden. SGML selbst ist keine Auszeichnungssprache, sondern eine generelle "Grammatik", aus der eine konkrete Auszeichnungssprache mit Hilfe einer "Document Type Definition"(DTD) abgeleitet wird. Die bekannteste DTD von SGML ist HTML.

PDF Portable Document Format. Bezeichnet ein Dokumentformat von Adobe, mit dessen Hilfe Dokumente jeglicher Art (hauptsächlich Handbücher, Prospekte, Statistiken usw.) auf allen möglichen Plattformen elektronisch veröffentlicht werden können.

Stylesheets Stylesheets bieten eine gute Möglichkeit, die Darstellung des Inhalts von Web-Seiten in einem einheitlichen und konsistenten Layout zu bewirken.

URI Der URI (Uniform Resource Identifier, dt: einheitlicher Ressourcen-Identifikator) dient zur eineindeutigen Identifizierung eines bestimmten Inhalts - Text, Grafik, Animation oder Audio, die "Ressource im Internet.

URL Uniform Resource Locator. URL ist die „Adresse“, die eine Browser benötigt, um eine bestimmte Information vom jeweiligen Server zu erhalten. Der URL enthält zu diesem Zweck Informationen wie die Art des Zugriffs (Protokoll), die Adresse des Server, eventuell mit einem Username und Paßwort oder einer Port-Nummer, und das Directory und den Filenamen der Datei, in der die gewünschte Information gespeichert ist.

WEBDAV Web-based Distributed Authoring and Versioning. Nähere Informationen: `http://www.webdav.org/`.

WML Wireless Markup Language. WML ist ein Gegenstück zu HTML für die Darstellung von Informationen auf Mobil-Telefonen (Handys) und auf anderen Geräten mit kleinen Displays. WML baut nicht auf HTML sondern auf XML auf, ist also ähnlich wie HTML, aber nicht mit HTML kompatibel. WML-Files werden wie HTML-Files auf Web-Servern gespeichert, der Zugriff erfolgt über ein WAP-Gateway mit dem Protokoll WAP.

W3C World Wide Web Consortium. Dieses von verschiedenen Verbänden und Firmen mitbegründete, unabhängige Konsortium macht Vorschläge zu Sprachstandards und Technologien im Internet. Diese werden in Form von Spezifikationen veröffentlicht.

XLink XML Linking Language. Ein XML-Vokabular für Elemente, die in XML-Dokumente eingefügt werden können, um Links zu Ressourcen aufzubauen und zu beschreiben.

XML Die Extensible Markup Language (XML) ist eine Sprache zur Speicherung von Daten. In einer XML-Datei kann mittels einer Definition, einer DTD oder eines Schemas, festgelegt werden, in welcher Form die Daten gespeichert werden können. Dies kann z.B. beim Eingeben von Daten durch entprechende Editoren überpüft werden.

XML:DB API Die XML:DB API definiert einen Satz allgemein gültiger Zugangsmechanismen, um auf XML Datenbanken zuzugreifen. Eine allgemein gültige API für native XML Datenbanken zu haben, ist aus demselben Grund für Anwender attraktiv, aus dem auch JDBC für Anwender relationaler Datenbank ist: eine allgemein gültige API erleichtert die Entwicklung von Applikationen, die unterschiedliche Datenbanken unterstützen, einfacher.

XML-Parser Ein XML-Parser ist ein Programm, das ein XML-File liest und den Inhalt in der Form von DOM oder SAX liefert. Ein validierender Parser überprüft zusätzlich die Richtigkeit der Daten an Hand der DTD oder des Schemas.

XML Remote Procdure Call XML-RPC (XML Remote Procedure Call) ist eine Spezifikation, die es Software auf verschiedenen Systemen und unter verschiedenen Umgebungen erlaubt, miteinander über das Internet zu kommunizieren. Nähere Informationen: `http://www.xmlrpc.com/`.

XML Schema Ein Schema beschreibt die Struktur einer Klasse von XML-Dokumenten, also einer XML-Applikation, ähnlich wie eine DTD, jedoch nicht in der DTD-Syntax sondern in einer eigenen XML-Syntax.

XPointer XML Pointer Language. XPointer wird auf der Basis von XPath-Ausdrücken eingesetzt, um über entsprechend erweiterte URI-Verweise Fragmente in Web-Ressourcen zu identifizieren, etwa einzelne Elemente oder auch Teile einer Zeichenkette, die über Positionsangaben angesteuert werden können.

XSL Extensible Style Language.Mit XSL wird ein Style-Sheet definiert, das angibt, wie der in einem XML-Dokument definierte Inhalt vom Web-Browser oder von anderen Programmen dargestellt werden soll. XSL ist mächtiger als CSS und DHTML:

- XSLT (Transformation) bietet die Möglichkeit, aus einem XML-File ein anderes XML-File erzeuge, also z.B. bestimmte Elemente wegzulassen, die Elemente in anderen Reihenfolgen anzuordnen und zusätzliche Elemente hinzuzufügen,
- und mit XSL-FO (Formatierung) das Layout der Darstellung für die Elemente festzulegen.

XHTML (Extensible Hypertext Markup Language) Mit XHTML wird ein HTML-File bezeichnet, das den strengeren Syntax-Regeln von XML entspricht und deshalb besser von Computer-Programmen weiterverarbeitet werden kann. (XHTML 1.0 entspricht dem Funktionsumfang von HTML 4.0)

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[MOZI2004] Mozilla Organisation: *What's New in Mozilla 1.7 Alpha*, URL: `http://www.mozilla.org/releases/mozilla1.7a/README.html`, Datum des Zugriffs: 16.01.2004.

[ABM2000a] Anderson R., Birbeck M., Kay M. u.a.: *XML Professional, Kapitel 6: SAX 1.0: Die Simple API für XML*, 1.Auflage, MITP Verlag GmbH Bonn, 2000, Seite 197ff.

[ABM2000b] Anderson R., Birbeck M., Kay M. u.a.: *XML Professional, Kapitel 4: Datenmodellierung in XML*, 1.Auflage, MITP Verlag GmbH Bonn, 2000, Seite 123ff.

[APAC2003a] Apache Software Foundation: *Apache Xindice*, URL: `http://xml.apache.org/xindice/`, Datum des Zugriffs: 05.10.2003.

[BEMI2004] Behme, Henning und Mintert, Stefan: *XML und Apache*, Kapitel 18.2.1 Extensible Server Pages, URL: `http://www.linkwerk.com/pub/xmlidp/2000/unterabschnitt22461.html`, Datum des Zugriffs: 22.04.2004.

[BOUR2004a] Bourret, Ronald: *XML Database Products*, URL: `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`, Datum des Zugriffs: 02.04.2004.

[BOUR2004b] Bourret, Ronald: *XML and Databases*, URL: `http://www.rpbourret.com/xml/XMLAndDatabases.htm`, Datum des Zugriffs: 02.04.2004.

[BURK2001] Burke, Eric M.: *Java and XSLT*, 1. Auflage, September 2001, Kapitel 5.

[COCO2003a] Cocoon Wiki: *Woody*, URL: `http://wiki.cocoondev.org/Wiki.jsp?page=Woody`, Datum des Zugriffs: 03.11.2003.

[COCO2003b] Cocoon, Apache Software Foundation: *Apache Cocoon Performance Tips*, URL: `http://cocoon.apache.org/2.0/performancetips.html`, Datum des Zugriffs: 07.11.2003.

[DIA2004] Deutsche Informatik Akademie: *Web-Performance: Metriken, Modelle und Methoden für Planung und Test von E-Business-Websites* , URL: `http://www.dia-bonn.de/web2_2004/web.html`, Datum des Zugriffs: 05.04.2004.

[FHWE2000] Fachhochschule Wedel: *XML und Java*, Ausarbeitung des Informatik-Seminars, Wintersemester 2000, URL:`http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/5.xslt/xslt1.htm`, Datum des Zugriffs: 20.02.2004.

[FIWO] Fitschen, Arne und Lezius, Wolfgang: *Sonnige Aussichten, Einführung in XML und XSLT, Teil 2; Ein starkes Team?*, XML Magazin, ohne Jahrgang, URL:`http://www.ims.uni-stuttgart.de/projekte/corplex/paper/lezius/EinStarkesTeam2.pdf`, Datum des Zugriffs: 05.02.2004.

[FRAU2004] Fraunhofer Institut für integrierte Publikations- und Informationssysteme: *PDOM und XQL-Prozessor, der XML-Datenserver*, URL: `http://www.ipsi.fraunhofer.de/oasys/projects/pdom/index_d.html`, Datum des Zugriffs: 02.04.2004.

[FRFD2004] Fraunhofer Institut für integrierte Publikations- und Informationssysteme und FH Darmstadt: *Das W3C*, FH Darmstadt, Sommersemester 2004, URL: `http://www.ipsi.fraunhofer.de/~putz/FHD/XML/PDFs_2004/W3CundSprachen.pdf`, Datum des Zugriffs: 02.03.2004.

[FRIE2003] Frieb Werner: *XML Databases compared*, Institute for Software Technology and Interactive Systems, Vienna University of Technology, URL: `http://www.studierstube.org/world/xml_databases_compared.html`, Datum des Zugriffs: 20.12.2003.

[GART2002] Gartner: *Gartner: J2EE und .NET bergen Risiken*, in: Tecchannel, 18.11.2002, URL: `http://www.tecchannel.de/news/allgemein/10372/`, Datum des Zugriffs: 16.12.2003.

[GVEM2001] Grigor, A., Valentini, C., Eysoldt, K., Mikosch, O.: *Vortrag über die Bordnetzentwicklung*, Fachhochschule Braunschweig/Wolfenbüttel, 04.12.2001.

[GULP2004] GULP Information Services GmbH: *GULP Trend Analyzer*, URL: http://www.gulp.de/cgi-gulp/trendneu.exe/SSEEK#Trend, Datum des Zugriffs: 17.04.2004.

[HOEF2001] Höfling, Jürgen: *Performance ist geschäftskritisch*, in: Informationweek, Ausgabe 8, 5. April 2001, URL: http://www.informationweek.de/index.php3?/channels/channel20/010834.htm, Datum des Zugriffs: 02.04.2004.

[LANE2001] Lane, Eoin: *Add XML to your J2EE applications*, in: JavaWorld, Februar 2001, URL: http://www.javaworld.com/javaworld/jw-02-2001/jw-0209-xmlj2ee.html, Datum des Zugriffs: 15.04.2004.

[LIAN2002] Liantis IT COnsulting: *Moderne Konzepte und IT-Architekturen in der Softwareentwicklung*, Forschungstag der Universität Wuppertal, 21.09.2002, URL: http://www.liantis.com/Downloads/Moderne_Konzepte_IT-Architekturen_Softwareentwicklung_Uni_Wuppertal.pdf, Datum des Zugriffs: 16.04.2004.

[MART2001] Martin, Lars: *Entwicklung neuer Standards für XML-Datenbanken - Teamgeist!*, Linux-Magazin, April 2001, URL: http://www.linux-magazin.de/Artikel/ausgabe/2001/04/xmldb/xmldb.html, Datum des Zugriffs: 27.09.2003.

[MCLA2000] McLaughlin, Brett: *Java and XML*, O'Reilly, Juni 2000, Kapitel 9.

[MCLA2001] McLaughlin, Brett: *Einführung in XML*, O'Reilly Verlag 2001, URL: http://www.oreilly.de/artikel/xml_einf.html, Datum des Zugriffs: 02.02.2004.

[MEBO2002] Mercay, Julien und Bouzeid, Gilbert: *Boost Struts with XSLT and XML*, Java World, Februar 2002, URL: http://www.javaworld.com/javaworld/jw-02-2002/jw-0201-strutsxslt.html, Datum des Zugriffs: 03.02.2004.

[MEIE2003a] Meier, Wolfgang: *eXist – Feature Overview*, URL: http://exist.sourceforge.net/features.html, Datum des Zugriffs: 01.10.2003.

[MEIE2003b] Meier, Wolfgang: *eXist Open Sorce Database*, URL: http://exist.sourceforge.net/, Datum des Zugriffs: 01.10.2003.

[MEIE2003c] Meier, Wolfgang: *eXist: An Open Source Database*, Darmstadt University of Technology, URL: http://exist-db.org/webdb.pdf, Datum des Zugriffs: 15.10.2003.

[MEIE2003d] Meier, Wolfgang: *eXist – Developers Guide*, URL: http://exist.sourceforge.net/devguide.html, Datum des Zugriffs: 03.10.2003.

[MEIE2003e] Meier, Wolfgang: *eXist – Server Configuration*, URL: http://exist.sourceforge.net/configuration.html, Datum des Zugriffs: 03.10.2003.

[MITT2003] Mittermeier, Ludwig: *XML und Datenbanken, naiv nativ*, in: iX, 2003, Heft-Nr. 8, Seite 42 ff.

[MSH2002] Middendorf, Stefan; Singer, Reiner; Heid, Jörn: *Programmierhandbuch und Referenz für die Java-2-Plattform, Standard Edition*, dpunkt.Verlag, Heidelberg, 2002, URL: http://www.dpunkt.de/java/Programmieren_mit_Java/XML/11.html, Datum des Zugriffs: 02.04.2004.

[NAMI2002] namics ag: *.NET vs. J2EE In welche Plattform investieren?*, namics Whitepaper, August 2002, URL: http://www2.namics.com/files/kno_whi_net.pdf, Datum des Zugriffs: 15.04.2004.

[PART2000] Partl, Hubert: *XML - Extensible Markup Language*, September 2000, URL:http://www.boku.ac.at/htmleinf/xmlkurz.html#applikationen, Datum des Zugriffs: 20.03.2004.

[SCHM2003a] Schmiderer, Johannes: *Webapplikationen mit Unterstützung von XML*, Kapitel 2.3 Verteilte Computersysteme, URL: http://www.schmiderer.cc/xmlwebapp/chapter_2_3.php, Datum des Zugriffs: 12.11.2003.

[SCHM2003b] Schmiderer, Johannes: *Webapplikationen mit Unterstützung von XML*, Kapitel 3.5 Sicherheitsrisiken bei Webapplikationen, URL: http://www.schmiderer.cc/xmlwebapp/chapter_3_5.php, Datum des Zugriffs: 12.11.2003.

[SEFF2003] Hendrik Seffler: *XML-Datenbank Exist*, HU Berlin, URL: http://www.informatik.hu-berlin.de/~seffler/2-ausarbeitung.pdf, Datum des Zugriffs: 15.10.2003.

[SUNM2004] Sun Microsystems Inc.: *Designing Enterprise Applications with the J2EE Platform*, URL: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/index.html, Datum des Zugriffs: 29.03.2004.

[TECC2001] Tecchannel: *Server- oder Client-seitige Verarbeitung*, Tecchannel, 13.03.2001, URL: `http://www.tecchannel.de/internet/670/2.html`, Datum des Zugriffs: 02.03.2004.

[UNIV2003] Universität Stuttgart: *Softwareentwicklungsprozesse und Vorgehensmodelle*, Lehrmaterial zur Vorlesung Softwaretechnik I, 2003, URL:`www.ias.uni-stuttgart.de/st1/lehrmaterialien/umdruck/kap003.pdf`, Datum des Zugriffs 26.02.2004.

[VIRA2002] Virag, Anton: *VW und gedas entwickeln Lösung für virtuelles Bordnetzdesign*, in: Automotive Engineering Partners, Februar 2001, Ausgabe 02.

[VOCK2003] Vockeroth, Johannes: *Das XML Publishing Framework „Apache Cocoon“*, Ausarbeitung im Rahmen des Proseminars Multimediatechnik, Fakultät Informatik TU Dresden, Januar 2003, URL: `http://www-mmt.inf.tu-dresden.de/lehre/Wintersemester_02_03/Proseminar/JohannesVockeroth/ausarbeitung.pdf`, Datum des Zugriffs: 24.10.2003.

[WILK2003] Wilcox, Mark: *Using Embedded XML Databases to Process Large Documents*, 22. Oktober 2003, URL: `http://www.xml.com/lpt/a/2003/10/22/embed.html`, Datum des Zugriffs: 06.11.2003.

[WINK2004] Winkler, Christoph: *Open Source Projekte im XML/JAVA-Umfeld*, Zentrum der Medizinischen Informatik, Uniklinikum Frankfurt, URL: `http://www.mug-d.de/download/ft_2001/open_source_xml.pdf`, Datum des Zugriffs: 01.03.2004.

[XMLM2003] XML Magazin und Web Services: *Kurzschreibweise, Tipps zur Verkürzung von XSLT-Skripten*, in: XML Magazin und Web Services, 2003, Heft-Nr. 03, Seite 39 - 41.

[ZILA2002a] Ziegeler, Carsten und Langham, Matthew: *Cocoon: Building XML Applications*, New Riders, 1. Juli 2002, .

[ZILA2002b] Ziegeler, Carsten und Langham, Matthew: *Cocoon – aber sicher! Eine Einführung in personalisiertes XML Publishing mit Apache Cocoon*, in: XML Magazin und Webservices, 2002, Ausgabe 3, Seite 20ff.