Diplomarbeit

Peer Hartmann

Erstellung eines Systems zur Integration
HTML-basierter Dienste

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Peer Hartmann

Erstellung eines Systems zur Integration HTML-basierter Dienste

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Helga Carls

Abgegeben am 19.Juli 2005

Peer Hartmann

Thema der Diplomarbeit

Erstellung eines Systems zur Integration HTML-basierter Dienste

Stichworte

HTML HTTP Kommunikation Extraktion WWW Parsing Analyse Integration

Kurzzusammenfassung

Das World Wide Web stellt derzeit die größte Sammlung an elektronisch verfügbaren Informationen dar. Insbesondere Dienste, die individuelle Informationen wie persönlich angepaßte Fahrpläne anbieten, könnten in darauf aufbauenden Computersystemen sinnvoll genutzt werden. Mit dieser Arbeit soll gezeigt werden, wie die Integration eines solchen Dienstes in ein darauf aufbauendes Computersystem realisiert werden kann. Dazu werden zunächst die Probleme und Hindernisse, die sich durch die Nutzung des World Wide Web durch ein Computersystem ergeben, identifiziert. Aus den dabei gewonnen Erkenntnissen soll ein prototypischer Adapter für einen bestehenden Dienst erstellt und getestet werden.

Ende des Textes

Peer Hartmann

Title of the paper

Creation of a System for Integrating HTML-Based Services

Keywords

HTML HTTP Communication Extraction WWW Parsing Analysis Integration

Abstract

Today the World Wide Web exhibits the widest collecton of information available in an electronic form. Some of the
information available, especially those offered by services for individual queries, like rail enquiries, could in a
reasonable manner be used by computer systems. The intention of this paper is to find a way to realize such an
integration of HTML-based services.

End of text

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Zuge der Erweiterung des von Gunnar Steffen [Ste04] entwickelten und von der Firma portrix.net GmbH umgesetzten Mitfahrclubs, eines Systems zur Optimierung der Nutzung von Kraftfahrzeugen, entstand die Idee, dem Nutzer des Systems auch Verkehrsmittel des ÖPNV anzubieten. Hierzu müssen Daten über die Fahrten (Fahrpläne, Fahrstrecken, Preise usw.) beschafft werden. Aufgrund der Tatsache, daß das System an möglichst vielen verschiedenen Stellen eingesetzt werden soll, ist es notwendig, auf einen möglichst breiten Pool von Informationsquellen zugreifen zu können. Zum Zeitpunkt der Erstellung dieser Arbeit stellt dies das World Wide Web (WWW, siehe Kapitel 2.1) dar. Viele Anbieter von Transportdienstleistungen stellen – wenn sie überhaupt Informationen in elektronischer Form anbieten – diese zumindest über das WWW zur Verfügung. Aus dieser Überlegung entstand die Idee, das Mitfahrclub-System mit Daten von Diensten mit einer Schnittstelle zum WWW zu versorgen.

Es soll in der folgenden Ausarbeitung ermittelt werden, ob es möglich ist, Daten aus dem HTML-basierten WWW zu extrahieren und in einem darauf aufbauenden System zu verarbeiten. Dieser Prozeß muß unmittelbar auf Verlangen (on demand) verfügbar sein, um ein laufendes System mit dem zur Weiterverarbeitung benötigten Material zu versorgen.

1.1 Die Herausforderung

Die Erstellung eines Systems zur Extraktion von Daten aus dem WWW ist Thema dieser Ausarbeitung. Die Herausforderung besteht vor allem darin, daß ein für die Bedienung durch Menschen geschaffenes Medium (dem WWW) von einem Computersystem benutzt wird, um Daten zu beschaffen. Über die Jahre, in denen sich das WWW entwickelt hat[1] wurde auf eine solche Benutzung durch Maschinen keine oder nur sehr wenig beachtet. Das fängt schon bei der Sprache, in der Informationen im WWW üblicherweise publiziert werden, dem HTML (siehe Kapitel 2.3), an. Diese ist zur Beschreibung von Dokumenten konzipiert worden. Dadurch fehlen einem HTML-Dokument im Gegensatz zu anderen Datenaustauschformaten,

[1] Hier kann kaum von einer zielgerichteten Entwicklung gesprochen werden, da das WWW von zu vielen Parteien in zu viele unterschiedliche Richtungen gedrängt wurde

wie etwa XML, spezifische Informationen über Typ und Format der darin befindlichen Daten. Heute wird HTML dazu eingesetzt, um Daten in einem bestimmten Layout darzustellen. Das führt dazu, daß viele unwichtige Daten, die einer anderen Art der Verarbeitung als der Anzeige dienen, in einem HTML-Dokument enthalten sind.

Es gibt bereits technische Lösungsansätze für das hier betrachtete Problem (siehe Kapitel 5). Diese allerdings sollen in Zukunft aufgrund der Entwicklung hin zu einer serviceorientierten Architektur nicht mehr notwendig sein. Genaueres hierzu findet sich in Kapitel 2.2. Eine weitere intressante Entwicklung stellt die Arbeit an der Erstellung des sogenannten semantischen Webs dar. Hier wird versucht, den Inhalt beliebiger Seiten beziehungsweise Dokumente mit dem Computer zu verstehen, um so genauere Informationen darüber zu erhalten, was in den Dokumenten beschrieben wird. Genaueres hierzu findet sich auf [SD] und in Kapitel 3.5. Diese Ansätze werden nur oberflächlich als Vergleichsszenario behandelt.

Zusammengefaßt besteht die Herausforderung darin, relevante Daten aus einem fremden System zu beschaffen, diese zu interpretieren und die benötigten Teile in eine für das Zielsystem verständliche Form zu übersetzen.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, herauszufinden, ob und wie es gegebenenfalls möglich ist, Daten aus einem Dienst im WWW zu extrahieren und in einem Computersystem weiterzuverarbeiten. Zur Klärung dieser Frage soll ein prototypisches System erstellt werden,

- welches Daten von einem HTML-basierten Dienstanbieter aus dem WWW extrahiert
- und einer anderen Applikation (dem Mitfahrclub) zur Verfügung stellt.

Da es in der Regel mehrere Dienstanbieter für dieselbe Dienstleistung gibt, soll das System so gestaltet werden, daß auch ohne Programmieraufwand weitere Dienste derselben Art eingebunden werden können. Auf diesem Weg wird erreicht, daß die Pflege des Systems von möglichst vielen Benutzern durchgeführt werden kann.

Das WWW stellt die verschiedensten Informationen zur Verfügung, daher soll die Lösung so gestaltet werden, daß sie sich auch in andere Softwareumfelder als Komponente integrieren läßt.

1.3 Aufbau dieser Arbeit

Diese Arbeit ist in sieben Kapitel unterteilt. Im zweiten Kapitel wird ein Überblick über die verwendeten Technologien gezeigt. Im daran anschließenden Analyse-Kapitel werden Anforderungen an ein System zur Integration HTML-basierter Dienste erarbeitet. Aus diesen wird in Kapitel 4 ein allgemeiner Designvorschlag erstellt. Diesem Designvorschlag und den Anforderungen aus dem Analysekapitel werden in Kapitel 5 zwei Lösungsansätze gegenübergestellt und auf Benutzbarkeit überprüft. Das Kapitel 6 befaßt sich mit einem selbst entwickelten Lösungsansatz. Dieser wird dort vorgestellt und auch bewertet. Den Abschluß der Arbeit stellt das Kapitel 'Ausblick und Fazit' dar.

2 Grundlagen

In den vergangenen Jahren hat sich das Internet als breiteste Quelle für Informationen etabliert. Die Form in welcher die Informationen abgefragt und geliefert werden ist allerdings keinesfalls einheitlich. Aus diesem Grunde ist es notwendig einige grundlegende Technologien vorzustellen und zu analysieren.

2.1 Das WWW

Das WWW wurde 1990 auf der ersten, über das HTTP-Protokoll erreichbaren, Seite wie folgt definiert: ”The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents. ” [BL, www-intro]

Das WWW stellt die populärste Art von Informationsangeboten im Internet dar. Heute ist es insbesondere für Firmen, die sich dem Massenmarkt widmen, immer wichtiger, Informationen über ihre Angebote im WWW zur Verfügung zu stellen. Diese Informationen können zum einen statisch, also etwa in Form einer einfachen Produktbeschreibung, als auch dynamisch, zum Beispiel als Individual-Fahrpläne für bestimmte Fahrten, vorliegen.

Ursprünglich war das WWW als Netz von untereinander verlinkten Dokumenten geplant. Es wurde hierzu eine Dokumentenbeschreibungssprache (HTML) entworfen. Als Abfrageprotokoll wurde das HTTP gewählt.

Heute ist das WWW weit mehr als ein "großes Universum von Dokumenten". Es ist vielmehr eine Präsentations-, Informations- und Werbeplattform von bislang unbekannter Komplexität und Interaktivität geworden.

2.2 Web Services

”Today, the principal use of the World Wide Web is for interactive access to documents and applications. In almost all cases, such access is by human users, typically working through Web browsers, audio players, or other interactive front-end systems. The Web can grow

significantly in power and scope if it is extended to support communication between applications, from one program to another. The purpose of this Working Group is to create a simple foundation to support the needs of such communicating applications.” [DF]

Aus dieser – vom W3C beschriebenen – Überlegung sind die Web Services entstanden. Im Gegensatz zum WWW stellen diese kein zusammenhängendes, über Hyperlinks verbundenes System dar, sondern werden explizit für einzelne Aufgaben angeboten. Der große Unterschied zum WWW besteht darin, daß ein Webservice über eine umfangreichere Schnittstelle in beide Richtungen verfügt. Es wird, aufbauend auf anderen niedriger angeordneten Protokollen, ein Dienst mit klar definierten Ein- und Ausgabeparametern angeboten.

In [Cor] heißt es weiterhin: ”To undergo this transition, machines must turn from passive clients and servers of information into agents actively exchanging and managing information on behalf of their owners.” Das beschreibt recht passend, was der Mitfahrclub werden soll: ein Agent, der für den User ein Angebot aus verschiedenen Quellen erstellt.

Das W3C gibt einige Vorgaben, wie ein Webservice aufzubauen ist. So wird beschrieben, wie diese Services spezifiziert werden und wie die Abfrage und Antwort-Protokolle funktionieren. In dieser Ausarbeitung wird Webservice jedoch nicht nach dieser Definition betrachtet, sondern vielmehr als ein Synonym für einen über das Internet verfügbaren Service. Web Services werden daher als Kontrast zum unstrukturierten WWW begriffen.

Im Gegensatz zum WWW wird vom W3C XML als Format zum Datenaustausch für Web Services vorgeschlagen. XML steht für eXtensible Markup Language. Das bedeutet, daß die Sprache erweiterbar ist. HTML ist dies – zumindest weitestgehend – nicht. XML gibt dem Entwickler die Möglichkeit, eigene Datentypen und Datenstrukturen zu definieren. Im Gegensatz zur Dokumentenbeschreibungssprache HTML ist XML eine universelle Beschreibungssprache und von daher für den elektronischen Austausch von Daten mit beliebigen, aber eindeutigen Strukturen besser geeignet. Die in dieser Arbeit aufgezeigten Schwachpunkte bei der Datenextraktion können durch die Verwendung von XML vermieden werden.

2.2.1 DELFI

Ein Beispiel für einen Webservice, der Dienste anbietet, die für die Mitfahrbörse von Interesse sein könnte ist das System für ”Durchgängige ELektronische FahrplanInformation” kurz DELFI. Dieses System wurde vom Bundesministerium für Verkehr initiiert und soll den Zugang zu öffentlichen Verkehrsmitteln erleichtern.

Das DELFI System ist kein Webservice, wie er vom W3C spezifiziert wird. Es baut vielmehr auf CORBA auf und bietet so eine Unabhängigkeit von Plattform und Programmiersprache. Eine Einbindung dieses Dienstes soll also prinzipiell möglich sein, wird aber gleichzeitig nicht Teil dieser Ausarbeitung sein, da auch mit DELFI nicht alle Verbindungen – insbesondere im

öffentlichen Personennahverker – gefunden werden können und da auch DELFI nur einen begrenzten Einzugsbereich hat. Würde das abfragende System beispielsweise in den USA eingesetzt, so wäre nicht damit zu rechnen, daß dort ein vergleichbares, einheitliches System anzutreffen wäre. Und selbst wenn dies der Fall wäre, so ist es sehr unwahrscheinlich, daß die Schnittstelle oder auch nur die eingesetzten Technologien dieselben sind.

Weitere Informationen zu DELFI finden sich auf [Ver]

2.3 Aufbau einer HTML-Seite

HTML ist eine Seitenbeschreibungssprache, die vom 'Vater' des WWW, Tim Berners-Lee, entwickelt wurde. Sie basiert auf der SGML ISO-Spezifikation 8879. Ursprünglich wurde sie entwickelt um die wissenschaftlichen Dokumente am CERN in eine besser nutzbare Form zu bringen. Die HTML zeichnet sich vor allem durch ihre Einfachheit aus.

Grundsätzlich ist der Inhalt einer HTML-Seite mittels so genannte "Tags" strukturiert und zählt damit zu den Auszeichnungssprachen (Englisch: "mark-up languages"). Tags stellen die formatierenden Elemente einer HTML-Seite dar. Durch die große Popularität von HTML wuchsen die Anforderungen an die Sprache sehr schnell. Sie wurde um diverse Möglichkeiten der Integration von Medien und auch Möglichkeiten zur verbesserten Interaktion (Formulare) erweitert.

Eine geeignete Einführung zum Thema HTML findet sich auf [Mün].

2.4 Kommunikation im WWW

Im weiteren Verlauf dieser Arbeit wird auf Details der Kommunikation im WWW eingegangen. Diese werden hier kurz zusammengefaßt. Ein Überblick über die Kommunikation im WWW wird in Abbildung 2.1 gezeigt.

Eine Konversation im WWW beginnt typischerweise mit dem Anfordern einer Eingabe-Seite. Diese enthält ein Formular, welches ausgefüllt wird. Das Absenden des Formulars führt dazu, daß der Browser die in dem Formular eingegebenen Daten sammelt und an den im Formular angegebenen Server zurückschickt. Der Server wird diese Daten dann verarbeiten und eine Antwort auf die spezifische Anfrage zurücksenden. Abhängig von den Eingabeparametern kann dies bereits die gewünschte Information sein oder aber eine Nachfrage oder – im ungünstigsten Fall – eine Meldung, daß der Dienst die Anfrage überhaupt nicht verarbeiten kann. Die Antwort des Servers wird vom Browser aufgearbeitet und dem User dargestellt.

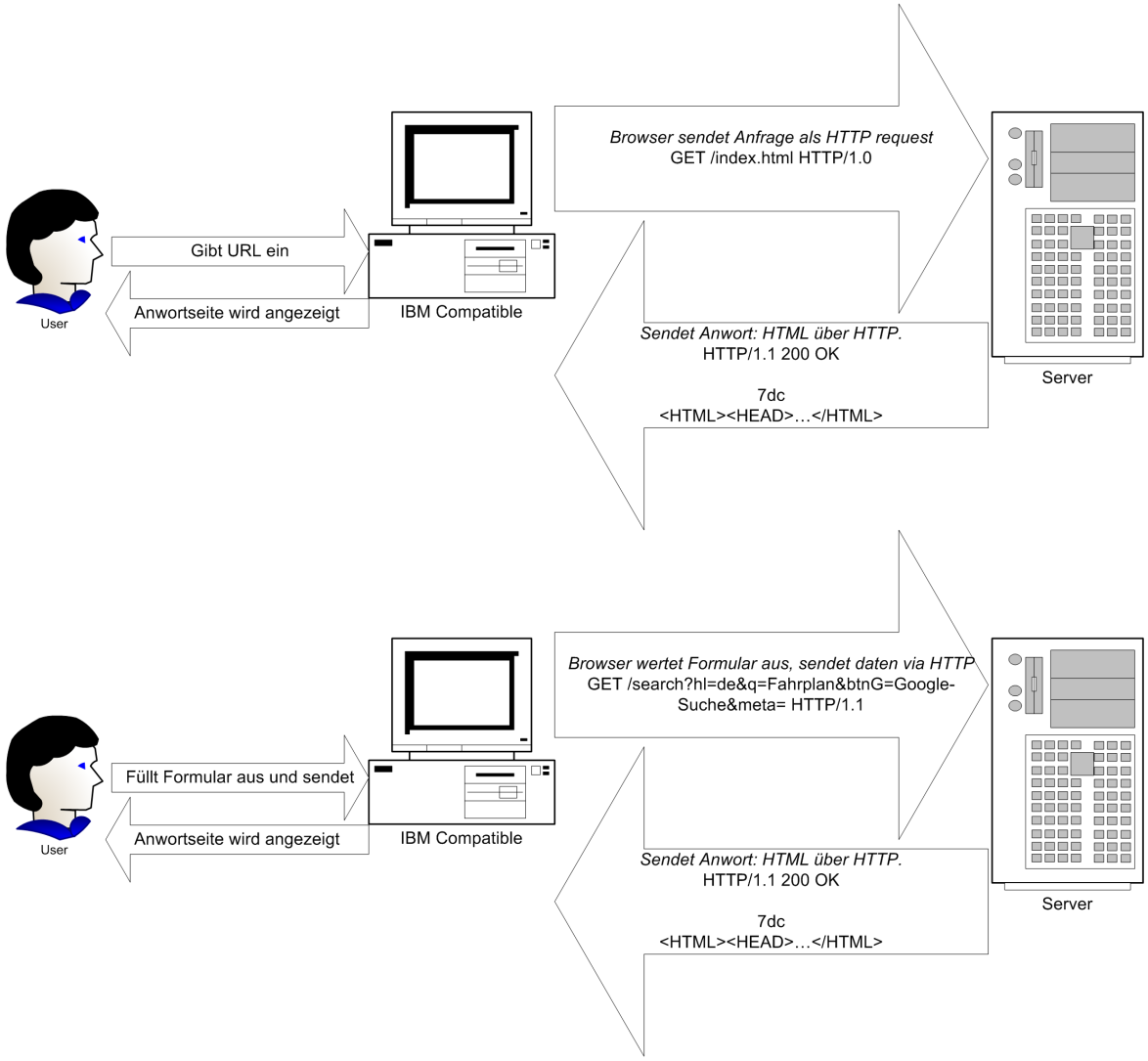

Abbildung 2.1: Schematische Darstellung einer einfachen Konversation im Internet

Abbildung 2.2: Darstellung eines Formulars im Browser

Ein Formular, wie es der Browser darstellt, wird in Abbildung 2.2 dargestellt. Der Quelltext zu diesem Formular sieht in vereinfachter Form so aus:

```
<form name="personalForm" method="post"
  action="/base/showPersonalSchedule.do">
[...]
  <input type="hidden" name="INIT" value="false">
[...]
  <input type="text" name="startName" maxlength="80" size="28"
  value="">
[...]
  <input type="text" name="startRegion" maxlength="50"
  size="10" value="">
 [...]
  <input type="submit" name="checkStart" value="Start prüfen"
  style="width: 100pt;">
[...]
  <input type="text" name="destName" maxlength="80" size="28"
   value="">
  <input type="text" name="destRegion" maxlength="50" size="10"
  value="">
[...]
  <select name="departureOrArrival" size="1">
  <option value="1" selected="selected">Abfahrt</option>
  <option value="0">Ankunft</option></select>
[...]
  <input type="text" name="startTime" maxlength="13" size="8"
  value="19:49">
[...]
  <input type="text" name="startDate" maxlength="15" size="10"
  value="21.04.2005">
[...]
  <input type="submit" name="search" value="Suche starten">
</form>
```

Diese Darstellung ist keinesfalls komplett, da die meisten nicht formularspezifischen Elemente, wie etwa Text oder Formatierungen, entfernt wurden. Eine komplette Version der dargestellten Seite findet sich auf der beiliegenden CD. Man sieht in dieser Darstellung jedoch einzelne Elemente, die zu einem HTML-Formular gehören können. Hier werden Text-Eingabefelder (`<input type="text" ...>`), Optionsfelder

(`<input type="select" ...>`) und versteckte Felder (`<input type="hidden" ...>`) dargestellt. Dies ist nur ein Auszug der möglichen Formular-Elemente, die aber repräsentativ für die verschiedenen Formen der Eingabe sind. Textfelder werden benutzt, um dem User die Möglichkeit zu geben, Freitext einzugeben. Mit Hilfe von Optionsfeldern wird ein Wert aus einer Menge von Werten ausgewählt. In versteckte Felder können Daten zwischengespeichert werden.

Allen diesen Feldern ist gemeinsam, daß sie durch den Browser interpretiert werden und dann als String-Literal in einem HTTP-Request an der Server gesendet werden. Hierbei geht der Typ verloren. Der Server hat somit keine Möglichkeit, festzustellen, aus was für einem Elementtyp die gelieferte Information stammt.

Für das Verständnis der Kommunikation im WWW ist es wichtig, daß eine Konversation im WWW immer vom Browser aus gestartet wird. [1] Es ist dem Server nicht möglich, aktiv Daten zum Browser zu senden. Einzig über Verknüpfungen im Browser, also etwa Links zu Bildern, kann der Transfer von Daten zum Browser initiiert werden. Ob der Transfer allerdings tatsächlich vom Browser gestartet wird, liegt im Endeffekt im Ermessen der Browsers und damit des Endanwenders.

2.5 Konversation

Im vorigen Kapitel wird eine einfache Frage-Antwort-Situation im Web dargestellt. Tatsächlich jedoch sind die Dienst-Angebote im WWW häufig mehr als das: es sind vollwertige Applikationen, die den Browser als universelle und zugleich verbreitete Schnittstelle nutzen.

Die Bedienung dieser Applikation führt zu einer Konversation zwischen dem Menschen und der Maschine, beziehungsweise des Servers. Mittels seines Browsers kann der User eine Applikation auf dem Server nach seinen Wünschen steuern.

Ein gutes Beispiel für einen Dienst-Anbieter, der ein umfangreiches Frage- und Antwortverfahren benötigt, um zu einen Ergebnis zu kommen ist [AG]. Hier wird nach einer Anfrage auf der Startseite immer eine gesonderte, verfeinerte Abfrage durchgeführt, die noch weitere Optionen enthält.

Wenn sich eine Konversation über mehrere Schritte erstreckt, ist es notwendig, daß der Server, auf dem die Applikation läuft, weiß, welche HTML-Seite zu der jeweiligen Anfrage geführt hat. Er muß wissen, in welchem Zustand die Applikation auf Seiten des Anwenders

[1] Eine Ausnahme bilden hier Java-Applets. Das sind kleine Java-Programme, die im Browser laufen. Sie beinhalten Programm-Logik, welche gegebenenfalls beliebige Operationen auf Seiten des Clients ausführen können. Java-Applets sind nicht Teil dieser Ausführung, da sie aufgrund der Tatsache, daß sie Programme sind, nicht in das Schema das WWW passen, wie es in Kapitel 2.1 beschrieben ist.

gerade angezeigt wird.[2] Dies ist nicht trivial, da auf Serverseite kein dauerhaft mit dem Browser verbundener Prozeß läuft. Es ist nicht mal mit Sicherheit gegeben, daß eine dauerhafte Netzwerkverbindung besteht, und es ist dem Server auch nicht möglich, Daten an den Browser zu senden, die dieser nicht erfragt hat. Um allerdings dennoch mehrstufige Abfragen ermöglichen zu können, ist es notwendig, daß der Server den Zustand des Browsers kennt. Hierzu gibt es verschiedene Möglichkeiten, die kombiniert genutzt werden. Zum einen kann eine Applikation die Formulare, die sie ausliefert, soweit mit eindeutigen Parametern versehen, daß die angefragte Antwort komplett auf Basis dieser Daten generiert werden kann. Ein einfaches Beispiel verdeutlicht dies: Eine Webapplikation möchte, daß der Anwender eine Stadt auswählt. Um nicht alle der Applikation bekannte Städte anzeigen (und damit auch zum Browser des Nutzers übertragen) zu müssen, wählt man ein zweistufiges Abfrageverfahren, in dem der Anwender zunächst in einem Formular das Land, in welchem die gewünschte Stadt liegt, auswählt. Als Antwort erhält er ein Formular, mit den Namen der Städte dieses Landes. In diesem Formular ist – zum Beispiel in Form eines `hidden`-Felds – die Information des Landes enthalten. Für den User erscheint dieses Vorgehen, als hätte der Server sich das Land, welches er angefragt hat, gemerkt.

Ein anderer Ansatz, um eine scheinbar dauerhaft bestehende Verbindung zwischen Browser und Server zu erzeugen, ist die Verwendung von Session-Identifikatoren. Hierbei werden Daten auf dem Server gespeichert und einem Identifikator, der SessionID, zugeordnet. Bei jedem Zugriff auf eine Seite wird diese SessionID vom Browser übertragen. Auf dem Server kann dann auf die Sitzungsdaten des Nutzers zugegriffen werden. Das hat den Vorteil, daß zum einen nur eine kleine Menge von Daten (nämlich die SessionID) übertragen werden muß und zum anderen, daß der User auch keine Möglichkeit hat, die Daten zu modifizieren.[3]

Eine letzte Möglichkeit, Zusatzinformationen der Konversation zu speichern, stellen Cookies dar. Es sind vom Server gesendete Informationen, die der Browser speichert und beim wiederholten Besuch der Website an den Server zurücküberträgt. Die Dauer der Speicherung läßt sich dabei frei wählen. Näheres zu Cookies findet sich auf [Coo] und [SSb] .

[2] Die im Browser des Benutzers angezeigte Seite muß nicht immer die letzte, vom Server ausgelieferte Seite sein. Der User könnte beispielsweise über den Zurück-Button auf eine viel früher ausgelieferte Seite gelangt sein.

[3] Da die Daten, wenn sie wie oben beschrieben, übertragen werden und sich in dem Formular versteckt auf dem Computer des Anwenders befinden, ist es ihm auch möglich, diese zu modifizieren.

3 Analyse

Die nachfolgende Analyse soll ermitteln, welche Anforderungen an ein System, welches Daten aus dem WWW extrahiert und an eine bestehende Anwendung weiterreicht, gestellt werden. Das System soll als Brücke zwischen dem WWW und der Anwendung fungieren. Es muß möglich sein, auf einfache Weise Daten aus dem WWW zu extrahieren und diese in eine für die Anwendung benutzbare Form zu überführen. Hierbei soll im fertigen System kein Kodierungsaufwand notwendig sein, so daß das Extraktionssystem auch von Integratoren ohne umfangreiche Kenntnis von Softwareentwicklung wie etwa Systemadministratoren an neue Systeme angeschlossen werden kann.

3.0.1 Gliederung

Es wird zunächst analysiert, wie das System in Hinblick auf die Integration mit dem Basissystem zu gestalten ist. Dann werden Anforderungen an die Schnittstellen zwischen den einzelnen Systemteilen innerhalb des Gesamtsystems ermittelt. (Kapitel 3.2)

Das Kapitel Kapitel 3.3 befaßt sich mit der speziellen Natur HTML-basierter Dienste. Hier wird untersucht, welche Besonderheiten ein HTML-basierter Dienst im Vergleich zu einer klassischen Softwarekomponente ausmacht und wie damit umzugehen ist. Es wird gezeigt, wie Daten aus HTML-Seiten zu extrahieren sind und welchen Anforderungen ein Extraktionssystem gerecht werden muß. Hierzu wird exemplarisch ein als HTML-Seite aufgearbeiteter Datensatz analysiert. Aus den so gefundenen Erkenntnissen wird ein allgemeines Anforderungsmodell erstellt. Dieses wird in Kapitel 3.6 beschrieben. Im Kapitel 3.7 werden technische Details von HTML-Seiten und damit verbundene Anforderungen an das Extraktionssystem dargestellt. Der letzte Teil der Analyse beschäftigt sich mit der Interaktion zwischen dem Extraktionssystem und dem Dienstanbieter über das HTTP-Protokoll.

3.0.2 Rollen im System

In den folgenden Kapiteln wird von verschiedenen Benutzergruppen gesprochen. Ein Mensch kann mehrere dieser Rollen inne haben. Sie werden hier kurz vorgestellt:

- *Endanwender* oder *Anwender* sind Menschen, die das Endprodukt, also beispielsweise die Mitfahrclub-Applikation oder einen Web-Browser, benutzen
- *Entwickler* sind Personen mit Verständnis vom Programmieren und den hier benutzten Technologien. Unter anderem benutzen sie das hier vorgestellte System zur Integration HTML-basierter Dienste, um die von ihnen entwickelten Anwendungen mit Daten aus dem WWW zu versorgen.
- *Administratoren* sind Menschen mit Verständnis von Computersystemen, die aber im Gegensatz zu Entwicklern nicht notwendigerweise über Wissen vom Programmieren verfügen. Ihre Arbeit besteht typischerweise in der Konfiguration bestehender Anwendungen und Systeme
- *Integratoren* sind diejenigen Personen, die das hier analysierte System zur Integration HTML-basierter Dienste dahingehend anpassen, daß es in anderen Umfeldern (andere Dienste gleichen Typs, siehe Kapitel 3.2) funktionieren kann.
- *Benutzer* sind Menschen, die eine der hier beschriebenen Rollen innehaben. Welche Rolle das ist, variiert je nach Kontext.

3.1 Integration des Systems

Das Ziel dieser Arbeit ist, ein System zu schaffen das ein Basissystem mit einem HTML-basierten Dienst verbinden kann. Dies ist insbesondere im Hinblick auf die Integration HTML-basierter Dienste in die Mitfahrbörse gefordert. Zusätzlich ist aber gewünscht, ein möglichst universelles System zu erhalten, welches sich auch in einem anderen Kontext beziehungsweise einer anderen Anwendung nutzen läßt.

In Bezug auf die Mitfahrbörse muß es zum einen möglich sein, andere Arten von Diensten wie im Beispiel regionale- und interregionale Anbieter von Transport-Dienstleistungen, an das Basissystem anzuschließen. Außerdem gilt es, mehrere Anbieter derselben Dienstleistung in das System zu integrieren.

Diese Unterscheidung ist im Hinblick auf die Anforderungen an ein zu erstellendes System zur Integration HTML-basierter Dienste wichtig: wird das Basissystem um neue Funktionalitäten erweitert, ist davon auszugehen, daß hierfür ohnehin Programmieraufwand betrieben werden muß. Wenn dieser Aufwand aber für die Integration von einem Anbieter geleistet ist, so soll danach bei der Integration eines anderen Anbieters derselben Dienstleistung kein weiterer Programmieraufwand entstehen.

3.1.1 Anwendungsbereich 1: Integration des ÖPNV

Ein Anwendungsbereich für externe Dienstleistung wird die Anbindung des öffentlichen Personennahverkehrs (ÖPNV) sein. Hierbei ist geplant, daß der Endanwender auch mit Bus und Bahn in Kombination mit einem Kfz fahren können soll. Derzeit ist die Routing-Engine noch nicht darauf ausgelegt, Routen über den ÖPNV zu suchen. Um dies zu erreichen, gibt es verschiedene Möglichkeiten:

- Zum einen könnte die Routing-Engine gezielt nach Einstiegspunkten zum ÖPNV in Nähe des Start- beziehungsweise Endpunktes der angeforderten Route suchen.
- Zum anderen ist es denkbar, daß die Routen des ÖPNV als Routen, die keine Ablaufzeit haben, im System vorhanden und somit immer verfügbar sind. Andere Routen haben eine Ablaufzeit, nämlich das Eintreffen des Autos am Zielpunkt.[1] Eine Illustration dieses Vorgehens findet sich in Abbildung 3.1.

Wegen der Vielfalt der möglichen Änderungen am Mitfahrclub wird im Rahmen dieser Arbeit darauf nicht weiter eingegangen. Es wird davon ausgegangen, daß für die Planung einer Fahrt vor allem die Informationen über die Abfahrts- und Ankunftszeit sowie eventuell die Start- und Zielbahnhöfe benötigt werden.

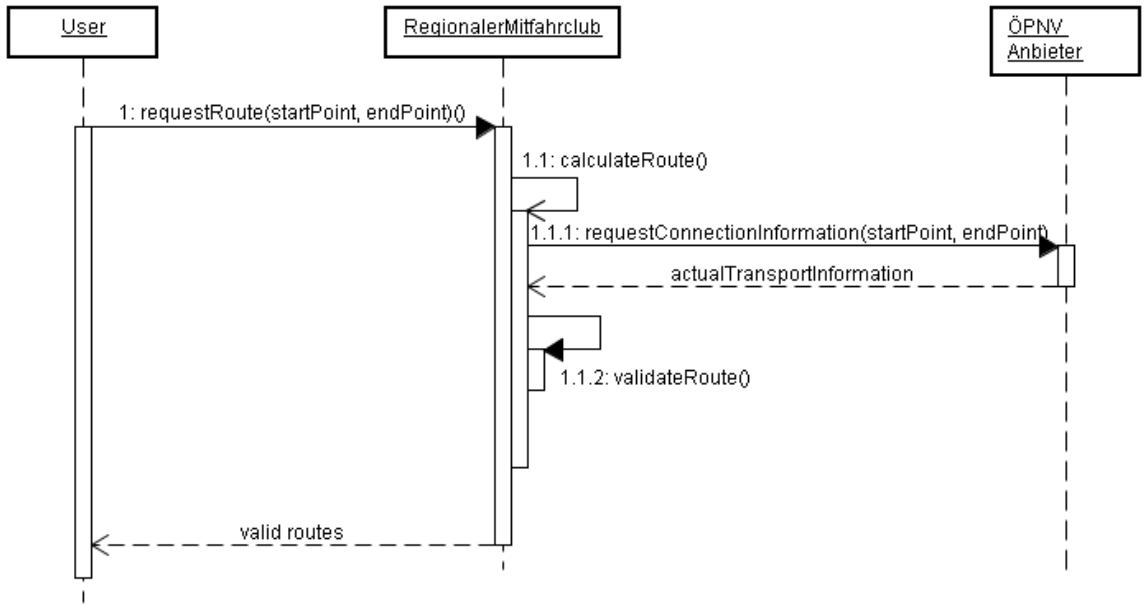

Abbildung 3.1: Anbindung des ÖPNV

[1] Die genaue Umsetzung und die damit verbundenen Probleme, wie etwa die Veränderung der Routen (nicht der Abfahrtszeiten) durch neue Fahrpläne, werden hier nicht weiter diskutiert, da sie nicht Teil des Themenkomplexes sind.

3.1.2 Anwendungsbereich 2: Interregionale Routen

Neben der in Kapitel 3.1.1 vorgestellten Integration des ÖPNV, welcher, wie der Name schon impliziert, nur kurze Streckenabfragen abdecken kann, soll es in Zukunft möglich sein, mit dem Mitfahrclub auch – im geographischen Sinne – weitere Strecken abfragen zu können. Hierzu müssen Verkehrsmittel wie Flugzeuge und Bahnen eingebunden werden. Aufgrund der Vielzahl der Einstiegspunkte (näheres hierzu in Kapitel 3.1) ist es hier nicht möglich, die Routing-Engine zu benutzen, um die Umstiegspunkte zu finden. Es ist vielmehr notwendig, zuerst eine Planung der interregionalen Route vorzunehmen und darauf aufbauend die regionalen Routen zum Startpunkt der interregionalen Verbindung und vom Endpunkt der interregionalen Verbindung zum Ziel zu suchen. Dieses Modell wird in Abbildung 3.2 veranschaulicht.

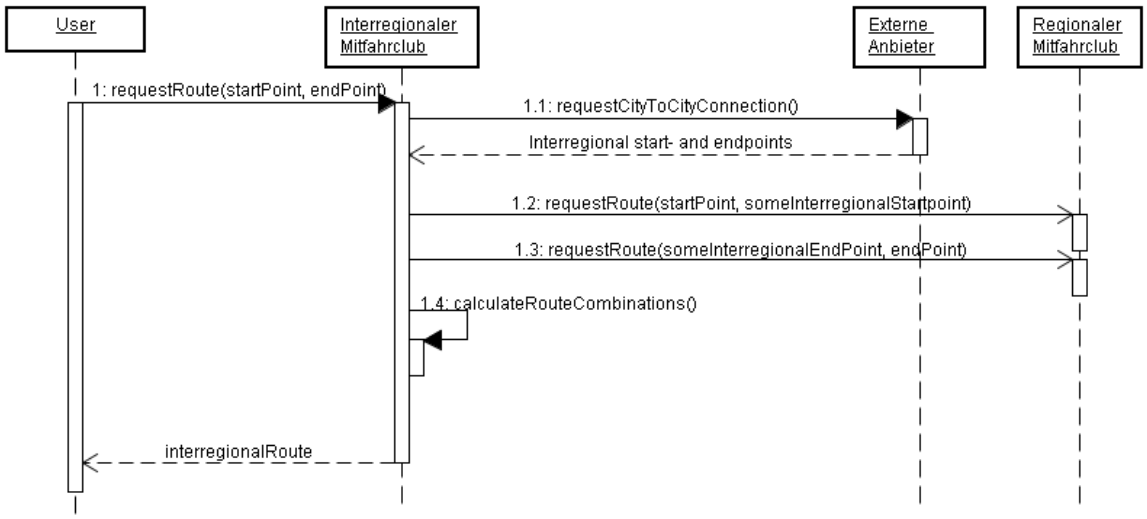

Abbildung 3.2: Hierarchische Planung

3.2 Erweiterbarkeit des Basissystems

Im Folgenden wird von einem Basis-, einem Sub- und einem Extraktionssystem gesprochen. Der Zusammenhang dieser Systeme ist in Kapitel 3.3 dargestellt. Die drei hier vorgestellten Komponenten sind fiktiv und können in der tatsächlichen Anwendung vom vorgestellten Schema abweichen.

Im einzelnen übernehmen die Komponenten die folgenden Aufgaben:

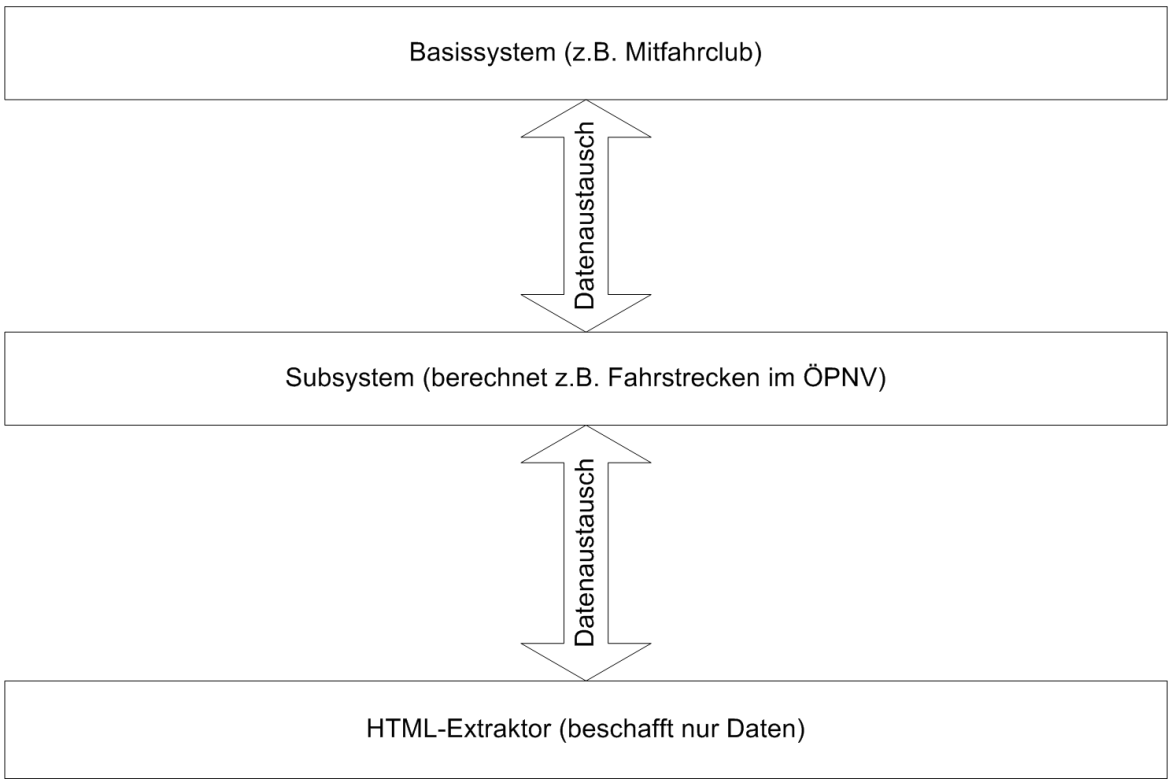

Abbildung 3.3: Komponenten der Schnittstellenarchitektur

- Das Basissystem stellt den Kern der Anwendung dar. Im Falle des Mitfahrclubs ist dies die bislang bestehende Anwendung
- Das Subsystem ist ein System, welches dem Basissystem neue Funktionalität hinzufügt. Dieses könnte im Falle des Mitfahrclubs beispielsweise die Organisation und Planung der Routen mit öffentlichen Verkehrsmitteln übernehmen.
- Das Extraktionssystem beschafft auf Basis von Eingabeparametern Daten, welche das Subsystem verarbeitet. Für den Mitfahrclub könnten dieses beispielsweise Fahrplaninformationen sein.

Wie die in der Hierarchie erste Schnittstelle zwischen dem Basissystem und dem Subsystem zu gestalten ist, ist nicht Teil dieser Ausarbeitung, da diese keinen Einfluß auf die Extraktionsschnittstelle haben sollte. Es ist – je nach konkreter Realisierung – nicht einmal notwendig, daß eine solche Schnittstelle existiert.

Entsprechend den Anforderungen müssen neue Anbieter gleichen Typs ohne größeren Aufwand zu integrieren sein. Es soll außerdem möglich sein, das Extraktionssystem, beziehungsweise die für die Interaktion mit speziellen Anbietern erstellten Module auch in anderen Anwendungen zu benutzen. Daraus ergibt sich, daß alle Daten, die aus einem HTML-

basierten Dienst in das System gebracht werden, so aufgearbeitet werden, daß sie vom entsprechenden Subsystem sinnvoll weiterverarbeitet werden können. Die Schnittstelle muß einheitlich für alle Anbieter sein, da sonst unter Umständen eine Modifikation des Subsystems durch den Anschluß eines weiteren Extraktors notwendig wird.

Diese Anforderung erscheint zunächst trivial. Bei näherer Untersuchung stellt sich aber heraus, daß diese Forderung gerade im Umfeld des Mitfahrclubs, welcher mit Adressen und anderen Geoinformationen arbeitet, nicht einfach einzuhalten ist.

Generell wird zwischen strukturierten (etwa Uhrzeiten oder Preise) und unstrukturierten Daten (etwa Namen) unterschieden. Auf beide Arten von Daten muß möglicherweise bei der Weiterverarbeitung eingegangen werden. Deswegen müssen solche Daten, die im weiteren Programmverlauf in interpretierbarer Form vorliegen müssen, in ein entsprechendes Format übersetzt werden. Die Daten, die durch das Extraktionssystem geliefert werden, müssen, sowohl in ihrer Struktur als auch in ihrer Semantik, klar definiert sein. Diese Definition muß Teil der Schnittstellendefinition sein.

Nachfolgend werden zwei Beispiele aufgeführt. Im Falle des Mitfahrclubs soll es möglich sein, einen ÖPNV-Dienstleister einzubinden. Ein Teil der Daten, die das System von diesem Anbieter anfordert, werden die Fahrzeiten des Verkehrsmittels sein, welches der ÖPNV-Anbieter zur Verfügung stellt. Diese müssen syntaktisch und semantisch einheitlich sein, wenn sie im ÖPNV-Subsystem ankommen. Die Datenbasis allerdings besteht aus HTML und somit aus Zeichenketten. Zeichenketten haben keine gleich bleibende Syntax, etwa für Zeitpunkte oder Zeiträume. Sie müssen in einen Fachwert überführt werden, der für alle Extraktionssysteme desselben Typs einheitlich ist.

Ein Beispiel für unstrukturierte Informationen, die semantisch einheitlich sein müssen, stellen Namen von Bahnhöfen dar. Die Information ’Hamburg Hauptbahnhof’ kann auf verschiedene Arten, wie etwa ’Hamburg, Hbf’ oder ’HH Hauptbahnhof’ dargestellt werden. Jede dieser Darstellungen steht für denselben Bahnhof. Die Maschine aber erkennt nur zwei voneinander verschiedene Zeichenketten und kann somit weder feststellen, daß es sich hier um den Hauptbahnhof in Hamburg handelt, noch, daß diese beiden Informationen äquivalent sind.

Daten, die im weiteren Programmverlauf nicht interpretiert werden müssen, sind in diesem Zusammenhang kein Problem. Diese werden unabhängig von ihrem Inhalt weiterverarbeitet. Eine Unifizierung, wie sie für semantische Daten vorzunehmen ist, entfällt hier.

Zusammengefaßt gilt für die Art der verwendbaren Daten, daß diese entweder eine Struktur haben oder nicht weiter interpretiert werden müssen. Die Übergänge sind hierbei fließend. Gut geeignet für eine Interpretation sind stark strukturierte Daten wie etwa Zahlen, gänzlich ungeeignet beispielsweise natürlichsprachliche Daten. Dazwischen werden beispielsweise

Daten wie die genannten Bahnhofsnamen eingeordnet. Diese können eventuell über ein einfaches Übersetzungsschema interpretiert und vereinheitlicht werden. [2] Alle Arten von Daten, deren Grad an Strukturiertheit zwischen diesen beiden Extremen liegt müssen im Einzelfall geprüft werden.

3.3 HTML als Datenquelle

> In a remarkably short space of time, HTML became wildly popular and rapidly outgrew its original purpose. Since HTML's inception, there has been rapid invention of new elements for use within HTML (as a standard) and for adapting HTML to vertical, highly specialized, markets. This plethora of new elements has led to interoperability problems for documents across different platforms. [Divb]

Um ein System wie den Mitfahrclub mit Daten zu versorgen, die über die Daten der teilnehmenden Autofahrer hinausgehen, zum Beispiel Daten von ÖPNV-Anbietern, muß ein Zugang geschaffen werden, über den Daten dieser Anbieter abgefragt werden können. Dieser Zugang wird typischerweise in Form einer Programmierschnittstelle geliefert, welche Methoden zur Abfrage von den entsprechenden Verbindungsdaten anbietet. Eine Abfragemethode wird dann beispielsweise als

```
TimeQueryAnswer getTransportTimes(
    Date startOfTransport, Station origin, Station destination)
```

definiert. Die Antwortmenge ist eine Instanz der Klasse TimeQueryAnswer, welche die Abfahrt- und Ankunftszeit der angeforderten Fahrt enthält. Im Falle eines Fehlers wird eine Ausnahme gemeldet, die das abfragende Programm behandeln muß.

Im vorliegenden Falle muß davon ausgegangen werden, daß so eine wohldefinierte Schnittstelle nicht vorhanden ist. Stattdessen gibt es eine Schnittstelle, welche aus einer Technologie entstanden ist, die ursprünglich nicht hierfür vorgesehen war: dem WWW. Das WWW war – in seiner ursprünglichen Form – wie bereits in der Definition dargestellt, als eine Sammlung von Dokumenten gedacht, welche untereinander über Hyperlinks miteinander verbunden sind. Aus diesem Grund wurde eine Sprache für das WWW entwickelt, welche für die Beschreibung von Dokumenten optimiert ist. Diese Sprache wird heute überwiegend zur reinen Darstellung benutzt. Diese Fixierung auf HTML ist besonders wichtig: HTML ist eine Dokumentenbeschreibungssprache, die dazu genutzt wird, Informationen zu visualisieren.

[2] Es gibt in der Informatik Bestrebungen, auch diese Daten interpretieren zu können. Das ist aber nicht Teil dieser Ausarbeitung.

Das macht es für den Anbieter von Informationen sehr leicht, diese zu publizieren. Im Laufe der Zeit hat sich das WWW stark verändert. Es ist nicht länger ein Netz von statischen, untereinander verlinkten Seiten. Eine große Anzahl von Infomationsangeboten sind interaktiv. Dazu wurden die Grundtechnologien HTML und HTTP erweitert. Es wurden Formulare eingeführt, die in Webseiten eingebunden werden können. Außerdem wurden auf Serverseite Mechanismen geschaffen, die Daten aus Formularen zu empfangen und entsprechend individuelle Antworten zu erzeugen.

Leider ergibt sich daraus der Nachteil, daß eine Integration in darauf aufbauende Informationssysteme nur schwer möglich ist: zwar stellt das WWW über die eben genannten Mechanismen eine gewisse Interaktivität her, diese ist jedoch nicht zu vergleichen mit einer Programmierschnittstelle. Für die Kommunikation ist lediglich das Übertragungsprotokoll definiert. Eine genaue Definition der Schnittstelle zu den einzelnen Diensten fehlt gänzlich oder ist zumindest nur in Form der Formulare auf den Eingabeseiten vorhanden. Diese wiederum läßt sich nur schwer auswerten. Ein Mechanismus, wie er beispielsweise bei Web Services in Form der WSDL[3] oder bei CORBA in Form der IDL vorliegt, existiert nicht. Hieraus ergibt sich eine konkrete Anforderung an die Extraktionssoftware: Die Kommunikation muß in Bezug auf die zu übertragenden Parameter und Werte untersucht werden können.

Ein noch größeres Problem als das Fehlen von formal definierten Abfrageschnittstellen, stellt die Fixierung auf HTML als Beschreibungssprache für die ausgetauschten Daten dar. Es ist in einer einfachen[4] HTML-Seite weder möglich anzugeben, welcher Art die gegebene Information ist, noch welches Format die angegebene Information hat. Obgleich also die Anbieter über geeignete Applikationen zur Beschaffung von Informationen verfügen, sind diese Informationen über Format und Typ der Daten nicht verfügbar. Stattdessen werden die Informationen für die Anzeige aufgearbeitet und nur in dieser Form ausgegeben.

Ein Beispiel verdeutlicht das damit verbundene Problem: Würde ein Mensch das in Kapitel 3.1.2 erläuterte Szenario ohne die Integration der Fremdanbieter manuell nachstellen wollen (was in anderer Form tagtäglich passiert), so würde er nach einer Möglichkeit suchen, um von einer Stadt in die nächste zu gelangen. Dazu wird eine Anfrage an einen oder mehrere Dienstleister gesendet siehe Abbildung 3.4. Danach wird der Anwender eine Anfrage an den örtlichen ÖPNV-Anbieter senden. Das Ergebnis ist in Abbildung 3.5 dargestellt. Die letzte

[3] über die WSDL lassen sich auch Anfragen an HTML-basierte Dienste spezifizieren. Zum einen allerdings sind die Antworten nicht notwendigerweise genauer spezifiziert als 'eine HTML-Seite' oder 'ein Bild'. Zum anderen ist es immer noch so, daß die Mehrheit der bereits verfügbaren Dienste nicht im Hinblick der Nutzung als Webservice geplant und umgesetzt wurde. Eine formalisierte Beschreibung fehlt also weiterhin.

[4] Im Zuge der ständigen Weiterentwicklung des WWW und seiner Technologien gibt es Bestrebungen, diese Einschränkungen zu beheben. HTML bietet bereits seit einiger Zeit die Möglichkeit, Meta-Informationen anzugeben, welche den Inhalt einer Seite beschreiben. Außerdem ist es mittels der neuren HTML-Versionen möglich, über Attribute zusätzliche Informationen, wie eine Klassifikation der dargestellten Daten, in einer HTML-Seite unterzubringen. Mehr zu diesem Thema findet sich auf [Divb],[Dum], [SD] und in Kapitel 3.5.

Etappe der Reise könnte dann aus der Mitfahrbörse kommen und z.B. per SMS auf das Mobiltelefon des Teilnehmers gesendet werden.

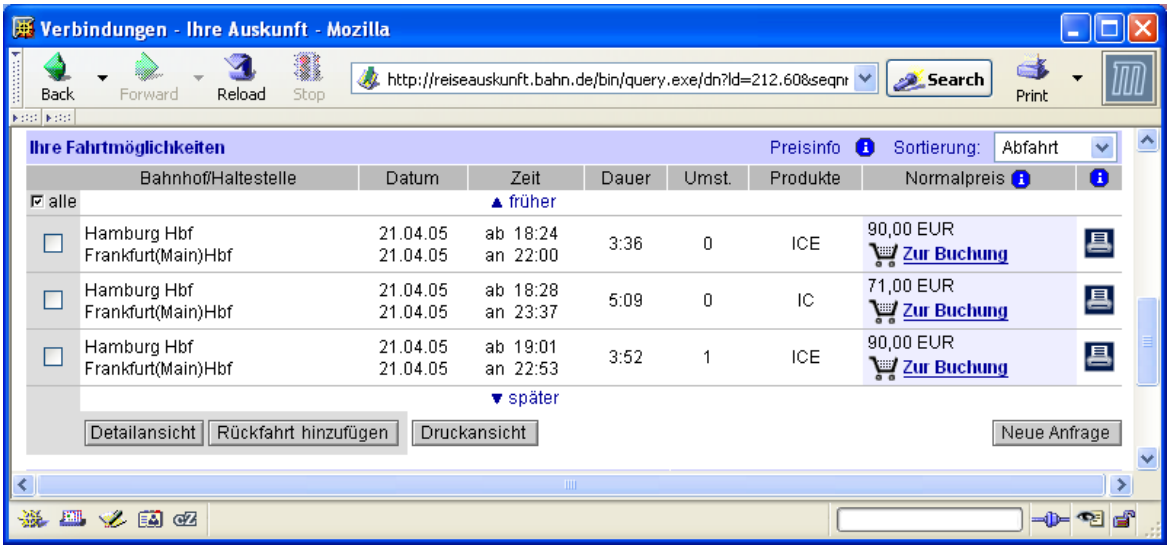

Abbildung 3.4: Das Ergebnis einer Anfrage aus Sicht des Benutzers

Auch wenn dieser Weg für Menschen gangbar ist, stellt er noch keine optimale Lösung hinsichtlich der Benutzerfreundlichkeit oder Informationssicherheit dar. Bereits die dritte vom ÖPNV Anbieter vorgeschlagene Verbindung in Abbildung 3.5 ist nicht zu gebrauchen, da der Anschlußzug in Abbildung 3.4 bei Erreichen des Bahnhofs bereits abgefahren ist. Es ist besser, wenn die – hier mühselig manuell – gesammelten Informationen von einem übergeordneten System verarbeitet werden.

So angenehm die Aufarbeitung der Daten in HTML für den Dienstanbieter auch ist, so schwer ist es, diese für einen Computer in eine andere Form als die direkte Darstellung zu überführen. Ein Blick in den Quellcode illustriert dies. Der Seitenquelltext, der die erste Zeile der Ergebnistabelle aus Kapitel 3.4 darstellt lautet:

```
</nobr></td><td class="sepline  screennowrap">
<nobr>
<a class="nolink"
href="http://reiseauskunft.bahn.de/bin/query.exe/dn?ld=212.60
&amp;seqnr=3&amp;ident=5h.0944360.1114099991&amp;
guiVCtrl_connection_detailsOut_add_C1-0#cis_C1-0"
title="Details zu dieser Verbindung anzeigen">
Hamburg Hbf
<br>
Frankfurt(Main)Hbf
</a>
</nobr>
```

```
</td>
<td class="sepline  right top nowrap"> <br> </td>
<td class="sepline  top nowrap">
<nobr> <br> </nobr></td>
<td class="sepline  right top nowrap">
<nobr> <br> </nobr></td>
<td class="sepline  center nowrap">
21.04.05<br>21.04.05</td>
<td class="sepline  right">
ab<br>an</td><td class="sepline  left">18:24<br>22:00</td>
<td class="sepline  center nowrap">3:36</td><td class="sepline
center nowrap">0
</td><td class="sepline  center screennowrap">
<a class="nolink"
href="http://www.bahn.de/hilfe/view/auskunft/
r4.28/pk/de/alle_ergebnisseiten_produktinfo.shtml"
onclick="popUp('about:blank','help','cms');" target="help">ICE
</a></td><td class="sepline emphasize screennowrap">
90,00 EUR <br>
<img src="/img/ekw.gif" width="22" height="21" border="0"
alt="" class="middle">
<a href="https://reiseauskunft.bahn.de/bin/query.exe/dn?
ld=212.60&amp;seqnr=3&amp;ident=5h.0944360.1114099991
&amp;selectedTariffId=1&amp;outConId=C1-0&amp;order=yes&amp;
waitForBooking=yes&amp;protocol=https:&amp;"
class="emphasize" title="Mit Tickets zum Normalpreis haben Sie
volle Flexibilität und können jeden Zug
benutzen.">Zur Buchung</a> </td>
```

In der visualisierten Version erkennt der Mensch mühelos, daß der zu nehmende Zug am 21.04.2005 um 18.24 Uhr in Hamburg-Hauptbahnhof abfährt und am selben Tag um 22.00 in Frankfurt (Main)-Hauptbahnhof ankommt. Die Maschine hat hiermit ihre Probleme, da die einzelnen Teile der Informationen in keiner Form typisiert oder formalisiert sind. Ein genauer Blick auf die dargestellten Daten verdeutlicht dies. Der Quellcode:

```
<a class="nolink"
href="http://reiseauskunft.bahn.de/[...]" title="Details zu
dieser Verbindung anzeigen">
Hamburg Hbf
<br>
```

```
Frankfurt(Main)Hbf
</a>
```

stellt den Start und den Zielbahnhof dar. In der HTML-Spezifikation, welche die einzige formale Beschreibung dieser Seite ist, kann man nur finden, daß der Bereich, den das <a href..>-Tag einschließt, eine Verbindung zu einer weiteren Seite beinhaltet. Desweiteren wird nur ein
-Tag benutzt, um den Start- und den Zielbahnhof zu trennen. Formal bedeutet dies nicht, daß jetzt ein anderer Bahnhof oder überhaupt ein anderes Datum kommt, sondern vielmehr, daß hier die Zeile umzubrechen ist.

Dasselbe Szenario sieht, sehr viel einfacher aus, wenn entsprechende Programmierschnittstellen vorhanden sind. Es werden eindeutig definierte Daten in einer eindeutig definierten Datenstruktur übertragen. Außer den angefragten Daten wird nichts übertragen. Eine Fehlerbehandlung ist aufgrund von Ausnahmen möglich.

Ein weiteres Problem stellt die Umgestaltung von HTML-Seiten dar. Eine Programmierschnittstelle wird – solange sie den Anforderungen genügt – nicht verändert (und wenn doch, dann wird ein Software-Entwickler sich bemühen, sie so zu gestalten, daß die Änderungen abwärtskompatibel sind). Eine HTML-Seite wird im Gegensatz dazu jedoch verändert, weil das Layout modifiziert werden soll. Damit unterliegt eine HTML-Seite und damit die Schnittstelle größeren Schwankungen als eine Programmierschnittstelle. Dieser Umstand unterstreicht noch einmal die Wichtigkeit der eingangs bereits erwähnten Konfigurierbarkeit.

Die bei der Benutzung des WWW verwendeten Protokolle und Mechanismen sind für sich betrachtet im allgemeinen einfach. Ein gutes Beispiel ist das HTTP-Protokoll: Hier werden wie in Kapitel 2.4 beschrieben einzelne Anfragen gesendet und empfangen. Zur vollständigen Darstellung einer HTML-Seite gehört aber neben dieser einen Anfrage (und der Erzeugung der eigentlichen Darstellung) wesentlich mehr. Es werden diverse Anfragen an DNS-Server gesendet, es wird JavaScript ausgeführt, es werden Inhalte von Frames aufgelöst und so weiter. Von dieser Arbeit wird der Benutzer eines Browsers nichts merken. Um eine Analyse der Schnittstelle zu einem Dienst vorzunehmen, ist es notwendig, diese einzelnen Operationen, sofern sie für die Benutzung des Dienstes relevant sind, aufzubereiten. Hierbei muß der Integrator vom Computer sinnvoll unterstützt werden.

3.4 Exemplarische Analyse einer HTML-Seite

Im folgenden wird exemplarisch eine Seite des Informationsdienstes Geofox des Hamburger Verkehrsverbundes analysiert. Sie wird in Abbildung 3.6 dargestellt. In dieser Darstellung wurden bereits Veränderungen vorgenommen um die grobe Struktur besser erläutern zu können. Die Seite ist in drei Tabellen-Ebenen unterteilt. Sie sind in der Dar-

stellung (ihrer Rekursionstiefe nach sortiert) mit den Farben Blau, Gelb und Rot umrandet. Außerdem wurden einige der für die Darstellung der Tabellen benötigten HTML-Tags (`<table>`, `</table>`, `<td>`, `</td>`) sichtbar gemacht und Rahmen um die einzelnen Zellen gelegt. Der komplette Quelltext findet sich auf der CD im Anhang.

Die Seite besteht auf der untersten Ebene der Darstellung aus vier Tabellen. Von diesen in der Darstellung rot umrandeten Tabellen ist nur die zweite von Interesse, da sie die eigentlichen Informationen enthält. Diese Tabelle besteht aus einer Spalte mit sechs Zeilen (im Beispiel an den `<td>`-Tags zu erkennen). In der dritten Zeile befindet sich eine weitere Tabelle, in welcher die Fahrzeitinformationen zu finden sind.

Jede einzelne der Fahrzeittabellen besteht aus vier Zeilen. In anderen Abfragen kann die Anzahl dieser Zeilen variieren, je nachdem wie viele unterschiedliche Verkehrsmittel für diesen Streckenabschnitt benutzt werden. Im Beispiel ist dies ein Verkehrsmittel (U1) mit zwei unterschiedlichen Fahrzielen.

Bis zu diesem Punkt sieht die Grobstruktur der Seite im Quelltext so aus:

```
<html>
<head>...</head>
<body>...
<table>...Persönlicher Fahrplan...Haltestellenaushang..</table>
<table>
    <tr><td>Ihr persönlicher Fahrplan</td></tr>
    <tr><td> </td></tr>
    <tr><td>
        <table>
            <tr>
                <td> </td>
                <td>Abfahrt (gewünscht)...</td>
            </tr>
            <tr>
                <td>Start:</td>
                <td>Garstedt</td>
            </tr>
            <tr>
                <td> </td>
                <td><table>
                        <tr><td>Garstedt</td></tr>
                        <tr><td>[U1]Grosshansdorf</td></tr>
                        ...

                    </table>
                </td>
                ...
                <td><table>
```

```
                        <tr><td>17:02</td></tr>
                        <tr><td><img src="U1.gif"></td></tr>
                        <tr><td>|</td></tr>
                        <tr><td>17:17</td></tr>
                    </table>
                </td>
                <td><table>...</table></td>
                <td><table>...</table></td>
            </tr>
            <tr>
                <td>Ziel:</td>
                <td>Ohlsdorf</td>
            </tr>
        </table>
    </td></tr>
    ...
</table>
<table>...</table>
<table>...</table>
</body>
</html>
```

Der tatsächliche Quellcode ist um ein Vielfaches unübersichtlicher, da hier noch weitere Tags benutzt werden, die die Darstellung verändern. Der komplette Pfad [5] zum ersten Abfahrtsdatum kann folgendermaßen dargestellt werden:

```
/HTML/BODY[1]/DIV[1]/
    TABLE[1]/TR[3]/TD[1]/DIV[1]/
        TABLE[1]/TR[3]/TD[4]/
            TABLE[1]/TR[1]/TD[1]/SPAN[1]/
```

In diesem Beispiel sieht man bereits, daß diverse Ebenen durchlaufen werden müssen, um die gewünschte Information zu finden. Das wird noch dadurch erschwert, daß diese Ebenen durchaus nicht homogen sind. So finden sich in der Zeile, in der die Fahrplaninformationen zu finden sind, weitere Tabellen. In der Zeile der gelben Tabelle wird nur ein Link dargestellt.

Es gibt abgesehen davon eine Reihe verschiedener Möglichkeiten, Informationen darzustellen. Tabellen sind nur ein Mittel. Der Pfad zu den Informationen über die Reisezeit sowie die Preisauskunft wird auf eine andere Weise dargestellt: hier wurde keine Tabelle benutzt,

[5] Hier wird die XPath Syntax benutzt. Näheres wird in Kapitel 4.2.3 beschrieben.

sondern es wurde in eine Zeile der übergeordneten Tabelle eine über
-Tags (und damit Zeilenumbrüchen) separierte Liste eingesetzt.

3.5 Analyse von HTML-Seiten

Im folgenden wird wiederholt von Syntax und Semantik gesprochen. Um diese Begriffe in Bezug auf HTML zu verstehen, ist es notwendig, HTML ein wenig genauer vorzustellen. Ein HTML-Dokument gehört zur Gruppe der SGML-Dokumente. [6]. Dazu gehört immer eine Data Type Definition (DTD). Darin wird unter anderem festgelegt, in welcher Reihenfolge und Schachtelung die Tags zu benutzen sind. Daraus ergibt sich die Syntax der einzelnen Auszeichnugskonstrukte. [7] . In der Definition der SGML-Syntax beziehungsweise Grammatik (beschrieben in [Bin]) wird wiederum festgelegt, wie die einzelnen Auszeichnungskonstrukte zu formulieren sind. Dieser Teil der Syntax findet hier keine Beachtung. Weitere Informationen über Systeme zur Verarbeitung von solchen Strukturen finden sich in der Literatur etwa in [Aho85]). Stattdessen wird von der Syntax, welche über die DTD beschrieben ist, gesprochen. Der Zusammenhang zwischen SGML und HTML wird in [RHJc] erläutert.

Ein Beispiel soll dies verdeutlichen: Interessant ist, daß nach einem <head>-Tag ein <title> und (unter anderem) ein oder mehrere <meta>Tags geschrieben werden dürfen. Diese Definition wird über die DTD gegeben. Die Tatsache, daß Tags mit spitzen Klammern beginnen und wieder aufhören wird als gegeben angenommen.

Der semantische Wert eines solchen Tags ist durch die Spezifikation der Dokumentklasse (HTML-Version 4.01 wird beispielsweise in [RHJb] definiert) vorgegeben. Auch hierzu ein Beispiel: Das Tag
 hat den semantischen Wert 'beende an dieser Stelle den Textfluß und breche die Zeile um'.

Der Kernpunkt des Systems zur Extraktion der Daten besteht darin, Daten in einer HTML-kodierten Seite zu lokalisieren und zur weiteren Verwendung auszulesen. Das Hauptproblem besteht hierbei darin, daß die Daten in einer HTML-Seite zwar durchaus eine Struktur haben, die bestimmten Regeln folgt, diese aber leider in der Regel nur sehr allgemeine Informationen darüber zur Verfügung stellt, welcher Art die einzelne Informationen sind.

Im Vergleich zu anderen Datenaustauschformaten, wie etwa XML, ist HTML ein allgemeines Format zur Beschreibung von Dokumenten. Bei korrekter Anwendung lassen sich Dokumente damit auch gut beschreiben. Leider ist von einer solchen Anwendung meistens nicht auszugeben. Es werden dabei vor allem zwei Fehler gemacht:

[6] Man spricht auch von einer SGML-Applikation

[7] *Auszeichnung*, abgeleitet aus dem englischen *markup* (HTML ist – wie bereits erwähnt – eine 'markup-language')

- Syntax-Fehler [8]
- Verwendung von semantisch nicht korrekten Elementen der HTML-Sprache

Zunächst ein Beispiel für einen Syntaxfehler: Innerhalb einer Überschrift (`<h1>`) wird ein Wort betont (`<e>`). Dabei wurde die Reihenfolge der schließenden Tags vertauscht:

```
<h1>So ist es <e>FALSCH</h1></e>
<h1>So ist es <e>RICHTIG</e></h1>
```

Interessant ist, wie darauf reagiert wird: entgegen den Erwartungen stellt mindestens der Mozilla-Browser eine (ansonsten korrekte) Seite mit diesem Fehler richtig dar. Das ist mitverantwortlich dafür, daß diverse HTML-Seiten syntaktisch falsch kodiert sind. Web-Browser und andere User-Agents bedienen sich zur Kompensation bei Softwarekomponenten, die eine automatische Korrektur der falsch kodierten HTML-Seiten vornehmen. Bezogen auf den Browser hat das zur Folge, daß die Darstellung der HTML-Seiten nicht immer vollständig nachvollziehbar ist, da die Regeln für eine Korrektur zumeist[9] unbekannt sind. Für die Implementierung einer Lösung zum gegebenen Problem bedeutet dies, daß ein ähnlicher Mechanismus eingesetzt werden muß.

Ein zweites Beispiel soll die Probleme illustrieren, die in Bezug auf semantisch fehlerhaft ausgewählte Tags entstehen. Einige HTML-Elemente haben eine erfreulich präzise Definition dessen, was sie beinhalten. Das `cite` zählt zu diesen Tags. Anstatt dieses einzusetzen und damit zu signalisieren, daß der umschlossene Text ein Zitat ist, wird vielmehr ein "-Zeichen eingesetzt, welches den Browser instruiert, ein Anführungszeichen zu setzen. Durch die Benutzung dieses Zeichens ist im Gegensatz zur Benutzung eines korrekt gewählten Tags keinerlei Hinweis auf die Semantik des eingeschlossenen Texts gegeben.

Doch selbst wenn die vorhandenen Tags korrekt benutzt würden, ließe sich daraus immer noch kein zuverlässiger Weg ableiten, Daten zu identifizieren. HTML macht nur Angaben zu typischen Strukturelementen wie etwa Titel und Überschriften eines Dokumentes. Eine genauere Spezifikation ist nicht vorgesehen. Abhilfe könnte die Benutzung von `<span>` oder

[8] Beispielsweise ist keine der in diesem Dokument hauptsächlich verwendeten Seiten HTML-konform. Man kann dieses mittels des Validators des W3C (zu finden auf [cona]) leicht selbst überprüfen. Konkret überprüft wurden die Startseiten der Deutschen Bahn [AG], des Geofox Informationssystems [Gmb], des Dienstes Opodo [Lim] und die Seite der Suchmaschine Google [inc].

[9] Mindestens im Bereich der Open-Source-Software sind sie in Form des Browser-Quelltextes streng genommen bekannt. In der Praxis allerdings sind diese Regeln den Autoren von HTML-Seiten eher nicht geläufig. (Die Kodierung von korrektem HTML ist mit Sicherheit eine bessere Alternative als auf die Korrekturverfahren einzugehen)

-Tags bringen. Diese Elemente sind allgemeine Strukturelemente, denen über Attribute Meta-Informationen über ihren Inhalt beigelegt werden können.[10] Es ist somit prinzipiell möglich auch in HTML eindeutige Klassifizierungen von Daten vorzunehmen. Die Klassifizierung gibt eine gewisse Sicherheit, daß es sich bei dem Datum auch wirklich um eine gewünschte (oder nicht gewünschte) Information handelt. In der Praxis findet dies jedoch kaum Anwendung. Stattdessen werden diese Tags meistens benutzt, um eine Formatierug über Stylesheets vorzunehmen. Ein Beispiel für eine falsche Benutzung ist:

```
<span class="bold">12. Juni 2005</span>
```

Über ein Stylesheet kann dann eine Hervorhebung (im Beispiel ”bold” für Fettschrift) des Datums vorgenommen werden. Besser wäre dem `class`-Attribut einen beschreibenden Namen, etwa ”date”, zuzuweisen.

Tatsächlich werden solche Elemente und Klassifizierungen keinesfalls überall benutzt. Somit können sie nicht verwendet werden, um Kontext zu identifizieren.

Es wurde bereits erwähnt, daß zur HTML eine implizit vorgeschriebene DTD gehört.[11] Dies unterscheidet HTML von XML: In XML muß in jedem Dokument eine DTD explizit angegeben werden. Sie beschreibt die Syntax oder Grammatik der Dokumentklasse, zu dem das Dokument gehört. Das führt dazu, daß normalerweise alle Informationen innerhalb des Dokuments sinnvoll klassifiziert sind. Leider bedeutet es nicht, daß die Informationen von einem Computersystem besser verstanden werden können. Etwa ein in der DTD definiertes XML-Tag <zusammenfassung> hat für den Computer denselben Informationsgehalt wie etwa ein <title>-Tag in HTML. Außerdem verhindert auch das über die DTD definierte <zusammenfassung>-Tag nicht, daß hier ein semantisch völlig falscher Wert, wie eine

[10] Gerade diese beiden Elemente stellen ein sehr gutes Beispiel dafür dar, daß selbst heute noch HTML vielfach als Layout-Sprache verstanden wird und die Idee, semantische Informationen über das Dargestellte miteinzubetten, dabei nicht beachtet werden. Das deutsche Standardwerk selfhtml [Mün] schreibt zu dem div-Tag:

> Es ist dazu gedacht, um mit Hilfe von CSS formatiert zu werden.

[SSa]

In der Spezifikation wird es etwas anders beschrieben:

> The DIV and SPAN elements, [...], offer a generic mechanism for adding structure to documents. [...] Thus, authors may use these elements in conjunction with style sheets [...] to tailor HTML to their own needs and tastes.

[RHJa]

In der Spezifikation wird also darauf hingewiesen, daß es möglich ist, eine Formatierung über CSS vorzunehmen, nicht aber, daß es der Sinn des Tags ist.

[11] Mit der Einführung von XHTML hat sich das verändert. Zum Zeitpunkt der Erstellung der Ausarbeitung ist XHTML aber noch immer nicht weit verbreitet, insofern wird darauf nicht explizit eingegangen.

Uhrzeit[12], eingetragen wird. Die Wandlung in einen Wert, der von der Geschäftslogik eines mit XML-Daten versorgten Informationssystems verstanden wird, muß nach wie vor explizit erfolgen.

Zusammenfassend wird festgestellt, daß es in HTML diverse Ansätze gibt, Dokumente um semantische Informationen zu erweitern. Diese Ansätze finden in der Praxis bislang allerdings weder überall Anwendung, noch sind sie ausreichend für die Klassifizierung von beliebigen Daten. Deshalb sind sie nicht geeignet, Daten innerhalb einer HTML-Seite zu identifizieren beziehungsweise aufzufinden.

3.6 Systematisches Extrahieren von Daten aus einer HTML-Seite

Aus der bisher erfolgten Analyse geht hervor, daß die eigentlichen semantisch eindeutig definierten HTML-Tags keinerlei Hilfe bei der Auffindung der gewünschten Informationen sind. Aus diesem Grund muß ein anderer Weg gefunden werden. Ein denkbarer Ansatz ist die Byteposition innerhalb einer HTML-Seite zu speichern und Daten immer wieder von dieser Stelle zu lesen. Dieser Ansatz ist jedoch viel zu wenig robust, da jede kleine Veränderung in der Seite, die Bytepositionen verschiebt.

Ein weiterer Ansatz ist, sich die in der Praxis starke Benutzung von Tabellen zu Nutze zu machen.[13] Eine Tabelle ist ein leicht identifizierbares Konstrukt in HTML, welches relativ strengen Regeln folgt. Eine Adressierung einzelner Zellen ist mittels einer Zeilen- und Spaltenangabe möglich. Auch dieser Ansatz ist noch nicht universell genug. Ein Blick auf den in Kapitel 2.1 dargestellten Quelltext zeigt:

```
<td class="sepline  screennowrap">
<nobr>
<a class="nolink"
href="http://reiseauskunft.bahn.de/[...]>
Hamburg Hbf
<br>
Frankfurt(Main)Hbf
</a>
```

[12] In einer DTD können auch im gewissen Umfang Angaben zur Struktur der Daten gemacht werden, insofern ist der umgekehrte Weg, eine Zusammenfassung in ein Uhrzeit-Tag einzutragen nicht möglich.

[13] Tabellen werden in HTML-Seiten nicht nur benutzt, um tatsächlich tabellarische Daten darzustellen, sonden werden vielmehr sehr oft zur optischen Gestaltung der Seite eingesetzt.

```
</nobr>
</td>
```

Obwohl in dem Beispiel genau eine Tabellenzelle dargestellt wird, werden bereits zwei verschiedene Informationen (Start- und Zielbahnhof) ausgegeben. Getrennt werden diese nur durch ein `<br>`. Typischerweise sollen diese Informationen voneinander getrennt ausgewertet und weiterverarbeitet werden. Somit ist der Zugriff auf einzelne Tabellenzellen nicht feingranular genug.

Für das eben erwähnte Beispiel ist es notwendig, einen Zugriff auf die gesamte Tag-Struktur einer HTML-Seite zu erhalten. Dann lassen sich auch Start- und Zielbahnhof sauber trennen. Es muß ein Weg gefunden werden, der eine Adressierung aller Tags, beziehungsweise deren Inhalts in einer Seite ermöglicht.

Bei der Umsetzung dieser Anforderung muß darauf geachtet werden, daß HTML eine spezielle Struktur hat. In HTML gibt es auch Tags ohne Inhalt, zum Beispiel `<br>`. Trotzdem ist es notwendig, zwischen dem Bereich vor und nach dem Tag unterscheiden zu können.

Doch auch eine Adressierung auf Basis von Tags ist nicht für alle Anwendungsfälle ausreichend. Ein einfaches Beispiel für die Grenzen des Zugriffs über die Struktur sind Preisangaben. Diese werden oft im Zusammenhang mit einer Währung gemacht. Häufig wird die Angabe von Preis und Währungssymbol nur durch ein Leerzeichen, nicht aber durch ein Tag voneinander getrennt. Eine separierte Weiterbehandlung von Preis und Währung kann somit nicht erfolgen, wenn der Zugriff auf Daten nur über die Struktur der Seite möglich ist. Es muß auch möglich sein, die Inhalte der Tags weiter zu analysieren beziehungsweise zu zerlegen.

Mit einem System, welches die genannten Mechanismen zur Verfügung stellt, lassen sich eine große Menge an unterschiedlichen Daten aus verschiedenen HTML-Seiten extrahieren. Es gibt allerdings immer noch Grenzen: je weniger strukturiert die zu extrahierenden Daten sind, desto schwieriger wird es, sie zu isolieren. Insbesondere dann, wenn viele Daten ohne eine definierte Struktur, wie sie etwa bei einem Datum oder einer Zahl gegeben ist, zusammengefaßt dargestellt werden. In diese Klasse von Text fallen vor allem natürlichsprachliche Texte. Diese folgen zwar durchaus Regeln (nämlich der Grammatik ihrer jeweiligen Sprache), sind aber trotzdem – aufgrund der Komplexität der Sprache nur äußerst schwer zu verarbeiten. Das ist auch nicht Ziel dieses Systems.

HTML-Seiten sind – gemessen am Verhältnis von Information zu syntaktischen Bestandteilen – in der Art, wie sie derzeit eingesetzt werden nicht optimal für den Transport von Informationen. Es gibt immer einen Overhead an Informationen, die nur für die Darstellung und die Navigation benötigt werden. Das führt nicht nur zu unnötig komplizierten Strukturen, sondern auch zu einer Varianz der Struktur von Aufruf zu Aufruf. Als Beispiel können

hier Werbeeinblendungen genannt werden. Diese werden auf vielen Internetseiten von Aufruf zu Aufruf variieren. Mit den Einblendungen variiert auch ein Teil der Struktur der Seite. Dies erschwert den Zugriff auf die Informationen. In der Implementation eines konfigurierbaren Systems zur Extraktion muß auf derartig veränderliche Strukturen Rücksicht genommen werden.

Zusammengefaßt ist ein System notwendig, das konfigurierbar durch die Tag-Struktur einer Seite navigieren kann. Die Pfade, über die navigiert werden, müssen ausreichend flexibel beschreibbar sein, um auch bei strukturellen Schwankungen noch die richtigen Daten lokalisieren zu können. Desweiteren müssen auch innerhalb des von einem Tag umschlossenen Tags über geeignete Mechanismen Daten isoliert und selektiert werden können.

3.7 Erweiterungen von HTML

Im Laufe der Zeit wurde HTML mehrfach um neue Funktionalität erweitert. Einige dieser Erweiterungen müssen bei der Lösungsfindung beachtet werden. Neben neuen Elementen zur Definition von Struktur und Layout wird zunehmend Logik auf den Client, also in der Regel den Browser, verlagert.

Eine weit verbreitete Art Logik auf dem Client auszuführen, ist das Übertragen von in JavaScript kodierten Scripten. Mittels JavaScript eröffnen sich dem Autor einer HTML-Seite wesentlich flexiblere Möglichkeiten der Steuerung und der Benutzerinteraktion. Es ist zum einen möglich, den Browser zu instruieren, Daten nachzuladen. Diese Nachladen steht nicht im Gegensatz zu der Erkenntnis aus Kapitel 2.4. Es ist lediglich so, daß der Benutzer hier ein Teil seiner Verantwortlichkeit an den Browser delegiert. Letztendlich entscheidet also der Browser auf Anordnung des Benutzers. Nach wie vor ist keine explizite Aufforderung vom Server an den Browser möglich, da der Benutzer ebenso gut auf die Ausführung von JavaScript verzichten kann.

Desweiteren ist es möglich, daß die geladenen Dokumente verändert beziehungsweise erweitert werden können. Dabei ist insbesondere wichtig zu beachten, daß es möglich sein kann, daß die Daten in der Antwort in Javascript kodiert sind und erst durch den Browser in HTML übersetzt werden. Ein Beispiel für ein solches Vorgehen stellt – zumindest zum Zeitpunkt der Erstellung dieser Arbeit – der Dienst opodo.de [Lim] dar. Hier werden die Ergebnisse einer Flugabfrage komplett als JavaScript kodiert an den Client gesendet und erst dann vom Browser in darstellbares HTML umwandelt.[14]

[14] Selbstverständlich ist die ursprünglich übertragene Seite ebenfalls in HTML kodiert. Wird diese jedoch angezeigt, ohne daß JavaScript verarbeitet wird, wird der entscheidende Teil der Seite, nämlich die Fluginformationen, nicht dargestellt.

Neben JavaScript werden zunhemend Applikationen, wie Java-Applets (näheres auf [SM]) oder aber Flash-Applikationen (näheres auf [Lyn]), auf dem Client ausgeführt. Da der Inhalt, den diese Applikationen zur Anzeige bringen, nicht HTML-basiert ist, werden sie hier nicht weiter beachtet.

3.8 Ablaufsteuerung

In Kapitel 2.5 wurde beschrieben, wie eine Applikation im WWW funktioniert. Das hier zu erstellende System muß derartige Applikationen bedienen können, um zu einem Ergebnis gelangen zu können. Im Gegensatz zu einer klassischen Applikation ist die Steuerung des Ablaufs einer Web-Applikation komplizierter. In einer klassischen Applikation, wie sie in Kapitel 3.3 skizziert wurde, werden typischerweise einfache Funktionsaufrufe generiert, die zu einem Ergebnis kommen. Bei der Kommunikation im Web ist dies anders. Es werden mitunter mehrere nacheinander aufzurufende Seiten benötigt, um zu einem Ergebnis zu kommen. Um das zu gewährleisten, muß eine Steuerung geschaffen werden, die das Verhalten eines Menschen, der einen Browser bedient, zu simulieren.

Dabei müssen Daten teilweise auf dem Client zwischengespeichert werden, so daß die Applikation auf dem Server den Kontext der nachfolgenden Anfrage erfassen kann. Das ist ein wichtiger Punkt in Bezug auf die Ablaufsteuerung. Es muß der Abfrageapplikation möglich sein, Zugriff auf die Antwortseite zu haben. Hierbei muß darauf geachtet werden, daß nicht nur die Konversation aufrecht erhalten werden kann und somit Werte, die in Formularen an den Client gesendet wurden, weiterverwendet werden können. Es muß zusätzlich möglich sein, eine Erkennung der Art der Antwort durchzuführen, um eventuelle Fehler zu erkennen. An dieser Stelle unterscheidet sich das WWW deutlich von einer klassischen Programmierschnittstelle: entgegen der Richtlinie, daß eine Schnittstelle so zu gestalten ist, daß nur eine Art von Antwort über sie zurückgegeben wird, um eine Hybridkopplung zu vermeiden (siehe hierzu [Kah98a]), werden im WWW durchaus verschiedenartige Antworten auf dieselbe Anfrage zurückgeben. Da auf der anderen Seite der Schnittstelle ein Mensch sitzt, ist das eine durchaus sinnvolle Herangehensweise.

Die Erkennung, welcher Art eine Antwort ist, kann auf verschiedene Weisen geschehen. Zum einen gibt es die Möglichkeit, daß eine HTTP-Fehlermeldung, beziehungsweise ein von einem Ok abweichender Statuscode, zurückgegeben wird. Dieser Fall muß abgefangen und darauf entsprechend reagiert werden. Wenn es sich hierbei um einen Systemfehler seitens des Servers handelt, muß der Extraktionsprozeß unterbrochen und das Auftreten des Fehlers entsprechend weitergereicht werden. Weiterhin werden HTTP-Statuscodes im Web vielfach verwendet, um dem Client mitzuteilen, daß eine bestimmte Ressource derzeit nicht verfügbar ist, oder, daß der Zugriff verweigert wurde. Außerdem kann der Client darüber informiert werden, daß eine Ressource ”umgezogen” ist. Sie muß dann von ihrem neuen

Aufenthaltsort aufgerufen werden. Näheres zu HTTP-Statuscodes findet sich im RFC2616 [Diva].

Zum anderen ist es auch möglich, daß der Server eine Nachfrage stellt. Auf diese kann unter Umständen reagiert werden. Eine solche Abfrage ist nicht speziell gekennzeichnet. Sie ist lediglich eine HTML-Seite wie jede andere. Deswegen sollte das System so ausgelegt werden, daß es Entscheidungen auf Basis der empfangenen Daten treffen kann. In diesem Falle reichen hierfür die Daten der übertragenen Formulare nicht aus.

3.9 Allgemeine Anforderungen

Im Laufe der Zeit haben sich einige Kernanforderungen an Software-Projekte herauskristallisiert. Diese sind Portabilität, Wartbarkeit, Modularität, Wiederverwendbarkeit und Erweiterbarkeit. Gerade für die hier beschriebene Thematik ist das Beachten dieser Anforderungen sehr wichtig, da zumindest Teile der angestrebten Lösung als modulare Komponenten erstellt und benutzt werden sollen. Weitere Informationen zu diesen Themen finden sich unter anderem in [Kah98b] und [Bal96]

3.9.1 Modularität

Um einen möglichst hohen Grad an Flexibilität zu erreichen, sollte das System so modular wie möglich gestaltet werden. Damit ist nicht nur die externe Modularität, also die Fähigkeit das System als Ganzes auszutauschen, sondern auch die interne Modularität. Die Anforderung, insbesondere das Extraktionssystem als Ganzes austauschen zu können, wurde bereits in Kapitel 3.2 erwähnt. Im Allgemeinen führt ein hoher Grad an Modularität dazu, daß das System leichter zu warten ist und daß einzelne Komponenten leicht ersetzt werden können. Dies führt zu einer leichteren Erweiterbarkeit und dementsprechend zu besserer Software.

3.9.2 Wartbarkeit und Erweiterbarkeit

Im Gegensatz zu der Konfigurierbarkeit des Extraktionssystems und damit der Wartbarkeit der Anwendungen, gibt es die allgemeine Anforderung, das System so zu gestalten, daß es wartbar und erweiterbar bleibt. Es ist aufgrund der Veränderungen der Technologien im Web und aufgrund der Verschiedenheit der Einsatzorte dieses Systems nicht davon auszugehen, daß das System in einen finalen Zustand übergehen wird. Das System wird ständig erweitert und gepflegt werden müssen. Der Quelltext muß diesen Anforderungen genügen.

3.9.3 Wiederverwendbarkeit

Das System soll so gestaltet werden, daß es gut wiederverwendbar ist. Für das Extraktionssystem zumindest gilt diese Systemanforderung. Es ist aber auch hier bei dem Entwurf des Systems darauf zu achten, daß die einzelnen Subkomponenten diesem Kriterium entsprechen. Dadurch wird das entworfene System oder einzelne Teile davon für eine größere Anzahl von Anwendungen benutzbar. Dies führt zu kürzeren Entwicklungszeiten in anderen Projekten und damit im Endeffekt zu geringeren Kosten.

3.9.4 Portabilität

Die letzte Anforderung ist, daß das System portabel zu gestalten ist. Unter Portabilität wird im Allgemeinen die Fähigkeit der Übertragung eines Computerprogramms auf ein anderes (Betriebs-)System und den geschätzten Arbeitsaufwand, den eine konkrete Portierung nach sich ziehen würde, verstanden. Auch weil der Mitfahrclub bereits durch die Verwendung von Java als Programmiersprache äußerst portabel ist, sollte dies auch für die hier beschriebene Erweiterung gelten.

3.10 Zusammenfassung der Anforderungen

Aufgrund der bisher entwickelten Überlegungen läßt sich ein Anforderungsprofil erstellen. Es müssen zwei unterschiedliche Arbeiten vorgenommen werden. Zunächst muß ein Dienst analysiert werden. Mit diesem Wissen ist es möglich, eine Extraktion von Daten durchzuführen. Es wird deutlich, daß ein zweistufiges System benötigt wird. Für die erste Stufe wird ein geeignetes Werkzeug benötigt, das den Integrator dabei unterstützt, den Dienst des Fremdanbieters zu analysieren. Anschließend kann dann eine Konfiguration für die zweite Stufe des Systems erstellt werden. Um die Anforderungen einfacher beschreiben zu können, wird im Folgenden von einer Analyseapplikation gesprochen. Diese ist (zunächst) als Synonym für ein Tool oder einer Sammlung von Tools zu verstehen.

In der zweiten Stufe wird aufbauend auf die aus der Analyseapplikation stammenden Daten der eigentliche Prozeß der Datenextraktion durchgeführt. Diese Daten werden dann interpretiert und aufgearbeitet, um an das Basissystem zurückgegeben zu werden.

Das vorgestellte System muß neben den technischen Anforderungen, frei von Lizenzen sein. Der Quelltext muß frei verfügbar sein, um so einen maximalen Grad an Konfigurierbarkeit und Erweiterbarkeit zu erreichen.

3.11 Anforderungen an die Analyseapplikation

Die Analyseapplikation hat zwei Aufgaben zu bewältigen: Zum einen muß sie die Kommunikation mit einem Dienstanbieter darstellen können und zum andern muß sie aus den Antworten dieser Kommunikation, also den HTML-Seiten, eine Konfiguration für die Extraktionsapplikation erstellen.

Um eine möglichst kurze Analysezeit zu gewährleisten, sollten alle Daten in geeigneter Form aufgearbeitet werden. Das heißt insbesondere, daß nach Möglichkeit wenig HTML-Quelltext durch den Menschen gelesen (und verstanden) werden soll. Dieses gilt für die Kommunikation und für die Erstellung der Konfiguration gleichermaßen. Im einen Fall muß das Ergebnis der Kommunikation analysiert und die Struktur soweit formalisiert aufgearbeitet werden, daß daraus eine Konfiguration für die Extraktion zu erstellen ist.

Im anderen Falle ist es notwendig die Kommunikation dahingehend aufzuarbeiten, daß es einfach ist, herauszufinden welche Werte tatsächlich gesendet werden müssen, um die gewünschten Aktionen auf Seiten des Dienstanbieters durchführen zu können.

Zusammengefaßt ergeben sich folgende konkreten Anforderungen:

- Die Erkennung der Struktur einer HTML-Seite durch den Integrator muß unterstützt werden im Hinblick auf die Erstellung einer Konfiguration für die Extraktionsapplikation (Analyse der Rückgabewerte)
- Die Analyse der HTTP-Kommunikation muß ebenfalls dahingehend unterstützt werden, daß die zu übertragenden Parameter und Werte leicht festgestellt werden können (Analyse der Eingabeparameter)

3.12 Anforderungen an die Extraktionsapplikation

Aus der hier dargestellten Analyse lassen sich Anforderungen an die Abfrageapplikation ableiten. Die Abfrageapplikation muß Anfragen an einen beliebigen Webserver senden können. Dabei sollte sie nach Möglichkeit ein dem Browser ähnliches Verhalten zeigen, um einen maximalen Grad an Kompatibilität und damit auch Stabilität zu erreichen.

Die Abfragen müssen dabei kaskadierbar und jeweils einzeln interpretierbar sein. Anders ist es nicht möglich, komplexe Webapplikationen zu bedienen und von dort Daten zu extrahieren. Es muß möglich sein, das Verhalten, welches der Mensch zeigen würde, nachzustellen.

Die Applikation muß Mechanismen zur Verfügung stellen, um reproduzierbar Daten in einer HTML-Seite aufzufinden. Sie sollten ein möglichst hohen Grad an Flexibilität gewährleisten

um den vielen verschiedenen Möglichkeiten der Kodierung von HTML-Seiten begenen zu können.

- Die Schnittstelle muß Anbindung an beliebige Basissysteme ermöglichen
- Die Integration gleicher Dienstleister soll ohne Modfikation des Basissystems möglich sein
- Die Schnittstelle muß syntaktisch und semantisch eindeutige Daten liefern
- Änderungen am Seitenlayout sollen nur zu geringem Änderungsaufwand führen
- Die Lokalisierung der Daten sollte anhand der Struktur der Seiten erfolgen
- Für eine Ablaufsteuerung muß der Zugriff auf Daten aus Formularen möglich sein
- Eine Integration von neuen Diensten auch durch Nicht-Entwickler soll möglich sein

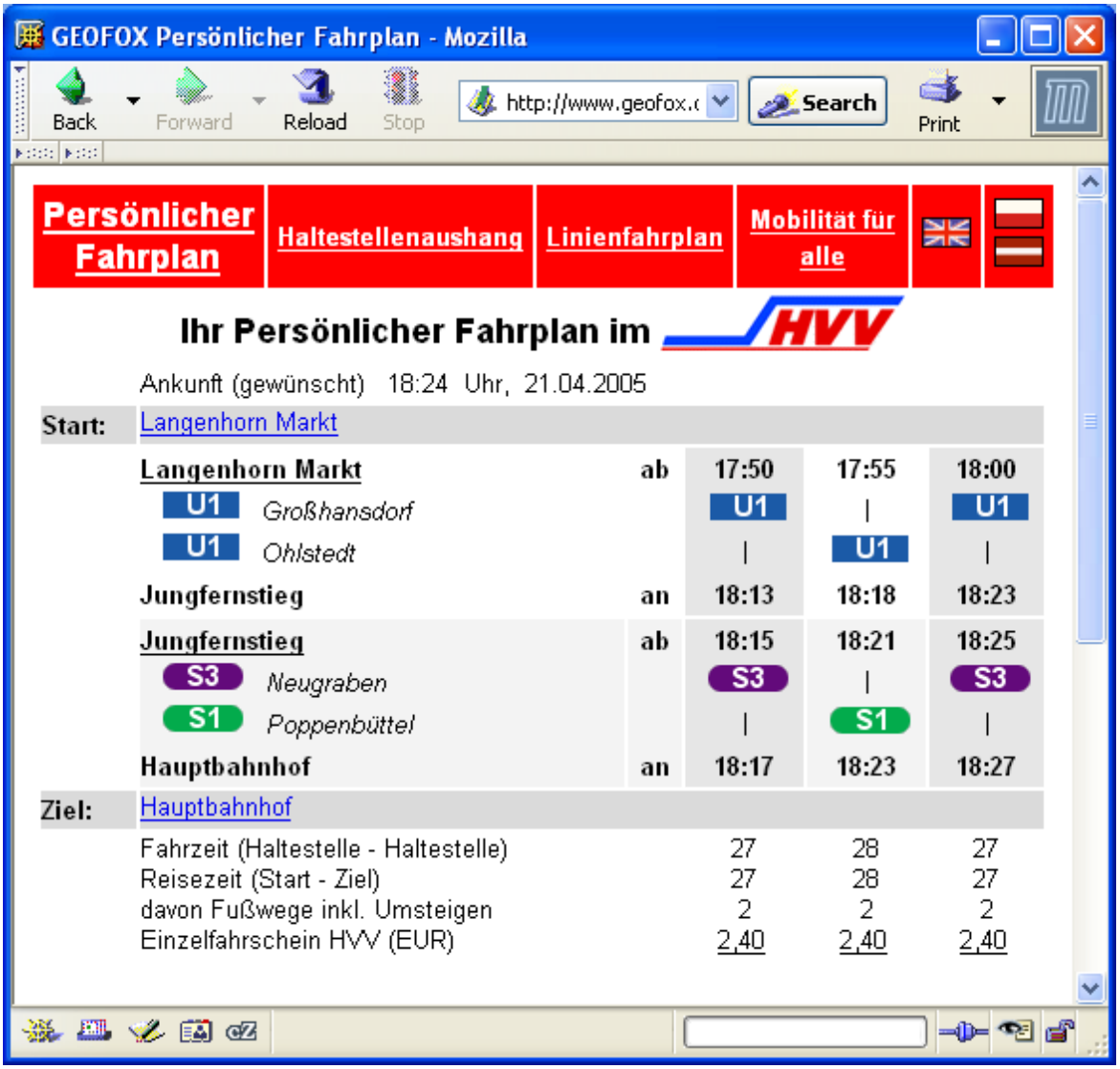

Abbildung 3.5: Persönlicher Fahrplan im ÖPNV.

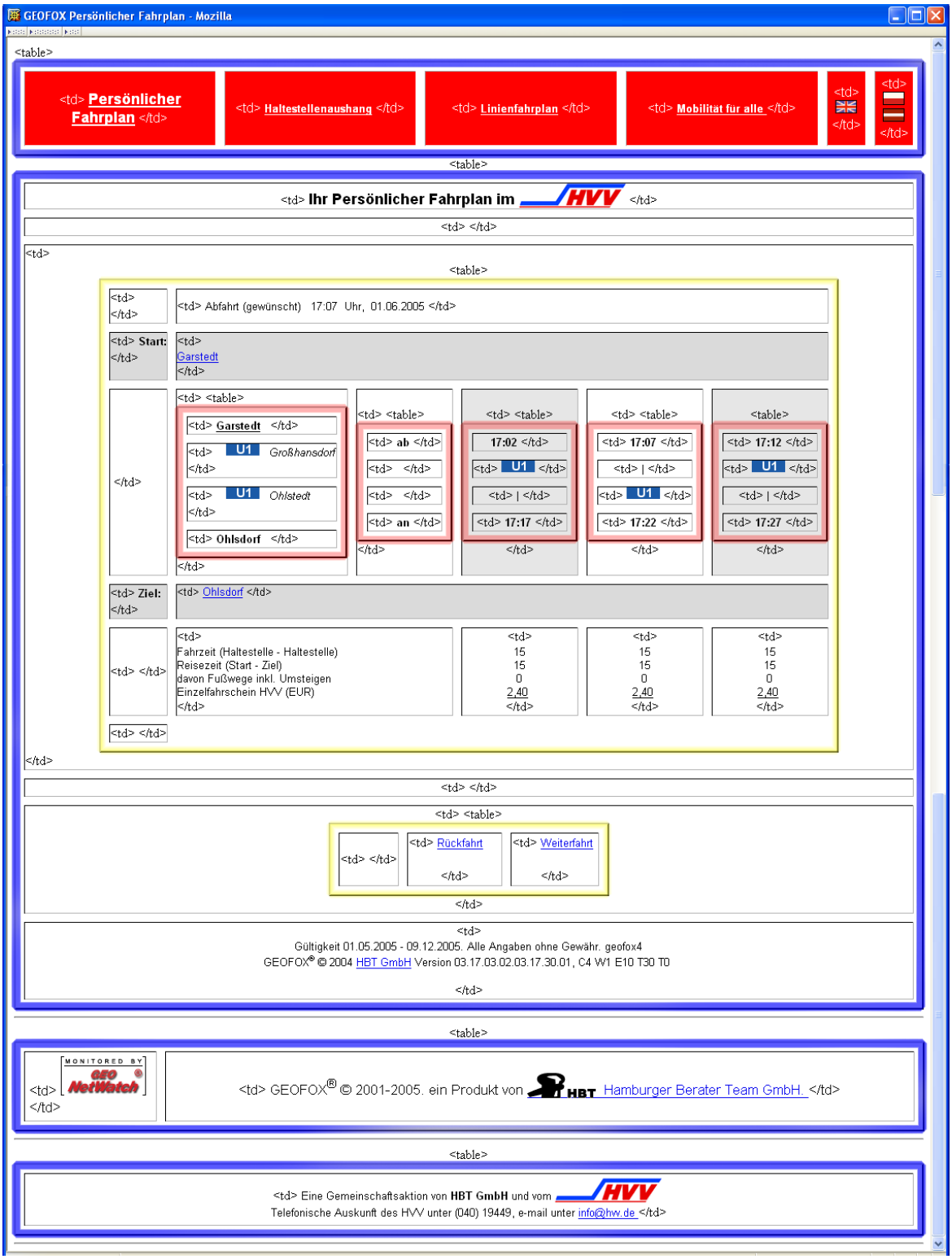

Abbildung 3.6: Darstellung der analysierten HTML-Seite

4 Design

Grundsätzlich muß das System zweistufig funktionieren: Es müssen bestehende Dienste analysiert und dann Daten aus diesen Systemen extrahiert werden. Die Erkenntnisse des ersten Schritts sollen genutzt werden, um dem zweiten Teil, die eigentliche Extraktion, durchführen zu können. Es werden zunächst Designansätze für die Analyseapplikation aufgezeigt und dann ein Vorschlag für das Design des Extraktionssystems und dessen Schnittstellenarchitektur gemacht. Ein Überblick über das System wird in Abbildung 4.1 vermittelt.

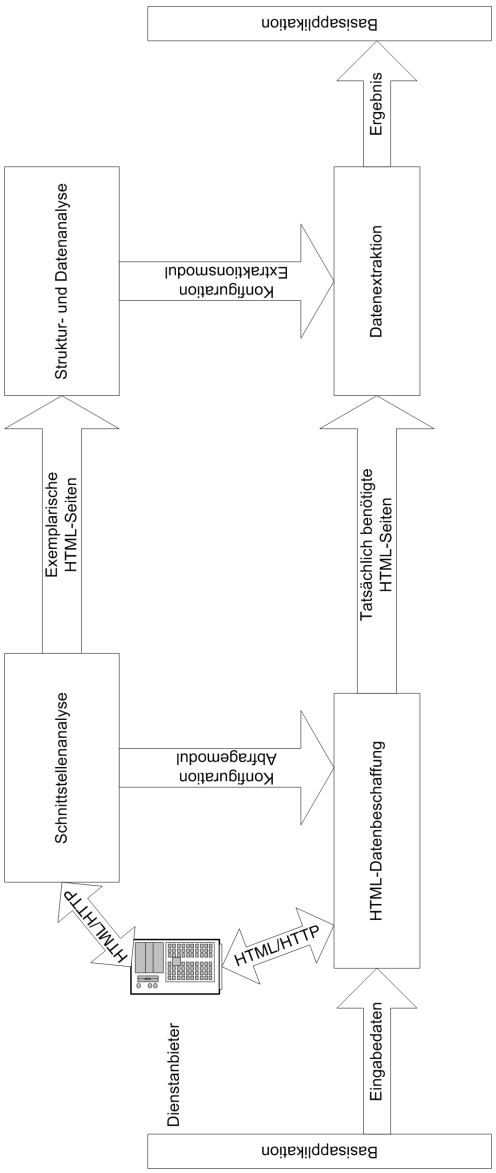

Abbildung 4.1: Vereinfachte Sicht auf das Gesamtsystem

4.1 Design des Analysetools

Aufgabe des Analysetools soll sein, dem Integrator gemäß den Anforderungen eine einfache Analyse bestehender Dienste zu ermöglichen. Daraus soll eine Konfiguration für eine Abfrageapplikation erstellt werden. Hierbei müssen zwei verschiedene Dinge untersucht werden. Zum einen muß die Kommunikation beziehungsweise Konversation analysiert werden kön-

nen. Zum anderen müssen die Antworten strukturell erfaßt werden, um erkennen zu können, wie die Daten in den Antwortseiten aufzufinden sind.

Es gibt verschiedene Ansätze zur Untersuchung der Konversation. Es ist zum einen möglich, die für die Steuerung der Konversation relevanten Daten aus den entsprechenden Seiten des Dienstanbieters zu extrahieren und zu analysieren, um daraus die Benutzung abzuleiten. Dies wird nur dann mit einem vertretbaren Aufwand möglich sein, wenn es seitens des Dienstanbieters eine umfangreiche Unterstützung gibt. Ansonsten kann eine solche Konversation – obgleich sie im Kern doch immer auf dem bereits beschriebenen Frage- und Antwort-Schema beruht – leicht kompliziert werden. Deswegen soll insbesondere im Hinblick auf die Anforderung, daß das System schnell an andere Anbieter angekoppelt werden können muß, ein anderer Ansatz als dieser Bottom-up verwendet werden.

Eine andere Art der Herangehensweise ist die Top-Down-Analyse, bei der die Konversation an der Stelle untersucht wird, die sehr schmal ist. Das ist das HTTP-Protokoll. Im vorigen Absatz wurde vom Frage- und Antwort-Schema gesprochen. Dieses ist einheitlich strukturiert und benutzt eine klar definierte und einfache Schnittstelle. Aus diesem Grunde ist es sinnvoll, daß das Analysetool an dieser Stelle eingreift. Es wird also von der strukturellen Analyse der HTML-Seiten beziehungsweise der darin enthaltenen Formulare abgesehen, und stattdessen werden nur die daraus entstehenden Anfragen betrachtet. Wie bereits in Kapitel 2.4 erläutert, kann der Server, der die Anfragen entgegennimmt, ohnehin nicht feststellen, wo diese herkommen. Es macht keinen Unterschied, ob eine Anfrage von einem Browser gestellt wird oder ob sie ein wie auch immer geartetes Programm durchführt. Generell sollte die Analyse von genügend Anfragen an einen Dienst das Muster der Abfragen offenlegen.

Der zweite Teil der Analyse beschäftigt sich mit der Struktur der Antwortseiten. Diese sind, wie in Kapitel 2.1 verdeutlicht, unter Umständen recht unübersichtlich. Daher muß die Analyseapplikation hier eine Unterstützung liefern. Da HTML-Seiten für die visuelle Dargestellung entworfen werden, soll der Einstieg zu diesem Teil der Analyse, eine dargestellte Version der zu analysierenden Seite sein. Der Integrator soll an dieser Stelle mittels der Maus auswählen können, welche Teile für ihn relevant sind. Daraufhin sollte automatisch ein Vorschlag für eine Konfiguration eines passenden Extraktors für diesen Datensatz erzeugt und dem User zur weiteren Verarbeitung zur Verfügung gestellt werden. Diese Konfiguration muß dann gegen mehrere Seiten desselben Typs, also beispielsweise Antworten auf dasselbe Eingabeformular, mit unterschiedlichen Eingaben verifiziert werden können.

4.2 Design des Extraktionstools

In diesem Kapitel wird ein Architekturvorschlag für das System zur Extraktion von Daten erstellt. Hierzu werden verschiedene Vorschläge erstellt und gegen die Anforderungen aus der Analyse geprüft.

4.2.1 Systemarchitektur

Nachfolgend werden verschiedene Vorschläge für die Integration der Extraktoren an das Basisystem vorgestellt und diskutiert. Die in Abbildung 4.2 Skizzierte Architektur sieht vor, daß es eine Vereinheitlichungsschicht gibt, an welche Extraktoren angeschlossen werden. Diese Extraktoren wiederum interagieren mit dem WWW. Das Ergebnis der Extraktionsvorgänge wird hierbei noch nicht semantisch aufbereitet. Das bedeutet, daß ein Datum nach wie vor als Zeichenkette mit einem beliebigen Format vorgelegt wird. Diese könnte beispielsweise ’11.2.2002’ sein, oder aber ’11Feb2002’. Es wird lediglich in eine einheitliche Struktur (zum Beispiel ein Java-Objekt) gebracht, in welcher jede abgefragte Information auf einem für alle Anbieter einheitlichen Weg abzufragen ist. In der Vereinheitlichungsschicht werden die Daten dann an der entsprechenden Stelle ausgelesen und interpretiert.

Dieses Vorgehen weist eine Reihe von Vorteilen auf: Dadurch, daß die Schnittstelle zwischen den Extraktoren für alle Dienste gleich ist, ist es ohne Probleme möglich, weitere Dienste anzuschließen. Sie müssen lediglich die gewünschte Datenstruktur liefern. Außerdem werden die Extraktoren – da die Daten nicht umgewandelt werden müssen – vergleichsweise einfach aufgebaut sein. Sie müssen lediglich die Daten in der HTML-Seite finden und diese in eine neue, einheitliche Datenstruktur überführen.

Das legt die Überlegung nahe, ob die Extraktoren nicht komplett ohne weiteren Kodierungsaufwand erstellt werden können. Ob das in der Praxis so ist, kann nicht generell bejaht werden. Letztendlich wird es in den meisten Fällen zwar mit vergleichsweise wenig Logik möglich sein, die Daten aufzufinden. Schwieriger ist die strukturelle Transformation, also das Schreiben der aufgefundenen Daten in das Zwischenformat. Um hier nicht zu eingeschränkt zu sein, sollten die Extraktoren zumindest die Möglichkeit bieten, um eine Logik erweitert zu werden.

Die in Kapitel 4.2 angedeutete Schnittstelle zur Applikation kann relativ frei gewählt werden. Hier werden bereits vom Programm interpretierbare Fachwerte übertragen. Es muß lediglich dafür Sorge getragen werden, daß alle von irgendeinem Dienstanbieter benötigten Daten übergeben werden können. Das bezieht sich nicht nur auf die Rückgabewerte aus der Vereinheitlichungsschicht, sondern vor allem auf die Eingabewerte. Diese unterliegen im Zweifel

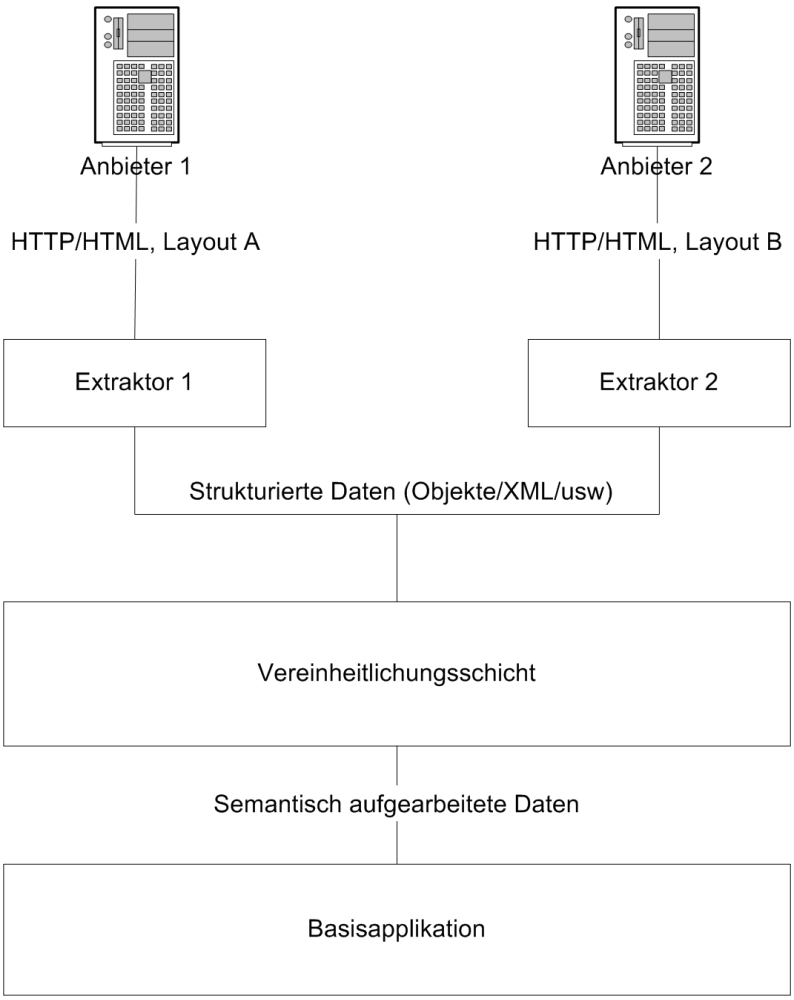

Abbildung 4.2: Architektur-Vorschlag 1 zur Integration externer Dienste

größeren Schwankungen, da unterschiedliche Anbieter unterschiedliche Informationen benötigen, um ein Angebot liefern zu können.

Obgleich dieses Vorgehen universell zu sein scheint, erfüllt es doch nicht alle Anforderungen. Das Problem der Transformation von Daten in einheitliche Fachwerte wurde hier aus den Extraktoren in die Transformationsschicht verschoben. Das setzt voraus, daß diese universell genug arbeiten kann, um das zu leisten. Bei einfachen Datentypen kann das noch funktionieren, aber es kann nicht garantiert werden, daß jede Art von Datum aufbereitet werden kann. Spätestens bei solchen Werten, für die es vom Anbieter gewählte Bezeichnungen gibt, wie etwa den Namen von Bahnhöfen, würde diese Schicht schon mit einer künstlichen Intelligenz ausgerüstet werden müssen, um die verschiedenen Daten auf Grund bestimmter Annahmen zu interpretieren. Auch solche Werte, die einem festen Übersetzungsschema folgen, wie etwa dem Wandel einer Typenangabe in ein für das Programm verständlichen Wert,

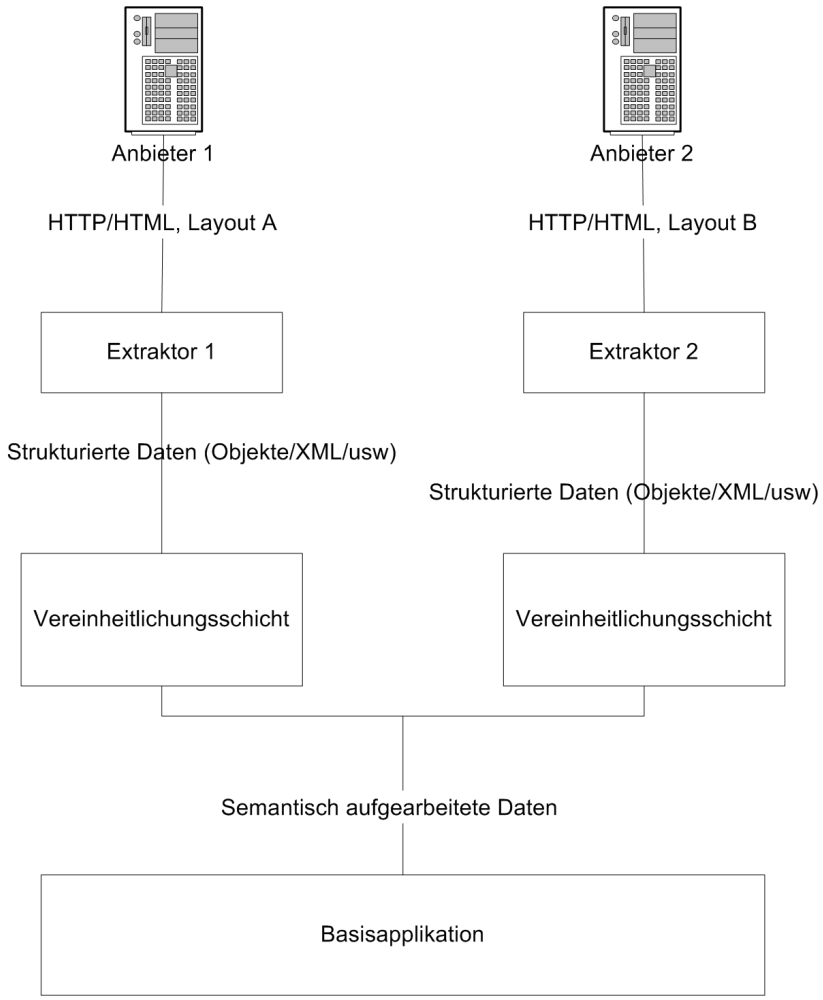

Abbildung 4.3: Architektur-Vorschlag 2 zur Integration externer Dienste

können nicht korrekt interpretiert werden. Beispielsweise ist es möglich, daß die Klasse einer Flugverbindung übergeben werden soll (Economy beziehungsweise Business Class). Im Programm gibt es für derartige Informationen in der Regel bestimmte Schlüssel, also etwa die Konstante ’1’ für Economy und die Konstante ’2’ für Business. In den HTML-Seiten des Anbieters wird sich dafür eine vergleichbare Information finden, die aber nach einem festen, jedoch anbieterspezifisch unterschiedlichen Schema ersetzt werden muß.

Aus den eben genannten Einschränkungen ergibt sich, daß eine solche Architektur nur selten sinnvoll einsetzbar ist. Eine veränderte Variante wird in Abbildung 4.3 vorgestellt. Hier wird pro Anbieter jeweils eine Vereinheitlichungsschicht benutzt. Dieses Vorgehen ist die flexibelste Variante, da jede Vereinheitlichungsschicht komplett dem zugehörigen Dienst angepaßt werden kann. Nachteilig ist, daß zum einen Kodierungsaufwand entsteht sowie ein neuer Dienst (gleichen Typs) hinzukommt. Es ist relativ wahrscheinlich, daß Komponenten

mehrfach verwendet und damit bei einer strikten Trennung der Vereinheitlichungsschichten untereinander auch mehrfach entwickelt werden müssen.

Diese beiden Nachteile leiten zu einem erweiterten Ansatz, welcher in Abbildung 4.4 dargestellt wird. Hierbei wird davon ausgegangen, daß jeder Übersetzer einer Zeichenkette in einen semantischen eindeutigen[1] Wert in eine Sammlung von Tools aufgenommen wird, die dann als Bibliothek zur Verfügung steht. Auf diesem Weg ist die maximale Wiederverwendbarkeit der einzelnen Komponenten gewährleistet, ohne daß die Möglichkeit besteht, daß das Anschließen eines neuen Dienstes an das System eine Veränderung an der Integration eines anderen Dienstes nach sich zieht. Die Vereinheitlichung ist somit modular zerlegbar.

Es stellt sich die Frage, ob man nicht auf die Vereinheitlichungsschicht komplett verzichten und die dort verankerten Operationen direkt in die Extraktoren übernehmen kann. Ein solches Design ist möglich, sollte aber nur dann gewählt werden, wenn mit nicht vielen Veränderungen des Layouts zu rechnen ist. Wird auf die Trennung von Extraktoren und Vereinheitlichungsschicht verzichtet, besteht die Gefahr, daß die Vereinheitlichung und die Extraktion zu dicht gekoppelt werden und damit letztendlich nur aufgrund einer Veränderung des Layouts auch die Vereinheitlichungsschicht verändert werden muß. Dieses kann man zwar auch bei einer Trennung nicht komplett ausschließen, da die Darstellung der Information, aus der sich die Extraktoren bedienen, beliebig aussehen kann. Es ist allerdings eher wahrscheinlich, daß die Informationen, einmal isoliert betrachtet, sich auch nach einer Layoutveränderung nicht wesentlich ändern. Letztendlich wird ein Datum in der Praxis nur in wenigen unterschiedlichen Formaten dargestellt.

In Bezug auf den Mitfahrclub ist eine der Herausforderungen die Einhaltung des Prinzips der gegenseitigen Nicht-Beeinflussung ([Kah98b]). Hier werden Adressen und somit sehr unterschiedliche und nicht immer einheitliche Daten zwischen den Systemen ausgetauscht. Diese haben, abhängig vom Land dem sie zugeordnet sind, unterschiedliche Formate. Außerdem ist eine Adresse schwer eindeutig zu identifizieren, da sich die eigenen Teile in der Darstellung unterscheiden können. Das bereits erwähnte Problem des Hauptbahnhofs, welcher auch HBF abgekürzt werden kann, erstreckt sich auch auf Adressen. Eine Straße beispielsweise kann als ”Str”. abgekürzt zurückgegeben werden. Es gilt also, im Falle des Mitfahrclubs darauf zu achten, daß die ausgetauschten Adressen untereinander verstanden werden. Da der Mitfahrclub derzeit mit Adressen arbeitet, wird es auch einen Mechanismus

[1] Semantisch eindeutig ist hier im Sinne der Beschreibungen aus Kapitel 3.2 zu verstehen: Ein Wert ist genau dann semantisch eindeutig wenn er – egal aus welchem Teil des Softwaresystems er kommt – bei einem geeigneten Vergleich als gleich erkannt wird, sofern der Wert in der realen Welt derselbe ist. Der Hamburger Hauptbahnhof ist immer der Hamburger Hauptbahnhof, unabhängig davon, ob ihn der Dienstanbieter nun mit ’HBF’ abkürzt, oder den Namen ausschreibt. Ab dem Punkt, wo die Information ’Hamburg Hauptbahnhof’ semantisch vereinheitlicht wurde, gibt es nur noch genau einen Wert für ’Hamburger Hauptbahnhof’. (Dieser muß nicht einmal eine Zeichenkette, sondern könnte auch ein Identifikator sein.)

geben, der Adressen verwalten und interpretieren kann. Diesen zu nutzen, um eine Unifizierung der Adressen durchzuführen, ist problematisch, da eine Rückkopplung vom Extraktor zum Mitfahrclub vermieden werden soll. Dies würde in einer Veränderung des Verhaltens des Extraktors bei einer Änderung des Mitfahrclubs führen.

Eine mögliche Lösung dieses Problems stellt die Nutzung der Geo-Engine dar. Diese sollte die verschiedenen Arten von Adressen verstehen können und diese vereinheitlicht – z.B. als Geo-Koordinate – ausgeben können. Diese Aufarbeitung darf allerdings nicht innerhalb der Basisapplikation oder des ÖPNV-Subsystems geschehen. Stattdessen sollte die Vereinheitlichung – ebenfalls gemäß des Prinzips der modularen Zerlegbarkeit – von einem getrennten Modul vorgenommen werden. So wird vermieden, daß die Basisapplikation modifiziert werden muß, wenn ein neuer Dienst hinzukommt, der eine Modifikation am entsprechenden Modul benötigt. Gegebenenfalls kann das Modul auch komplett ersetzt werden.

4.2.2 Die Extraktionsschicht

Im vorangegangenen Kapitel wird wiederholt von Extraktoren gesprochen. Diese übernehmen die Aufgabe, Daten aus einer HTML-Seite zu extrahieren. Gleichzeitig müssen sie die Daten in eine einheitliche Datenstruktur überführen. Die interne Struktur der Extraktoren ist in Abbildung 4.5 dargestellt.

Die konkrete Implementation dieser Extraktoren hängt dabei von der Art des Extraktionsmechanismus ab. Mindestens dieser Teil des Extraktors muß weitgehend konfigurierbar sein. Ein Vorschlag für einen solchen Extraktionsmechanismus wird in Kapitel 4.2.3 skizziert. Im optimalen Fall sollte also beim Einbinden eines neuen Layouts (sei es von einem neuen Anbieter oder durch eine Veränderung des Layouts bei einem bestehenden Anbieter) eine Rekonfiguration dieses Teils des Systems ausreichen, um eine vollständige Integration des geänderten Layouts vorzunehmen.

Bei der Implementation des Transformators muß beachtet werden, daß die Transformationen sehr verschieden (wohlgemerkt lediglich die strukturelle und nicht die inhaltliche Transformation) sein können. Aus diesem Grund muß hier ein System gewählt werden, welches ein maximales Maß an Flexibilität bietet.

Ein Beispiel: in den bisherigen Beispielen wurde davon ausgegangen, daß Fahrplaninformationen in Tabellen dargestellt werden, bei denen eine Zeile jeweils genau eine Fahrtmöglichkeit darstellt. Der Anbieter [Akt] geht hier einen anderen Weg: Es werden Hin- und Rückflüge als Matrix dargestellt. Die erste Spalte enthält hierbei jeweils die Informationen über den Hinflug und die erste Zeile jeweils die Informationen über den Rückflug. Das Innere der Matrix enthält nur noch Informationen über die Verfügbarkeit von Plätzen in den entsprechenden Flügen. Bei der Implementation des Transformators müssen diese Kombinationen

als Kreuzprodukt aufgelöst werden, damit jede Kombination aus Hin- und Rückflug genutzt werden kann. Das ist eine gänzlich andere Operation als das zeilenweise Analysieren einer Tabelle, wie es zum Beispiel für [AG] zu nutzen wäre.

Im Prinzip muß der Konverter mindestens so mächtig wie gängige Programmiersprachen, wie etwa Java, sein, da die Applikation, die die Eingabedaten liefert, dies normalerweise auch ist.[2] Um der allgemeinen Anforderung der Wartbarkeit zu genügen, sollte davon abgesehen werden, 'das Rad neu zu erfinden' und eine eigene Sprache für diesen Zweck zu konzipieren. Es gibt zum Zeitpunkt der Erstellung dieser Arbeit bereits eine große Anzahl von Programmiersprachen (sowohl direkt interpretierbare, als auch kompilierbare), die für einen solchen Einsatz ausreichend sind. Diese sind wohlbekannt. Insofern ist die Wahl einer solchen Sprache der Wartbarkeit und Zukunftssicherheit des Systems zuträglich. In der gewählten Sprache sollten Module erstellt und benutzt werden, die miteinander kombiniert werden können. Ein universeller Transformator, welcher durch Konfiguration an neue Umfelder angepaßt werden kann, ist somit nicht möglich es sei denn, die Konfiguration ist derartig komplex, daß sie einer Programmiersprache gleichzusetzen ist; genau das aber sollte dann durch eine bestehende Sprache erledigt werden.

[2] Es gibt in der Praxis eine große Auswahl verschiedenartiger Implementationen von Web-Applikationen und Frameworks. Inwieweit insbesondere die Frameworks in Bezug auf ihre Ausgabelogik die Mächtigkeit einer vollwertigen Programmiersprache haben, kann nicht allgemein beantwortet werden. Die Tatsache, daß es möglich ist, mit so gut wie jeder Programmiersprache auch Webapplikationen zu erstellen, hat zur Folge, daß zur Transformation eine Sprache mit solch einer Mächtigkeit benötigt wird.

4.2.3 Extraktion der Daten

In Kapitel 3.6 wird die Notwendigkeit beschrieben, ein System zu schaffen, welches Daten in einer HTML-Seite auffindbar macht. Dieses System muß konfiguriert werden können, so daß die Anforderung der Konfigurierbarkeit des Extraktionssystems erfüllt ist. Das Problem teilt sich hier also in zwei Teile auf: Es muß ein System gefunden werden, in dem sich zum einen die Struktur einer Seite abstrakt beschreiben läßt und zum welches anderen ermöglicht, auf die Struktur zuzugreifen.

Die Anforderung, eine HTML-Seite strukturiert zu erfassen, ist nicht neu. Hierzu wurde vom W3C die Document Object Model (kurz DOM)-Schnittstelle vorgeschlagen. Diese sieht vor, daß die einzelnen Teile eines Dokuments als Objekte und Unterobjekte interpretiert werden. Daraus ergibt sich eine baumartige Struktur. [3] Die einzelnen Tags innerhalb einer HTML-Seite werden hier als Knoten interpretiert, die gegebenenfalls Unterobjekte enthalten. Die Darstellung Abbildung 4.6 illustriert dies. Ob in der konkreten Implementierung tatsächlich auf das DOM aufgebaut wird, soll hier nicht festgelegt werden. DOM ist lediglich ein vielversprechender Ansatz um das Problem der Strukturierung zu lösen. Der Quelltext zu Abbildung 4.6 sieht folgendermaßen aus (kein gültiger HTML-Code):

```
<html><body>
<tr> <td>Abfahrt:</td> <td>12:34</td></tr>
<tr> <td>Abfahrt:</td> <td>13:45</td></tr>
</body></html>
```

Wird eine baumartige Struktur wie sie durch ein DOM gegeben ist, eingesetzt, so wird das konfigurierbare Auffinden der Daten in der Struktur, reduziert auf die Frage, wie der Pfad durch den Graphen, welcher den Baum darstellt, beschrieben werden kann. Es kann pro Knoten angegeben, welche der Unterknoten auszuwählen sind, um zu der gewünschten Information zu gelangen. An dieser Stelle lassen sich – falls benötigt – auch Kriterien, die auf Basis des Inhalts der Knoten oder deren Unterknoten basieren, einsetzen.

Auch für die Erfüllung der zweiten Anforderung gibt es einen Vorschlag vom W3C, die XPath-Sprache ([CD]). XPath wird beschrieben als

> ”The primary purpose of XPath is to address parts of an XML [XML] document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on

[3] Das W3C macht keine Angaben dazu, wie die Implementationen intern funktionieren sollen. Dementsprechend wird intern notwendigerweise mit einem Baum gearbeitet.

> the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document."

[CD]

Ursprünglich wurde XPath zur Verwendung mit XSLT verwendet. Ein XPath ist erinnert ein wenig an die Darstellung von Verzeichnissen. Vereinfacht besteht ein XPath aus einer durch Schrägstrichen getrennten Hierarchie von Adressierungsbedingungen. Diese können beispielsweise die Namen der Tags tragen (etwa `/tr[42]/td[2]`) oder aber Universaloperatoren, wie etwa einem `'*'`. Außerdem können weitere Bedingungsfunktionen in eckigen Klammern angegeben werden. Im eben genannten Beispiel sind das die Indizes. Das Beispiel greift also auf die 42. Zeile und die zweite Spalte einer Tabelle zu. Genauere Informationen finden sich auf [CD] und [Div02].

4.2.4 Ablaufsteuerung

Aus der Analyse ergibt sich, daß es notwendig sein kann, mehrere Anfragen zu tätigen, um an das gewünschte Ergebnis zu kommen. Das System muß daher die Möglichkeit bieten mehrere Anfragen hintereinander zu senden. Zur Analyse der Antworten muß ein ebenfalls konfigurierbarer Mechanismus geschaffen werden. Es ist hierbei zwischen zwei Arten von Daten zu unterscheiden: die Daten aus Formularen und die Daten aus der Seite selbst. Formulardaten werden benötigt, um den Aufruf des nächsten Schritts mit Parametern zu versehen und müssen somit in die nachfolgende Anfrage eingebaut werden können.

Der eigentliche Inhalt der Seite sollte ebenfalls analysierbar sein. Von diesem Teil der Daten ist es aber nicht notwendig, Daten in die nächste Abfrage einfließen zu lassen, da dies nicht von HTML vorgesehen ist.

Im System ist, wie aus Kapitel 4.2.3 ersichtlich, ohnehin eine Möglichkeit zur Extraktion von Daten vorhanden. Diese sollte auch genutzt werden, um die vorliegenden Daten analysieren zu können. Bei der Implementation diese Systems muß darauf geachtet werden, daß auch die Inhalte, beziehungsweise Parameter aus den Formularen extrahiert werden können.

4.2.5 Integration der Extraktionssysteme

Um mit einem System, welches den hier beschriebenen Designanforderungen entspricht, einen neuen Dienst gleichen Typs an das Basissystem anzuschließen, ist es notwendig, drei Module zu erstellen oder zu konfigurieren.

- einen Extraktor
- einen Transformator
- eine Vereinheitlichungsschicht

Es wird eine Konfiguration zur Lokalisierung der Daten innerhalb der HTML-Seite erstellt. Auf dieses so konfigurierete Extraktionssystem wird ein zu erstellender Transformator aufgesetzt. Im letzten Schritt wird ein Modul zur Vereinheitlichung der Daten geschaffen. Hierfür müssen gegebenenfalls weitere Tools zur Interpretation beziehungsweise Umwandlung der Daten geschaffen werden.

In einem Modul zur Extraktion von Daten aus dem ÖPNV, wie es für den Mitfahrclub benötigt wird, bedeutet dies, daß...

- der Extraktor Abfahrts- und Ankunftszeiten in einer HTML-Seite lokalisiert und in Listen speichert

- der Transformator aus diesen Listen Datenpaare erstellt, die
- in der Vereinheitlichungsschicht interpretiert und in Datumsobjekte (in Java `java.util.Date`) überführt werden, um dann in einer komplexen Datenstruktur an das Basissystem zurückgegeben zu werden.

Sollte sich das Layout ändern, aber die generelle Darstellung der Ergebnisse und das Format der Datumsdarstellung nicht verändern, so ist nur die Konfiguration des Extraktors zu verändern. Sollte sich die generelle Darstellung verändern (also etwa eine matrixartige im Gegensatz zu einer zeilenorientierten Struktur eingeführt werden), so muß die Extraktor- und die Transformator-Konfiguration verändert werden. Verändert sich auch das Datumsformat, so muß auch die Vereinheitlichungsschicht geändert werden.

4.3 Zusammenfassung

Die zentralen Designmerkmale, die hier vorgeschlagen werden, können wie folgt zusammengefaßt werden.

Für die Analyseapplikation:

- Die Analyse der HTML-Seiten muß visuell erfolgen
- Um die Kommunikation zu analysieren, muß die tatsächliche Kommunikation untersucht werden

Für die Abfrageapplikation:

- Einführung einer Zwei-Schichten-Architektur, um Extraktion und Überführung der Daten in ein einheitliches Format zu trennen
- Extraktion muß mindestens leicht konfigurierbar sein
- Benutzung des DOM, um Zugriff auf Elemente der HTML-Seite zu erhalten
- Benutzung bestehender Programmierwerkzeuge zur Erstellung der Transformatoren

Diese Designvorschläge sollen in der Realisierung wieder aufgegriffen werden.

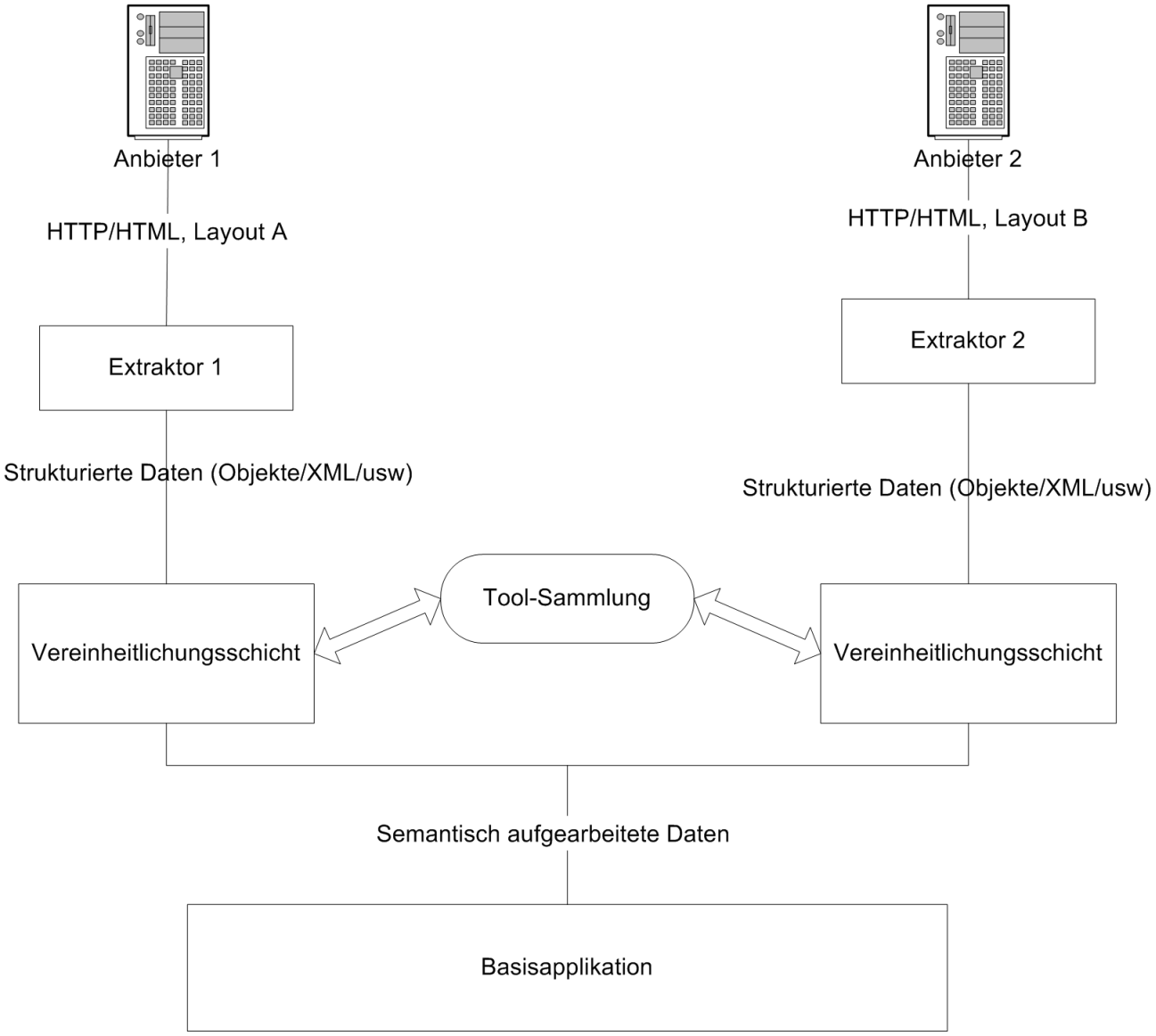

Abbildung 4.4: Architektur-Vorschlag 3 zur Integration externer Dienste

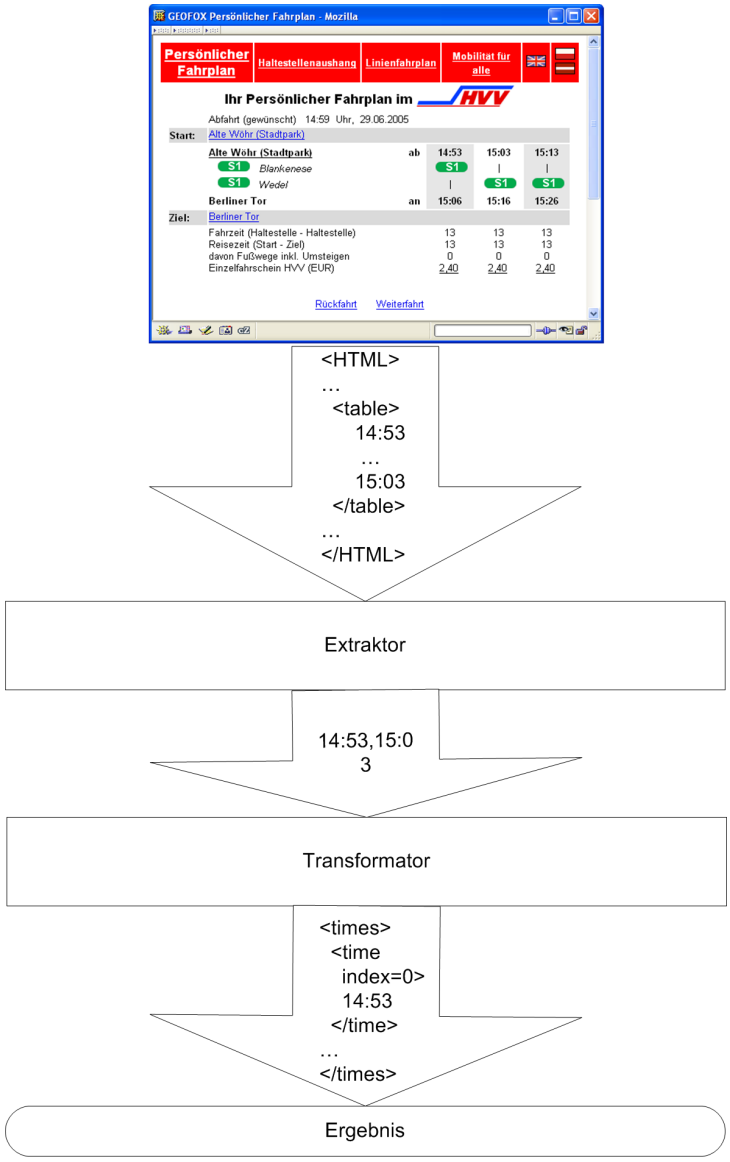

Abbildung 4.5: Struktur der Extraktoren

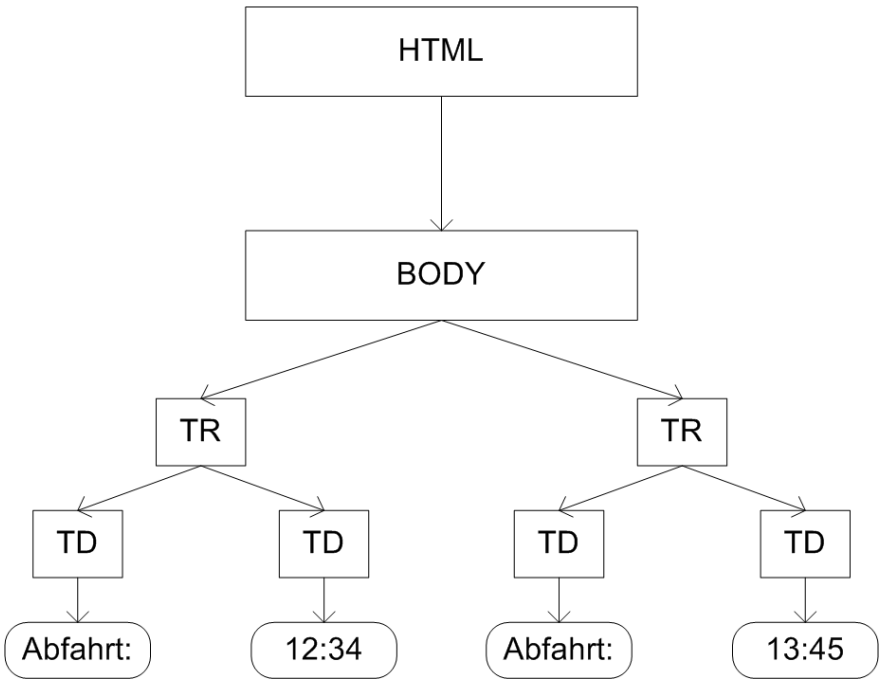

Abbildung 4.6: Beispiel eines DOM-Baumes

5 Vorhandene Ansätze

Im Folgenden werden zwei Ansätze zur Lösung des beschriebenen Problems vorgestellt. Nur einer von diesen ist derzeit als konkret implementierte Applikation vorhanden. Zunächst wird ein Überblick über das jeweilige System vermittelt, um die Systeme dann gegen die bislang erarbeiteten Anforderungen und architekturellen Vorschläge zu prüfen.

5.1 Celcorp WebRecorder

Ein vielversprechender Ansatz zur Extraktion von Daten aus dem WWW stellt das Produkt WebRecorder der Firma Celcorp dar. Der WebRecorder ist eine Suite von Programmen, die es ermöglichen, Extraktor-Systeme für Daten aus dem WWW zu erstellen. Im Kern werden hier Makros erzeugt, die später aus dem jeweiligen Basissystem ausgeführt werden können. Das gesamte System arbeitet konsequent mit Objekten.

Das Celware System ist ein zweistufiges System, das – wie hier in der Analyse vorgeschlagen – aus einer Analyse und aus einer Extraktionsapplikation besteht. Die Extraktionsapplikation läßt sich als Bibliothek in eigene Programme integrieren, so daß eine eigenständige Applikation nicht benötigt wird.

Um mit dem WebRecorder Daten aus dem WWW zu extrahieren und diese in eine Applikation zu importieren, muß der User zunächst eine oder mehrere Testanfragen an den gewünschten Dienstanbieter senden. Hierzu stellt das Programm einen eigenen Browser zur Verfügung. Alternativ kann auch ein beliebiger anderer Browser benutzt werden, sofern dieser die Möglichkeit bietet, einen Proxy-Server zu benutzen. Den Proxy stellt in diesem Falle das Analyseprogramm dar. Die Anfragen und die daraus resultierenden Antworten werden als so genannte Trace-Objekte dargestellt.

Im nächsten Schritt folgt die Analyse der Daten. Hierzu werden einzelne Anfragen und deren Antworten untersucht. Das Analyseprogramm zeigt die HTML-Seite an. Um Daten aus der Seite zu extrahieren, müssen zunächst sogenannte DataObjects angelegt werden. Diese Datenobjekte beinhalten wiederum so genannte DataMembers, welche als Träger der eigentlichen Daten fungieren. Es ist möglich, DataObjects mit anderen DataObjects in einer Vater-Sohn-Beziehung zu verknüpfen.

Um die erstellten DataObjects mit Daten zu befüllen, werden die zu extrahierenden Stellen aus der dargestellten HTML-Seite markiert. Sie werden dann mit den entsprechenden DataMembers verknüpft, beziehungsweise die Daten werden übernommen. Intern wird hierzu der XPath zu den jeweiligen Daten in der Seite erzeugt und abgespeichert. Für komplexere Übersetzungen wird hier außerdem die Möglichkeit gegeben, XSLT-Code einzubetten. Er kann eine Tag-Struktur erzeugen, die in die DataObjects eingebettet und als DataMember interpretiert wird. Die eben beschriebenen Arbeitsschritte werden benutzt, um Extracts zu erstellen. Diese fassen die zur Extraktion der Daten notwendigen Schritte zusammen. Alle DataObjects werden im so genannten DataStore gesammelt. Dieser scheint intern als XML-artiges Dokument[1] aufgebaut zu sein, so daß es möglich ist, einfach neue Knoten hinzuzufügen. Auch listenartige Strukturen sind – über XSLT – realisierbar.

Extracts und Traces werden zu einem Process-Objekt zusammengefaßt. Dieses beinhaltet Methoden, die wiederum die Anfragen und Extract-Objekte vereinigen. Innerhalb einer Methode gibt es Instruktionen. Über diese Instruktion kann der Extraktionsprozess gesteuert werden. Es gibt einen Satz von Basisinstruktionen, der zum einen die schon erwähnten Extracts enthält und zum anderen rudimentäre Kontrollstrukturen (Wenn-dann-Abfragen und Iterationen). Neben einigen weiteren Operationen besteht hier die Möglichkeit, JavaScript Code mit in die Methoden einfließen zu lassen, um so noch flexibler in den Abarbeitungsprozeß eingreifen zu können.

Die erstellten Methoden werden aus der Basisapplikation heraus gestartet. Dazu können zunächst – sofern vorhanden – Eingabeparameter an die Methoden gegeben werden. Danach werden die Methoden vom CelWebPlayer ausgeführt. Er ist als Softwarekomponente für verschiedene Programmiersprachen vorhanden.

Nachdem die Methode vom Player ausgeführt wurde, liegen die Ergebnisse im DataStorage. Innerhalb der Basisapplikation kann nun über den in der Extract-Definition definierten Pfad auf die konkreten Werte zugegriffen werden.

5.1.1 Kritik an diesem Modell

Das Produkt der Firma Celcorp stellt eine gelungene Implementation eines Lösungsansatzes dar, der sich mit den in Kapitel 4.2.1 beschriebenen Ansätzen vereinbaren läßt. Es überführt – wie dort vorgeschlagen – die Daten in ein Zwischenformat. Dieses kann für verschiedene Anbieter einheitlich sein. Außerdem wird eine relativ flexible Ablaufsteuerung und ein äußerst flexibles Extraktionssystem bereitgestellt. Es basiert auf erweiterbaren Technologien,

[1] Leider ist die Dokumentation sehr anwenderorientiert. Eine genaue Untersuchung der Interna des Produkts würde den Rahmen dieser Arbeit sprengen

die – durch den Einsatz der XSLT-Programmiersprache – mächtig genug sind, um der in Kapitel 4.2.2 Anforderung zu genügen. Es wird außerdem eine Architektur angeboten, die es ermöglicht, das System als Komponente in andere Systeme zu integrieren. Da der gesamte Extraktionsvorgang skriptbasiert ist, wird es weitgehend nicht notwendig sein, bestehenden Programmcode im Basissystem oder dem Extraktionsadapter, den das CelCorp-System benutzt, zu verändern.

Es ergeben sich durch die beschriebenen Vorgehensweisen auch Nachteile. In Kapitel 4.2.1 wird vorgeschlagen, die extrahierten Daten vorzuverarbeiten und sie in ein einheitliches Modell zu überführen. Dies ist mit dem CelCorp möglich. Die hierzu benötigte Logik wird – sofern sie einfach ist – komplett von den vorgefertigten Funktionen im Programm übernommen. Bei komplizierteren Prozessen jedoch muß der User jedoch auf XSLT-Transformationen zurückgreifen. Das alles wird unter einer grafischen Benutzeroberfläche versteckt. Dieses Vorgehen ist zwar weit von einer komplett in einer Programmiersprache erstellten Lösung entfernt, aber es droht dadurch auch an Übersichtlichkeit zu verlieren.

Dasselbe gilt für die Steuerung der Anfragen und des Abfrageprozesses. Es werden diverse Möglichkeiten in Form fester Instruktionen gegeben, die zusätzlich um Skripte erweitert werden können. Diese Mechanismen sind zwar flexibel, aber ob es der Übersichtlichkeit dienlich ist, wird sich erst in der praktischen Anwendung zeigen müssen.

Weitere Nachteile dieser Lösung sind die eingeschränkten Möglichkeiten der Fehlersuche, aufgrund der nicht vorhandenen Quelltexte zum System. Im konkreten Test hat das mit dem System mit gelieferte Beispielprogramm leider nicht funktioniert. Es hat lediglich den Computer voll ausgelastet, aber kein benutzbares Ergebnis gebracht.[2]

Im Zuge der Tests ist aufgefallen, daß das System Schwächen bei der Behandlung von Seiten, die mit JavaScript arbeiten, hat. Es kann nicht den tatsächlich vom Browser angezeigten Quelltext analysieren. Dieser weicht unter Umständen von dem vom Server gelieferten Quelltext ab, da er dynamisch durch JavaScript erweitert werden kann. In diesem Falle bleibt dem Benutzer[3] des CelCorp-Systems nichts anderes übrig als den JavaScript-Quelltext zu analysieren und Daten hieraus zu extrahieren. Dieser Quelltext ist zwar in einem bestimmten, klar definierten Format dargestellt, kann aber dennoch nicht so einfach aufgearbeitet werden wie HTML. Das liegt zum einen daran, daß die Syntax komplizierter ist und zum anderen auch daran, daß unter Umständen die Informationen erst zusammengesetzt werden müssen.

Neben den eben beschriebenen Kritikpunkten am System gibt es noch einige Kritik an der Benutzbarkeit. Die Dokumentation ist teils sehr schlecht. (Bildschirmausdrucke, die entscheidend für das Verständnis sind, sind wenig oder gar nicht erkennbar). Außerdem ist die GUI

[2] Es sei darauf hinweisen, daß die Versuche, das Testsystem lauffähig zu machen, nicht sehr umfangreich waren. Es ist jedoch nicht davon auszugehen, daß das gezeigte System nicht funktionieren kann. Die hier erhobene Kritik stellt viel mehr einen ersten Eindruck seitens des Autors dar.

[3] gemäss der Definitionen aus Kapitel 3.0.2 ist dies ein Integrator

alles andere als intuitiv, was das Fehlen einer guten Dokumentation um so schwerwiegender macht.

Trotz der genannten Kritikpunkte stellt das CelCorp-System eine interessante Lösung dar. Leider ist es nicht frei verfügbar und scheidet somit als Alternative aus.

5.2 WysiWyg Web Wrapper Factory (W4F)

Ein weiteres Projekt, welches sich mit der Extraktion von Daten aus dem Web beschäftigt ist das W4F Projekt [AS], welches von der University of Pennsylvania initiiert wurde. Leider existiert derzeit offenbar keine Implementierung der Anwendung. Das Ziel dieses Projektes ist, halbautomatisch ”Wrapper” zu erstellen, die Daten aus HTML in ein anderes Zugriffsformat überführen. Dazu wurden vier Teilsysteme entwickelt:

- eine Abfragesprache, um die anzufragenden Ressourcen zu identifizieren und aufzufinden (retrieval language)
- die HTML Extraction Language (HEL) zur Deklaration der Extraktionsregeln
- eine Abbildungsdefiniton (mapping layer specification) zur Übersetzung der Eingabedaten in ein Ausgabeformat
- einem Satz von Tools zur Analyse der Systeme der Dienstanbieter

Die eben genannte Abfragesprache ist eine proprietäre Sprache, die im Rahmen dieses Projektes entwickelt wurde. In dieser Sprache werden URLs der zu erfragenden Ressourcen angegeben. Es ist möglich, Parameter in Form von Konstanten oder von Parametern, die vom Basissystem geliefert werden, zu setzen. Es wird explizit darauf hingewiesen, daß eine Steuerung der Applikation, wie sie vom CelCorp WebRecorder implementiert wird, nicht vorgesehen ist. Die Syntax dieser Sprache ist proprietär. Sie gleicht sehr entfernt einer Programmiersprache wie Java oder Perl. Ein Beispiel findet sich auf [AS]

Die Sprache HEL ist ebenfalls eine Eigenentwicklung im Rahmen dieses Projekts. Die Adressierung der Informationen innerhalb der Seite geschieht hierbei durch eine XPath-artige Syntax. Es ist neben der reinen Adressierung durch einen Pfad durch den DOM-Baum möglich, mehrere Funktionen und reguläre Ausdrücke zu benutzen, um die Daten weiter einzugrenzen. Es können mehrere Datensätze, also etwa Daten aus einer Liste, adressiert und extrahieren werden.

Die extrahierten Daten werden in so genannte nested string lists, kurz NLS überführt. Im dritten und letzten Teil der Extraktionsanwendung werden diese über die Abbildungsdefinition in benutzerdefinierte Datenformate überführt. Eine Interpretation, wie etwa die Umwandlung von Zeichenketten in Zahlen, kann vorgenommen werden. Es wird beschrieben, daß dieser

Vorgang auch um individuell erstellte Wandler erweitert werden kann. Letztendlich werden die Informationen über die Abbildungsvorschriften auf Java-Klassen abgebildet.

5.2.1 Kritik am W4F Ansatz

Leider ist es nicht gelungen, nähere Informationen über das Projekt zu erhalten, als in [AS] zu lesen. Aus diesem Grund kann nur eine theoretische Bewertung (und Beschreibung) vorgenommen werden.

Die Architektur der Anwendung ist dem Ansatz, wie in Kapitel 4 empfohlen, wird sehr ähnlich. Theoretisch ist das W4F System offenbar recht mächtig. Leider läßt sich das nicht in der Praxis verifizieren, und das ist – gerade in diesem Feld – ein wichtiges Kriterium. (Einen Einblick in die Probleme, die in der Praxis entstehen können, finden sich in Kapitel 6.3.)

Die Analyseapplikation wird leider nicht umfangreicher vorgestellt. Sie liefert laut den Angaben Pfade zu visuell adressierbaren Teilen der zu untersuchenden Seite. Außerdem wird ein Formular-Analysetool zur Verfügung gestellt, welches bei der Analyse der Abfragesprache helfen soll.

Auf der theoretischen Seite ist vor allem die Verwendung proprietärer Sprachen zu bemängeln. So eine Implementation ist nicht empfehlenswert, da sie mehr Spezialwissen erfordert als notwendig. Eine bekannte, standardisierte Syntax wäre hier zu begrüßen.

Inwieweit andere hier angesprochene technische Hürden, wie etwa durch JavaScript modifizierte Dokumente, bewältigt werden können, kann nicht genau ermittelt werden. Zumindest aber entspricht es der Philosophie, daß eine so genannte multi-granulare Abfrage möglich ist. Diese ist definiert als Abfrage auf Tag-Ebene (sie wird durch den Pfad durch das Dokument umgesetzt) aber auch als Abfrage auf Intra-Tag-Ebene. Eine Abfrage auf Intra-Tag-Ebene ist hier definiert als eine Abfrage, die auch auf Ausschnitte der Texte, welche von den Tags umschlossen werden, zugreifen kann. Umgesetzt wird das durch die bereits erwähnten Funktionen und regulären Ausdrücke. Insofern ist es zumindest in der Theorie möglich, jeden Teil der HTML-Seite zu extrahieren.

Im Gegensatz zu der in Kapitel 4 vorgeschlagenen Methode der Analyse der Kommunikation werden hier die Elemente der Abfrageformulare ausgewertet. Dieses Vorgehen kann mitunter komplizierter sein als die direkte Analyse der Kommunikation, da die Aufarbeitung der Elemente, wie sie ein Browser vornimmt, hier vom User vorgenommen werden muß.

5.3 Zusammenfassung

Keine der hier gezeigten Lösungen sind geeignet, um im gegebenen Umfeld praktisch eingesetzt zu werden. Das ist insbesondere deswegen bedauerlich, da beide Lösungen durchaus viele der genannten Anforderungen erfüllen. Letztendlich aber scheiden beide aus. Im Falle des W4F-Systems fehlt ganz einfach eine Implementierung. Außerdem scheint das System schon alt zu sein und ist somit nicht mehr auf die Belange der modernen Softwareentwicklung zugeschnitten.

Das CelCorp-System ist leider nicht frei verfügbar und soll deswegen nicht eingesetzt werden. Abgesehen davon bietet es aber einen viel versprechenden Ansatz. Dieser allerdings ließ sich in praktischen Tests nicht verifizieren, was die Entscheidung gegen eine Verwendung dieses Systems noch bestärkt. Sollten sich die Voraussetzungen in der Zukunft ändern, so daß auch nicht-freie Software eingesetzt werden kann, so sollte eine eingehendere Prüfung des CelCorp-Systems durchgeführt werden. Es befindet sich bereits in einem Produkt-Status, inklusive eines kompletten GUI und einer Dokumentation. Dadurch ist dieses System am ehesten für die Benutzung durch Nicht-Entwickler geeignet.

6 Realisierung

Um einen möglichst hohen Grad an Flexibilität zu erhalten und um unabhängig von einer bestimmten Firma zu sein, wurde eine eigene Lösung für das vorliegende Problem gefunden. In der vorliegenden Implementation wurde, wie bereits in Kapitel 4 vorgeschlagen, eine Abfrage- und eine Analyseapplikation erstellt. Für beide Applikationen wurden soweit wie möglich bereits existierende Softwarekomponenten benutzt, um den allgemeinen Anforderungen an Modularität, aber auch Stabilität und Zuverlässigkeit gerecht zu werden. Dieses Vorgehen machte die Entwicklung im engen Rahmen, der zur Erstellung dieser Ausarbeitung gegeben ist, erst möglich.

6.1 Die Analyseapplikation

Die Analyseapplikation erfüllt – gemäß den Anforderungen – zwei Aufgaben. Sie unterstützt den User bei der Analyse der Kommunikation zwischen dem Server und dem Browser und sie hilft, die dabei geholten Seiten zu analysieren beziehungsweise deren Struktur zu erkennen.

Der User der Applikation muß, um einen Dienst analysieren zu können, ihn zunächst mit seinem Browser aufrufen und die gewünschten Daten exemplarisch zur Anzeige bringen. Die Kommunikation wird dabei durch einen Proxyserver, welcher vom Browser benutzt wird, überwacht. Der Proxyserver ist Bestandteil des Analyseprogramms und sammelt die Anfragen an den Dienstanbieter. Diese Daten können vom User eingesehen werden.

Wenn der User die gewünschten Daten in seinem Browser geladen und zur Anzeige gebracht hat, muß die Kommunikation untersucht werden. Hierzu werden die Daten, die an den Webserver gesendet worden sind, im Analyseprogramm eingesehen. Die wichtigen Anfragen können rausgesucht und in einer separaten Liste gespeichert werden.

Anschließend gibt das Programm dem User die Möglichkeit, die gesammelten Anfragen an der Server zu schicken. Sollten die richtigen Anfragen rausgesucht worden sein, so wird der Server wieder die vom User ursprünglich angeforderten Daten senden. Diese Daten werden in einem weiteren Browserfenster und außerdem als Baumstruktur im Analyseprogramm dargestellt.

Der User kann nun einzelne Bereiche der Seite zu markieren. Es wird eine Referenznummer für die Daten angezeigt. Im Analyseprogramm kann der User diese Nummer benutzen, um die entsprechenden Knoten in der Baumstrukturdarstellung aufzufinden. Das Programm wird dann zu jedem gewünschten Knoten einen XPath generieren. Dieser kann vom User weiter modifiziert werden. Es wäre beispielsweise denkbar, einzelne Zeilen gegen Zeilenbereiche auszutauschen.

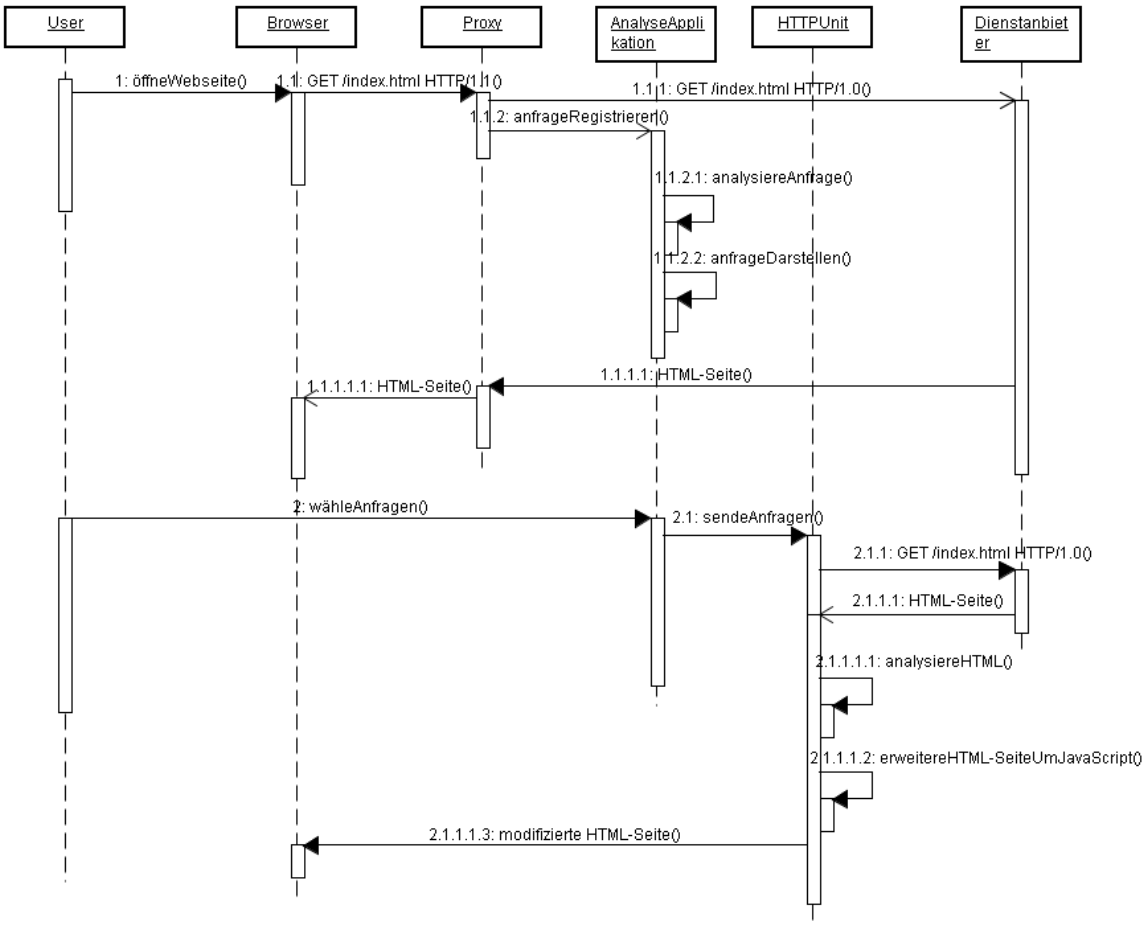

Abbildung 6.1: Sequentielle Darstellung des Analyseprozesses

6.1.1 Interner Aufbau der Analyseapplikation

Die Analyseapplikation vereint drei Komponenten:

- den Browser als Abfrage und Darstellungsmedium
- das HttpUnit-Framework [Gol] als interne Kommunikationsapplikation

- einen SOCKS Proxy-Server [Kou] als Brücke zwischen dem Browser und der Applikation

Das Zusammenspiel dieser Komponenten wird in Abbildung 6.2 dargestellt.

Für das System kann jeder gängige Browser eingesetzt werden, sofern er mit einem SOCKS Proxy Server arbeiten kann. Die Verwendung des Browsers als Abfragemedium bietet eine Reihe von Vorteilen. Trotz vorhandener Standards kann nicht jede Webseite durch jeden Browser korrekt wiedergegeben werden. Es kommt häufig zu Darstellungsfehlern. Die Benutzung eines Standardbrowsers minimiert hier das Risiko, daß eine Seite nicht analysiert werden kann, weil das Wiedergabemedium sie nicht darstellen kann. Da die Abfrageapplikation in Java geschrieben ist, ist sie auf diversen Plattformen lauffähig. Zumindest alle relevanten Plattformen verfügen auch über Browser, somit kann die Analyseapplikation auf einer großen Zahl von unterschiedlichen Systemen eingesetzt werden.

Der eingesetzte Proxyserver ist ein frei verfügbares Programm, welches als Quelltext vorliegt und auch modifiziert werden darf. Die Benutzung eines Proxy-Servers zur Analyse der Kommunikation stellt sich als einfachste Variante der Überwachung der Kommunikation dar.

Ein Proxyserver nimmt generell Anfragen von einem Anfrageprogramm entgegen und reicht sie dann an den ursprünglich gewollten Server weiter. Dieses Vorgehen wird normalerweise eingesetzt, um zum Beispiel das Datenaufkommen zu verringern. In diesem Falle speichert der Proxy die Antworten zu den Anfragen und liefert bei jeder weiteren Anfrage die gespeicherte Kopie aus. Im Falle der Analyseapplikation wird die Proxy-Schnittstelle benutzt, um Zugriff auf die Anfragen und die Antworten zu erhalten. Der Proxy liest dabei die Antworten und identifiziert das eigentliche Ziel der Anfrage. Er baut dann eine Verbindung zu diesem Ziel auf und sendet die Daten vom Browser. Gleichfalls werden die Antworten an den Browser geleitet. In der Analyseapplikation wird der Proxyserver eingesetzt, um die Anfragen des Browsers zu protokollieren. Derzeit ist eine Protokollierung der Antworten zu einer Anfrage nicht vorgesehen.

Nachdem die Anfragen protokolliert wurden, müssen diese – zunächst testweise – ausgeführt werden. Hierzu wird die HttpUnit-Software eingesetzt. Mittels HttpUnit ist es sehr leicht möglich, Zugriff auf die Quelltexte der übertragenen Seiten in dem Zustand wie sie ein Browser anzeigen würde, zu erhalten. JavaScript-Elemente wurden hierbei bereits ausgeführt.

Die übertragenen Seiten werden dann aus ihrer DOM-Struktur heraus in HTML zurückübersetzt. Während dieses Vorgangs werden die jeweils äußersten Elemente, welche normalerweise die Textdaten der Seite sein werden, von -Tags umgeben. Diese können mit JavaScript-Funktionen verbunden werden. Im vorliegenden Fall wird bei einem Klick auf ein solches Element eine JavaScript-Funktion aufgerufen, die dem User eine eindeutigen Identifikation des entsprechenden Elements darstellt.

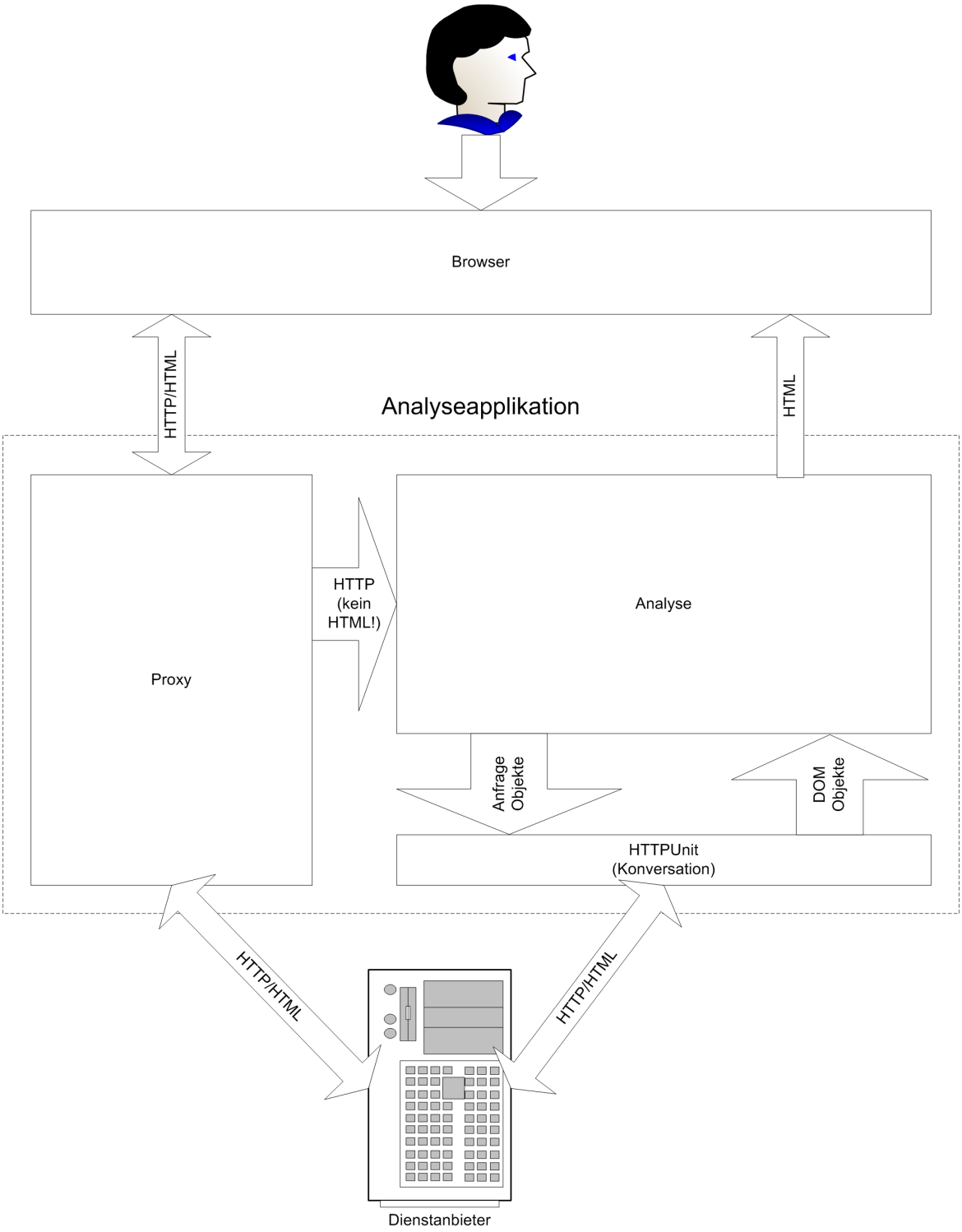

Abbildung 6.2: Zusammenwirken der einzelnen Komponenten der Analyseapplikation

Das Rückübersetzen eines DOM-Baumes in HTML zur Analyse, hat einen großen Vorteil: Es wird immer mit den wirklich vom Browser angezeigten Daten gearbeitet. Diese können sich durch Veränderungen mittels JavaScript von den ursprünglich angezeigten Daten unterscheiden.

Den letzten Schritt in der Analysearbeit stellt das Entwickeln von XPath-Definitionen dar. Das Analysetool unterstützt den User auch hierbei. Es stellt den DOM-Baum der angezeigten Seite dar. Durch Selektieren eines einzelnen Elementes wird ein eindeutiger XPath zu diesem Element generiert. Dieser kann dann vom User verändert werden. Es ist hierbei möglich, einige der von XPath spezifizierten Funktionen zu benutzen. Das Programm kann dann auf Befehl des Integrators den XPath gegen das gegenwärtige Dokument prüfen. Es werden (derzeit noch über die Standardausgabe) die Inhalte der jeweiligen Knoten ausgegeben. Dieser Vorgang soll genutzt werden, um die entsprechenden Pfade zu den gewünschten Daten ausfindig zu machen. Ein gefundener XPath kann dann in der Extraktionsapplikation benutzt werden, um die Daten aus jeder weiteren Seite gleicher Struktur zu extrahieren. Um sicherzustellen, daß der richtige Pfad gefunden wurde, sollte diese gegen mehrere exemplarische Seiten getestet werden.

6.1.2 Die einzelnen Komponenten im Detail

Der Proxyserver

Um den Proxyserver JSocks [Kou] für die Analyseapplikation benutzbar zu machen, wurde dieser so modifiziert, daß die Anfragen neben dem eigentlichen Ziel auch an einen Parser geschickt wurden, der die Anfragen analysiert und in Java-Objekte übersetzt. Dieser Vorgang wird von der Klasse CoyoteBridgeThread übernommen. Je geöffneter TCP Verbindung wird so eine Brücke erstellt. Die Brücke wiederum kopiert die Daten aus dem InputStream vom Browser in einen Kopier-Stream, welcher dem Anfrageparser übergeben wird.

Der Anfrageparser wurde ebenfalls nicht selbst entwickelt. Es wurde stattdessen der Parser der Servlet Engine Tomcat [Fou] eingesetzt. Das Modul, welches innerhalb von Tomcat für die Verarbeitung der Anfragen verantwortlich ist, heißt Coyote. Der Parser wiederum wird mit der Applikation über einen Adapter mit der eigentlichen Applikation verbunden. Dieser Adapter ist vergleichbar mit dem von [EG04] beschriebenen Befehlsobjekt. Der Parser wird solange parsen, bis ein Request vollständig übermittelt wurde, um dann die `service(Request,Response)`-Methode des Adapters aufzurufen. An dieser Stelle wird ein `Request`-Objekt geliefert. Dieses verfügt über diverse Methoden, welche die Attribute und sonstigen Charakteristika der Anfrage wiedergeben können.

Der im konkreten Fall eingesetzte Adapter registriert den Request an einer zentralen Sammelstelle für Requests, indem er ein `ActualRequest`-Objekt instantiiert. Dieses wird im

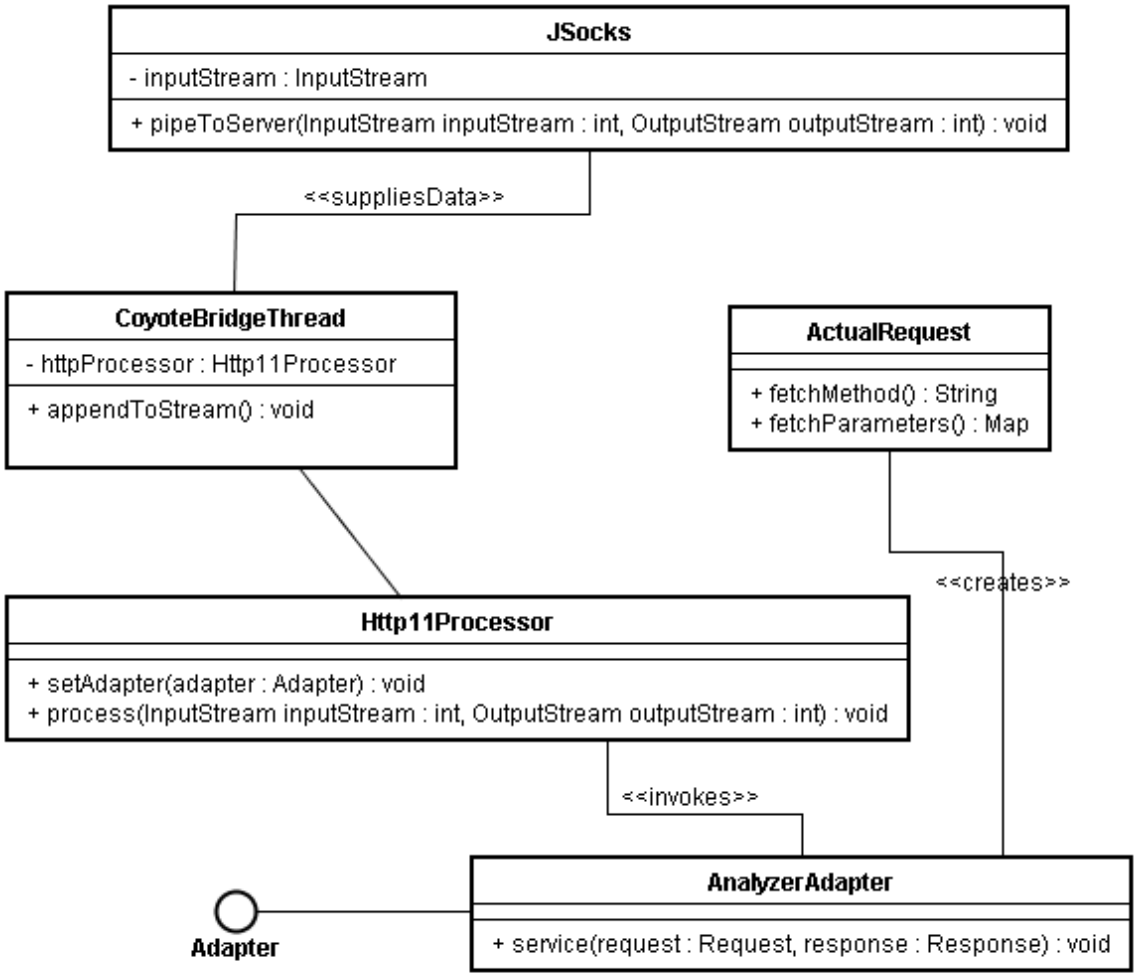

Abbildung 6.3: Zusammenspiel von Coyote und der Analyseapplikation

weiteren Programmverlauf benutzt, um Daten der Anfrage erfragen zu können. Der Adapter wurde als anonyme Klasse innerhalb von `CoyoteBridgeThread` implementiert.

HttpUnit

Um möglichst wenig Aufwand bei der Kodierung der Interaktionsschicht zu verursachen, wurde für diese das HttpUnit-Framework [Gol] eingesetzt. Diese wurde ursprünglich entwickelt, um Webseiten systematisch zu testen. HttpUnit ermöglicht einen einfachen Zugriff auf alle Teile einer Webseite. Außerdem ist ein einfacher JavaScript Interpreter in HttpUnit integriert. Das führt dazu, daß man mittels HttpUnit den Quelltext, der zur Darstellung einer Webseite im Browser führt, erfassen und verarbeiten kann. Dieser ist unter Umständen anders als der übertragene Quelltext, da er beispielsweise durch JavaScript erweiteret werden kann.

HttpUnit bietet dem Entwickler eines darauf aufbauenden Systems eine einfache Schnittstelle zur Kommunikation mit dem Web. Die gesamte Kommunikation wird über Objekte verkapselt. Neben dem reinen Übertragen der Daten kann HttpUnit noch weitere Aufgaben übernehmen: es verwaltet das Senden und Empfangen von Cookies. Somit muß der Entwickler an dieser Stelle keine eigene Arbeit investieren. Auch Weiterleitungen via HTTP werden von HttpUnit übernommen.

Im folgenden wird eine einfache Anfrage mittels HttpUnit dargestellt:

```
...

// first: create a conversation object
WebConversation conversation = new WebConversation();
// exceptions are used to report errors. Since
// we are not interested in the errors of the pages
// we query, we ignore them
conversation.setExceptionsThrownOnErrorStatus(false);
HttpUnitOptions.setExceptionsThrownOnScriptError(false);
// create a GET request here...
WebRequest request =
    new GetMethodWebRequest("http://www.somehost.com");
// set some parameters to tell what we want
request.setParameter("start", "Hamburg");
request.setParameter("dest","Frankfurt");
// get the response by actually executing the request.
WebResponse response = conversation.getResponse(request);
processDOM(response.getDOM());
```

Der hier dargestellte Quelltextauszug verdeutlicht, wie einfach die Arbeit mit HttpUnit ist. Es ist an keiner Stelle notwendig, manuell HTTP-Anfragen zu kodieren und das gesamte Verwalten und Parsen der Antworten wird ebenfalls von HttpUnit übernommen.

Intern kann HttpUnit zwei verschiedene Parser verwenden: NEKOHtml [Claa] oder JTidy [Giu] . Beide sind HTML-Scanner und -Parser, die HTML-Seiten verarbeiten und korrigieren können. Sie überführen die Seiten dabei (nach Möglichkeit) in ein standard-konformes HTML Format. Dieses kann leicht weiterverarbeitet werden. Beide Tools implementieren die DOM-Schnittstelle.

6.2 Die Abfragekomponente

Die derzeit vorliegende Abfragekomponente ist sehr einfach. Ein Überblick über das Zusammenspiel der einzelnen Komponenten wird in Abbildung 6.4 illustriert. Auch sie basiert auf HttpUnit als Abfragekomponente. Die Ablaufscripts sind derzeit als Listen von Objekten implementiert. Es gibt eine Klasse, welche die Requests der Reihe nach abfragt. Das Ergebnis ist dann mittels der DOM Schnittstelle analysierbar. Es können mit XPath Daten extrahiert werden. In Kapitel 3 wurde vorgeschlagen, hierfür eine Sammlung von Werkzeugen anzufertigen. Sie befindet sich im Paket `de.einfachelogik.tools`. Derzeit sind nur zwei Werkzeuge vorhanden. Eines schreibt den Inhalt der einzelnen Nodes, die über einen XPath adressiert werden, in eine Liste (`java.util.List`). Das Zweite gibt den Inhalt eines Nodes als `String` zurück.

Die eigentlichen Extraktor-Klassen befinden sich im Paket `de.einfachelogik.extractors`. Derzeit gibt es einen Extraktor, der Daten aus dem Geofox Informationssystem (siehe [Gmb]) extrahiert. Konkret werden hier die Fahrzeiten für eine Fahrt von einer Start- zu einer Zielhaltestelle extrahiert. Die Klasse `GeofoxQuery` stellt dabei bislang eine Vereinfachung des in Kapitel 4 vorgeschlagenen Schemas dar, da sie in der derzeitigen Version sowohl Extraktion, als auch die semantische Aufarbeitung und Überführung in ein Zielformat (`TimeQueryAnswer`) übernimmt. Diese Teile lassen sich aber leicht trennen, beziehungsweise die Überführung der Strings in ein `java.util.Date` Objekt externalisieren.

Die Klasse `GeofoxQuery` enthält einen kleinen Testtreiber, der die Benutzung dieser Klasse verdeutlicht:

```
// set the location of the previously recorded requests
System.setProperty(GeofoxQuery.REQUEST_FILE_LOCATION_PROPERTY
    ,"geofox.ser");
// create a request object
GeofoxQuery geofoxQuery = new GeofoxQuery();
```

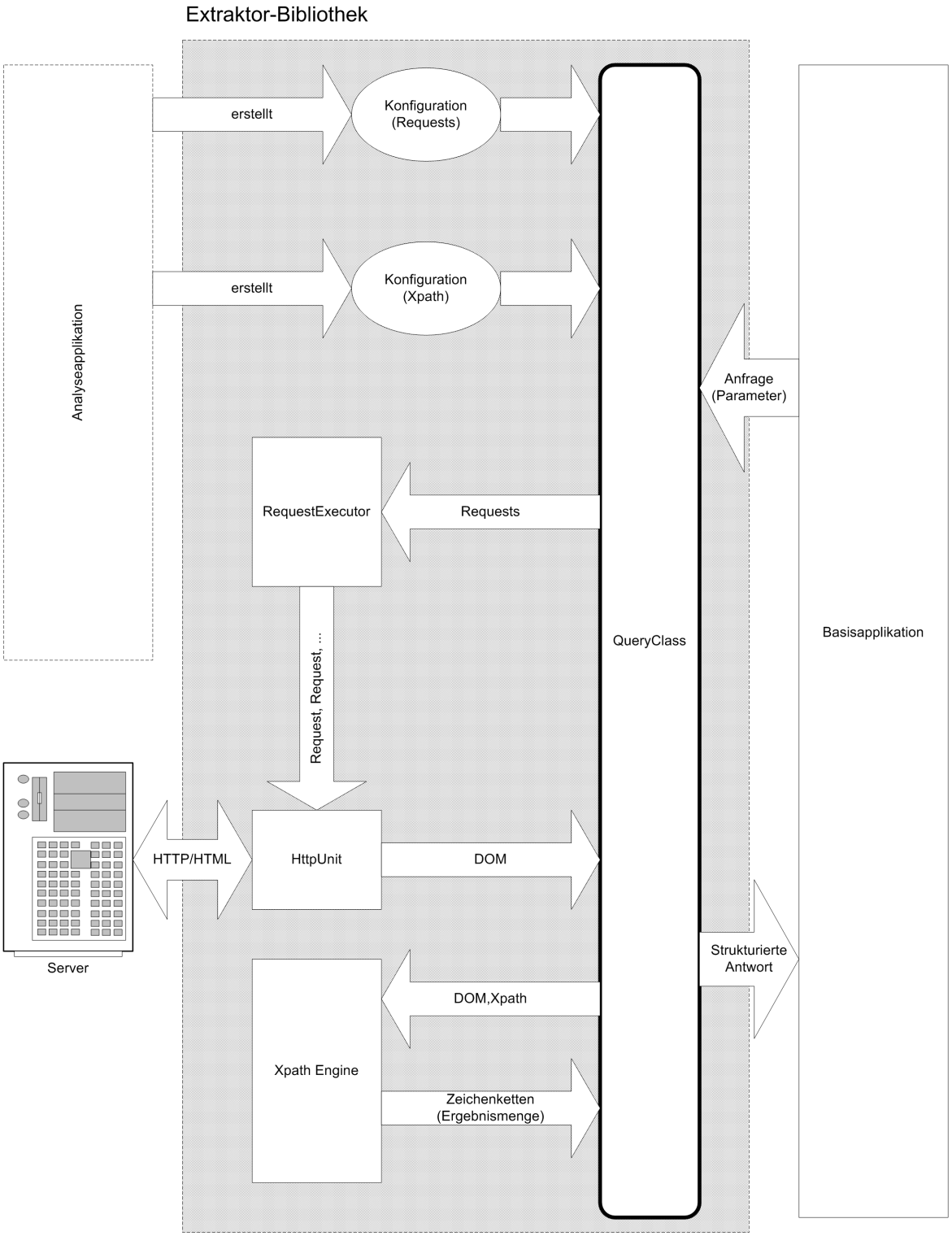

Abbildung 6.4: Zusammenwirken der einzelnen Komponenten der Abfrageapplikation

```
// and query the result
TimeQueryAnswer tqa = geofoxQuery.
        getStartAndArrival(args[0],args[1],new Date());
// check if this request was a success
if (tqa != null) {
    // success: output to user
    System.out.println("start: " + tqa.getStartTime());
    System.out.println("arrival: " + tqa.getArrivalTime());
} else {
    // no success.
    System.out.println("sorry, but no times could be determined.");
}
```

Die Namen der Start- und Zielhaltestelle werden hier in Form von Kommandozeilenparametern an den Testtreiber übergeben. Die Abfrage selbst stellt sich als äußerst einfach dar.

Die Request-Objekte werden derzeit serialisiert auf die Festplatte geschrieben. Ein solcher `IActualRequest` wird direkt vom Analyseprogramm erzeugt. Innerhalb der Analyseapplikation wird dieser mittels eines Wrappers (siehe [EG04]) um die Funktionalität erweitert, Parameter auch im Nachhinein verändern zu können. Er beinhaltet eine Kopie der Daten, die bei der exemplarischen Abfrage der Daten übertragen wurden. Die Parameter, die das Verhalten der Webapplikation verändern sollen, können überschrieben werden. Im Falle der Anfrage an Geofox sind das vier Parameter:

```
// start and destination station name needs to be set.
request.setParameter("startName",startStation);
request.setParameter("destName",destinationStation);

// the dates must be converted to strings.
SimpleDateFormat startDateParser =
    new SimpleDateFormat("dd.MM.yyyy");
// startDate is the day when we want to travel.
request.setParameter(
    "startDate",startDateParser.format(startTime));
SimpleDateFormat startTimeParser =
    new SimpleDateFormat("kk:mm");
// startTime is the time on that day.
request.setParameter(
    "startTime",startTimeParser.format(startTime));
```

Nachdem die Abfrageparameter gesetzt wurden, werden die Requests von dem `RequestExecutor` ausgeführt. Es ist im Kern dieselbe Komponente, die auch in der Abfrageapplikation benutzt wird, um die Testanfragen auszuführen. Auf diesem Weg wird ein Maximum an Stabilität erreicht, und es ist sichergestellt, daß die Abfragen, welche in der Analyseapplikation funktionierten, auch hier funktionieren.

In Kapitel 4 wird vorgeschlagen, eine zweite Schicht zu erstellen, die die Wandlung der Daten in ein strukturiertes Format übernimmt. In der vorliegenden Realisierung ist dies nicht umgesetzt worden. Stattdessen wird die Wandlung direkt vorgenommen. Hierzu wurde ein weiteres Werkzeug entwickelt, welches mittels regulärer Ausdrücke Datum- und Uhrzeitrelevante Daten (also Zahlen sowie die Trennzeichen "." und ":") aus einer Zeichenkette extrahiert. Diese können dann in ein Datumsobjekt umgewandelt werden. Eine umfangreiche Ablaufsteuerung ist derzeit noch nicht implementiert. Sie muß somit komplett manuell kodiert werden. Hierfür stehen dem Entwickler allerdings die Mechanismen des XPath zur Verfügung. Das ermöglicht ein einfaches Zugreifen auf die Inhalte einzelner Teile einer Seite, in denen sich beispielsweise Fehlerinformationen befinden können. Die eigentliche Extraktion der Daten wird im folgenden dargestellt:

```
// extract the times with the xpath
List times = XPathToList.xpathToList("/HTML/BODY[1]/DIV[1]/
TABLE[1]/TR[3]/TD[1]/DIV[1]/TABLE[1]/TR[3]/TD[4]/TABLE[1]/
TR[1]/TD[1]/SPAN[1]/child::text()[1]|" +
"/HTML/BODY[1]/DIV[1]/TABLE[1]/TR[3]/TD[1]/DIV[1]/TABLE[1]/
TR[position()=last()-3]/TD[4]/TABLE[1]/TR[last()]/TD[1]/SPAN[1]
/child::text()[1]", document);
// no times? --> failure.
if (times.size() == 0 ) return null;
// extract the date (not really required, but nice to have)
String dateString = XPathToString.xpathToString("/HTML/BODY[1]
/DIV[1]/TABLE[1]/TR[3]/TD[1]/DIV[1]/TABLE[1]/TR[1]/TD[2]/
SPAN[1]", document);
```

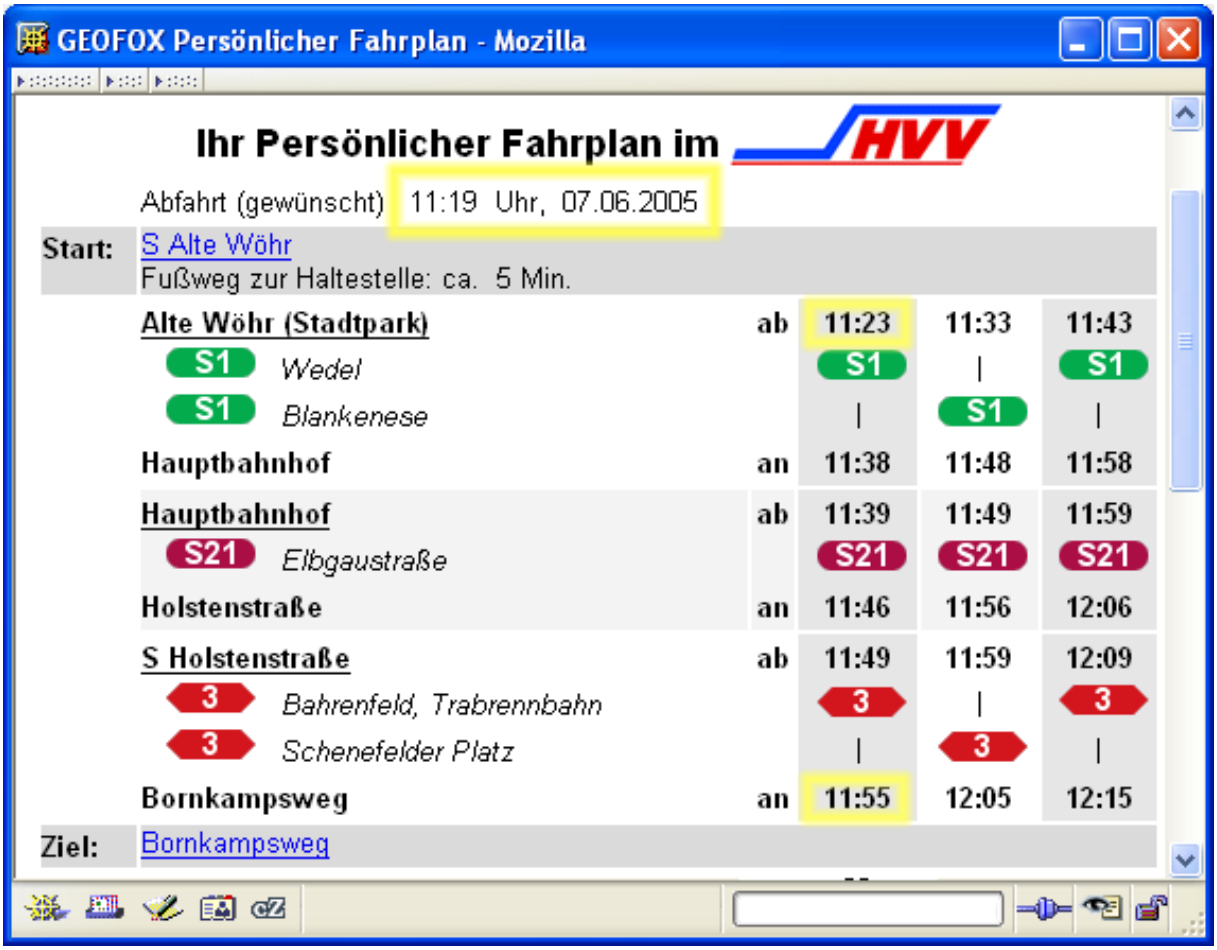

Abbildung 6.5: vom Extraktor ausgewählte Xpath-Regionen in der dargestellten Webseite

In dieser Applikation werden zwei verschiedene XPaths benutzt. Diese sind in Abbildung 6.5 farbig markiert. Die Ausgabe des zweiten, im Quelltext zu sehenden XPath, ist das Datum, an dem der Transport stattfinden wird. Es wäre nicht notwendig es zu extrahieren, da es durch die Eingabeparameter ohnehin festgelegt ist, wurde aber trotzdem zu Demonstrationszwecken implementiert. Dieser XPath ist vergleichsweise einfach, da er mit absoluten Indizes auskommt. Die Struktur, die beschrieben wird, lautet etwa: ”HTML-Element, davon erstes Unterelement vom Typ `body`, davon erstes Unterelement vom Typ `div`, davon erstes Unterelement vom Typ `table`, davon drittes Element vom Typ `tr` (dritte Zeile), davon erstes Element von Typ td (erste spalte)” und so weiter. Das Ergebnis sieht in etwa (um einige unsichtbare Zeichen wie Leerzeichen oder Tabulatorzeichen bereinigt) so aus:

```
Abfahrt

(gewünscht)
  11:19
Uhr,
07.06.2005
```

Zur Aufarbeitung dieses Datensatzes wurde das oben genannte Werkzeug zur Extraktion von Datum- und Zeitangaben benutzt.

Der im Quelltext erste genannte XPath ist etwas aufwendiger. Neben einer tieferen Schachtelung der Daten, wurden hier Funktionen der XPath Schnittstelle benutzt, um Zugriff auf mehrere Elemente zu erhalten. Folgende Daten wurden extrahiert:

```
11:23
11:55
```

Zum einen wurde hier die Funktion `position()` benutzt. Sie kann benutzt werden, um den Index eines Subelements in seinem übergeordneten Element zu ermitteln. Dies ist notwendig, da – je nachdem, wie oft umgestiegen werden muß – die Zahl der Tabellenzeilen variiert. Es wird mittels `position()=last()-3` nur auf das jeweils drittletzte Element des Pfades an diesem Knoten zugegriffen.

Zum anderen wurden hier mehrere Pfadsegmente mittels des "|" Operators zu einem Pfad zusammengefaßt. Das ist notwendig, da die Zeiten für jedes Teilsegment jeweils in einer Zeile dargestellt werden. Abhängig von der gewünschten Zeit, also der Abfahrts- oder Ankunftszeit, müsste der XPath ansonsten auswählen, welche Zeile in der Tabelle ausgewählt werden muß.

6.3 Evaluation

Um die Realisierung besser bewerten zu können, wurden verschiedene Tests durchgeführt. Zum einen wurden verschiedene Dienstanbieter untersucht und zum andern wurde ein weiteres Testsystem entworfen, welches auf der hier vorgestellten Realisierung aufbaut.

Bereits im vergangenen Kapitel wurde ein Testtreiber für das Informationssystem des Hamburger Verkehrsverbundes vorgestellt. Er kann insofern erfolgreich eingesetzt werden, als Daten extrahiert werden konnten. Insgesamt stellte sich dieses System als äußerst 'gutmütig' dar. Im Gegensatz zu anderen Anbietern gibt es hier keine Mechanismen, die den Zugriff

auf die Seiten erschweren. Es werden auch keine Sitzungsdaten benötigt. Daher ist die Extraktion von Daten aus diesem System sehr einfach.

Als problematisch haben sich solche Daten herausgestellt, die (serverseitig) nicht eindeutig identifiziert werden konnten. Eine Anfrage für eine Station, die nicht existiert, brachte (erwartungsgemäß) kein Ergebnis. Doch auch solche Anfragen, die korrekt waren, wurden teilweise nicht akzeptiert, da der genaue Stationsname auf dem Server nicht mit dem vom User eingegebenen übereinstimmte. Ein Beispiel hierfür ist der Name der Station ”Alte Wöhr”. Er wird vom System nur dann erkannt, wenn der exakte Name ”S Alte Wöhr” angegeben wird. An dieser Stelle muß im produktiven System nachgebessert werden. Das Informationssystem nämlich liefert durchaus Ergebnisse, nur sind diese nicht vom gewünschten Typ. In dem für den Gebrauch durch den Internetbenutzer geschaffenen System werden die Eingabeseite wiederholt ausgegeben und mögliche Treffer vorgeschlagen.

In einem weiteren Test wurde die Extraktion von Daten aus einer Suche nach Flugangeboten bei www.opodo.de untersucht. Zu diesem Test existiert kein Testtreiber. Es war dennoch möglich, Daten – über die Testfunktion innerhalb des Analyseprogramms – zu extrahieren. Im Vergleich zu dem von Celcorp entwickelten WebRecorder ist das hier entwickelte System in der Lage, von diesem Anbieter Daten zu beziehen. Die von Opodo gelieferten Seiten werden erst nachdem der Browser sie empfangen hat, via JavaScript in ein darstellbares Format überführt. In ihrem Urzustand sind alle Informationen nur als JavaScript Code vorhanden. Dieser ist für ein System das eine Tag-artige Struktur interpretieren muß, nur schlecht verarbeitbar. Das wäre zwar theoretisch auch mit dem Celware System als auch mit der hier vorgeschlagenen Lösung möglich, würde aber einen höheren Aufwand an Kodierung bedeuten. Außerdem ist es bei einer solchen Lösung weit schwieriger, eine stabile Konfiguration zur Extraktion zu erstellen. Die hier vorgeschlagene Lösung ist fähig, JavaScript auszuführen. Dies ermöglicht es auch, aus einer Seite, wie sie Opodo liefert, Daten über einen XPath zu extrahieren.

Die Grenzen des Systems zeigte der Dienst von www.bahn.de auf: hier war es nicht möglich, Daten zu extrahieren. Jede der Testabfragen führten – obgleich dieselben Parameter wie sie der Browser gesendet hatte, gesendet wurden – lediglich dazu, daß der Server die Abfrageseite wieder ausgab. Der Test wurde wiederholt und die bytegenaue Anfrage des Browsers wurde gesendet. Diese Abfrage führte zum Erfolg. Auch kleinere Modifikationen innerhalb dieser Anfrage, wie etwa das Verändern der Reihenfolge, in welcher die Parameter gesendet wurden, führten zur erneuten Ausgabe der ursprünglichen Antwortseite. Lediglich die von HttpUnit generierten Anfragen verliefen erfolglos. Eine genauere Untersuchung ergab, daß die Webapplikation von www.bahn.de die gesendeten Parameter in einer ganz bestimmten Reihenfolge erwartet. Wurde diese teilweise verändert, so funktionierte die Applikation nicht mehr.

Um dieses Problem in den Griff zu bekommen, wurde eine provisorische Modifikation an

HttpUnit vorgenommen, die die Reihenfolge der Parameter in der Anfrage in derselben Reihenfolge sendet, wie sie zuvor gesetzt werden. Leider ist das Festlegen der Reihenfolge bislang noch manuell notwendig: die vom Coyote Parser zur Verfügung gestellten Parameter werden anscheinend nicht in der Reihenfolge, wie sie empfangen wurden an, die auf Coyote aufbauenden Applikationsteile geliefert. Insofern besteht hier noch Bedarf einer Weiterentwicklung von Coyote.

Abgesehen von diesen Veränderungen stellte sich die Benutzung von HttpUnit als äußerst stabil dar. Das Parsen der HTML-Seiten funktionierte stets zuverlässig. Sogar absichtlich mit ungültigem HTML-Quelltext versehene Seiten wurden von HttpUnit verarbeitet. Während der gesamten Arbeit wurde der Neko Parser verwendet. Nach den in Kapitel A.1 beschriebenen Veränderungen war auch die Verarbeitung der JavaScript-Funktionen in den genannten Testfällen erfolgreich.

Gerade während der Arbeit an der Evaluation der www.bahn.de-Seiten stellte sich ein weiterer Vorteil heraus. HttpUnit ist fähig, das Verhalten eines Browsers zu imitieren. Ein Teil der Tests hat sich diese Funktionalität zu Nutze gemacht, in dem das Abfrageformular, welches eine Verbindungsanfrage darstellt, via HttpUnit versendet wurde. Sie verlief erfolgreich, was die Überlegung nahelegt, in einer Produktionsversion eventuell weitere Teile der Funktionalität von HttpUnit benutzbar zu machen. Dies ist allerdings derzeit nicht mit dem Analyseschema vereinbar und erfordert somit einen erhöhten Aufwand und auch mehr Wissen.

www.bahn.de ist auch ein gutes Beispiel für eine Webseite, welche darauf angewiesen ist, daß Parameter aus der übertragenen Seite in die nachfolgende Anfrage übernommen werden müssen. Auch dieses ist bislang nicht generisch umgesetzt. Um dennoch die Möglichkeit zu haben, Parameter zu extrahieren, wurde die Abfrage-Komponente modifiziert, so daß sie die betreffenden Parameter aus der vorhergehenden Seite übernommen hat.

Die Arbeit mit dem Analysetool hat gezeigt, daß die Entwicklung von XPath-Pfaden durchaus zeitintensiv sein kann. Dies liegt vor allem daran, daß bei einem nicht korrekten Pfad entweder ein Interpreter-Fehler gemeldet wird oder aber eine leere Ergebnismenge, was sehr störend ist, da es keinen Hinweis darauf gibt, wo der Fehler im XPath liegt.

6.3.1 Übertragbarkeit der Realisierung

Um die Übertragbarkeit demonstrieren zu können und um ein weiteres Testsystem zu schaffen, wurde eine kleine Applikation erstellt, welche ein Wörterbuch abfragt. Der Dienst, der hier abgefragt wird, heißt dict.leo.org. Dieses Wörterbuch bietet eine einfache HTTP-Schnittstelle. Ein entsprechender Adapter wurde innerhalb weniger Stunden erstellt und funktioniert sehr stabil. Auch in diesem Adapter wurde allerdings auf die in Kapitel 4 vorgeschlagene Zwischenschicht für die semantische Aufarbeitung verzichtet. Dies geschah zum

einen, da eine semantische Aufarbeitung bei einer Wörterbuch-Abfrage keinen Sinn macht, und zum anderen, um die Lösung schlank zu halten. Die Klasse LeoQuery findet sich auf der CD im Quellenverzeichnis.

Die Analyse der Webseiten von Leo zeigte einen weiteren Mangel an der derzeitigen Implementation der Analyseapplikation auf: Die in Kapitel 6.1.1 erwähnten eindeutigen Identifikatoren, welche als Brücke zwischen der visuellen Darstellung und der Darstellung in der Analyseapplikation fungieren, stimmten nicht überein. Dieses wurde offenbar durch die Werbeeinblendungen in der HTML-Seite verursacht. In diesem Fall war es notwendig, den DOM-Baum manuell nach den gewünschten Daten zu durchsuchen. Dieses Verhalten zeigt, daß es in einer Produktivversion möglich sein muß, auch diese Daten zu analysieren, die durch die exemplarischen Abfragen geholt wurden. Bislang ist es lediglich möglich, Daten zu analysieren, die aus der Antwort zu einer der exemplarischen Anfrage nachempfundenen Anfrage geholt wurden.

6.4 Zusammenfassung

Der hier vorgestellte Lösungsansatz ist nicht komplett. Er verdeutlicht aber, daß mit den benutzen Werkzeugen eine Umsetzung möglich ist. Sollte eine Überführung in einen Produktstatus angestrebt werden, so sind noch einige der nicht erfüllten Anforderungen durch geeignete generische Systeme zu ersetzen. Für den Gebrauch durch einen Entwickler ist das entwickelte System – wie sich nicht zuletzt durch den in Kapitel 6.3.1 beschriebenen Einsatz ableiten läßt – bereits brauchbar.

In der Analyse wurde eine Liste von Anforderungen erstellt. Diese wird hier wiederholt und jede der Anforderungen noch einmal gegen die Implementation geprüft.

- Die Schnittstelle muß Anbindung an beliebige Basissysteme ermöglichen

Da die Schnittstelle von Fall zu Fall zu definieren ist, kann diese Anforderung als erfüllt betrachtet werden.

- Die Integration gleicher Dienstleister soll ohne Modifikation des Basissystems möglich sein

Durch das Wählen einer geeigneten Schnittstelle kann diese Anforderung erfüllt werden.

- Die Schnittstelle muß syntaktisch und semantisch eindeutige Daten liefern

Derzeit ist der Teil des Systems, der die Daten extrahiert und verarbeitet, noch so wenig generalisiert, daß es möglich ist, diese Bedingung zu erfüllen. Es gibt aber – seitens des Systems – nur vage Vorgaben, wie dies zu erreichen ist.

- Änderungen am Seitenlayout sollen nur zu geringem Änderungsaufwand führen

Da die Daten über einen XPath adressiert werden, wird dies in vielen Fällen zutreffen. Inwieweit die Verarbeitung und Transformation der Daten davon betroffen ist, hängt vom Einzelfall ab. Es ist aber wahrscheinlich, daß eine Änderung des XPath (bei geschickter Implementation des Transformators) ausreicht.

- Die Lokalisierung der Daten sollte anhand der Struktur der Seiten erfolgen

Auch diese Bedingung wird durch die Verwendung der XPath API erreicht.

- Für eine Ablaufsteuerung muß der Zugriff auf Daten aus Formularen möglich sein

Die Ablaufsteuerung ist derzeit nicht in der Lage diese Anforderung zu erfüllen. Es muß – falls diese Anforderung in einem Projekt benötigt wird – eine Erweiterung zur Ablaufsteuerung entwickelt werden.

- Eine Integration von neuen Diensten auch durch Nicht-Entwickler soll möglich sein

Diese Anforderung wurde nicht erfüllt, da Extraktoren und Transformatoren derzeit noch manuell zu kodieren und zu kompilieren sind.

- Die Erkennung der Struktur einer HTML-Seite durch den Integrator muß unterstützt werden im Hinblick auf die Erstellung einer Konfiguration für die Extraktionsapplikation

Diese Anforderung konnte erfüllt werden.

- Die Analyse der Kommunikation muß ebenfalls dahingehend unterstützt werden, daß die zu übertragenden Parameter und Werte leicht festgestellt werden können

Auch diese Anforderung konnte erfüllt werden.

7 Ausblick und Fazit

7.1 Zusammenfassung

Ziel dieser Arbeit war es, zu ermitteln, ob es möglich ist, Dienste mit Schnittstelle zum WWW als Bezugsquelle für Daten aus einer bestehenden Anwendung heraus zu nutzen. Gleichfalls sollte die Möglichkeit bestehen, die dabei gefundene Lösung auch in anderen Systemen nutzen zu können.

Das Ergebnis dieser Untersuchung ist, daß es möglich ist, ein solches System zu erstellen. Hierzu wurde ein prototypisches Extraktionssystem entwickelt. Es kann Daten auf Verlangen von einem HTML-basierten Dienst beschaffen und an ein darauf aufbauendes System weitergeben. Das Ziel, eine Anbindung anderer Dienste derselben Art ohne Programmieraufwand zu integrieren, konnte nicht erreicht werden. Es wurden jedoch Ansätze vorgestellt, mit denen sich auch diese Anforderung zumindest teilweise umsetzen läßt. Um die Integrationsfähigkeit in andere Umfelder zu demonstrieren, wurden zwei völlig verschiedenartige Dienste an zwei Demo-Applikationen angeschlossen. Die Integration in den Mitfahrclub wurde jedoch bislang nicht umgesetzt.

Um zu dem genannten Ergebnis zu kommen, wurden in der Analyse eine Reihe von technischen Herausforderungen vorgestellt. Es wurden Wege aufgezeigt, wie die Probleme, die überwiegend dadurch entstehen, daß das Medium WWW von seiner Intention nicht für die Interaktion mit anderen Computersystemen vorgesehen war, bewertet und zu isoliert werden können. Es wurde außerdem der Vorschlag gemacht, eine Software, die Analyseapplikation, zu erstellen, welche bei der Integration eines HTML-basierten Dienstes Unterstützung bei der Konfiguration und Erstellung einer geeigneten Komponente assistiert.

Auf Basis der Analyse wurde ein allgemeines Design erarbeitet, das den Ansprüchen genügt. Gegen dieses Design wurden zwei vorhandene Lösungsansätze geprüft. Beide dieser Ansätze waren aus verschiedenen Gründen nicht einsetzbar.

Als Alternative wurde auf der in Design und Analyse geschaffenen konzeptionellen Basis ein prototypisches System erstellt. Um dieses möglichst schnell zu erstellen und aus der Überlegung heraus, daß Standardkomponenten eine höhere Stabilität in das System bringen,

wurden eine Reihe von bestehenden, standarisierten Technologien ausgewählt und zu zwei Anwendungen zusammengefaßt.

Zum einen wurde eine einfache Analyseapplikation implementiert. Sie kann die meistens nicht bekannte Struktur der HTML-Seiten des HTML-basierten Dienstes analysieren. Außerdem kann die Abfrage-Schnittstelle zu einem Dienst analysiert werden, was den Integrationsprozeß noch beschleunigt.

Zum anderen wurden zwei exemplarische Adapter zur Integration von verschiedenen HTML-basierten Diensten in die genannten Demo-Applikationen erstellt.

Die zuvor beschriebene Analyseapplikation erwies sich bei der Erstellung der Adapter als äußerst hilfreich. Es ist außerdem zum jetzigen Zeitpunkt [1] möglich, Daten von den beiden testweise angeschlossenen HTML-basierten Diensten zu bekommen und weiterzuverarbeiten.

7.2 Fazit

Mit dieser Arbeit wurde bewiesen, daß sich zumindest eine Teilmenge der Anforderungen umsetzen läßt. Es ist möglich, HTML-basierte Dienste in andere Computersysteme zu integrieren. Dies ist inbesondere auch ohne Mithilfe der entsprechenden Dienstanbieter möglich. Es wurde gezeigt, daß durch den Einsatz von Standardtechnologien eine stabile Lösung geschaffen werden kann.

Die Benutzung des für Menschen geschaffenen Mediums World Wide Web kann durch den Computer erfolgen. Hierbei muß jedoch immer noch eine Menge manueller Arbeit erledigt werden. Insbesondere die Interpretation der gelieferten Daten bereitet Probleme, da hier keinerlei Formalitäten gegeben sind.

Es hat sich im Zuge der Erstellung dieser Arbeit gezeigt, daß es – selbst unter Zuhilfenahme von ausgereiften Produkten wie etwa dem CelCorp-System – keine leichte Aufgabe ist, die Konfiguration für den Anschluß eines HTML-basierten Dienstes zu erstellen. Die eingangs beschriebene Anforderung, daß möglichst viele Benutzer die Integration eines neuen Dienstes vornehmen können sollen, ist somit nicht umsetzbar. Es wird immer ein hohes Maß an Spezialwissen erforderlich sein, um eine solche Integration vorzunehmen.

Insbesondere die Erfahrung mit dem Geofox-Informationssystem haben gezeigt, daß die Beschaffung von Daten über Fahrten von öffentlichen Verkehrsmitteln durchaus möglich ist.

[1] Stand: Juni 2005. Der Zusammenhang zwischen Zeit und Funktion der Systeme wurde in Kapitel 3 genauer erläutert

Somit ist nun der Weg geebnet, um eine Evaluation der Integration in das Mitfahrclub-System durchzuführen.

7.3 Kritischer Rückblick

Mit dieser Arbeit und durch die dazu erstellte Software konnten nicht alle Ziele erreicht werden. Insbesondere die Verwendbarkeit eines Adapters durch Nicht-Programmierer konnte bislang nicht erreicht werden. Auch sind derzeit noch Änderungen am Quellcode notwendig damit, sich das Design der HTML-Seiten der jeweiligen Dienstanbieter ändert. Daraus folgt, daß noch eine Menge Entwicklungsarbeit geleistet werden muß, bis das hier vorgestellte System in einen Produktstatus überführt werden kann. Abhängig von der Art – beziehungsweise der Wichtigkeit und der Frequenz der Benutzung – innerhalb eines Basissystems, lassen sich die geschaffenen Softwarekomponenten jedoch bereits produktiv einsetzen. Sollte es zu einem Einsatz kommen, ist zu prüfen, wie gewichtig die Anforderung der raschen Anpaßbarkeit durch Nicht-Programmierer ist. Werden viele Anpassungen erwartet, so sollte ein System wie das der Firma CelCorp noch einmal genauer daraufhin untersucht und überprüft werden, ob (sowohl technische als auch politische) Hindernisse nicht ausgeräumt werden können, da dieses System bereits sehr viel Unterstützung bei der Erstellung eines entsprechenden Extraktionssystem leistet. Somit wird hier das Spezialwissen auf ein Mindestmaß reduziert.

Die Möglichkeit, das System produktiv einzusetzen, gilt insbesondere für die Analyseapplikation. Diese hat sich – zwar durchaus noch nicht als völlig fehlerfrei – aber dennnoch als äußerst funktional erwiesen. Während der Benutzung dieser Anwendung im Zuge der Erstellung der Prototypen, war es immer wieder notwendig, zusätzliche Hilfsmittel, wie etwa Paket-Analyse Tools auf TCP-Ebene einzusetzen. Auch das Lesen von (HTML-) Quellcode konnte nicht immer komplett vermieden werden.

Die Auswahl der unterliegenden Komponenten erwies sich als gelungen. Das HttpUnit Framework stellt einen leicht einzusetzenden und zu integrierenden Baustein dar. Leider waren hier jedoch einige Veränderungen notwendig. Es bleibt zu hoffen, daß in späteren Versionen die Änderungen bereits vom Entwicklerteam von HttpUnit vorgenommen werden.

Gerade während der Untersuchung des Dienstes [AG] fiel auf, daß die Verwendung von derart komplexen, weit entwickelten Systemen wie HttpUnit durchaus auch Probleme bereiten können. In diesem Fall war es notwendig, eine relativ einfache Anforderung umzusetzen: das Senden der HTTP-Parameter mußte in einer bestimmten Reihenfolge geschehen. Das war von HttpUnit nicht vorgesehen. Diese Funktionalität hinzuzufügen – die bei einer Eigenentwicklung der Kommunikationskomponente auf entsprechend niedriger Ebene innerhalb von Minuten umzusetzen wäre – dauerte durch die unbekannte interne Struktur von HttpUnit

mehrere Stunden. Die entsprechende Anpassung der Coyote-Komponente, die es ermöglicht hätte, die Reihenfolge zu protokollieren, wurde gar nicht erst vorgenommen. [2] Dennoch konnten die Probleme letztendlich gelöst werden. Alles in allem hätte eine Eigenentwicklung keinen Sinn gemacht.

7.4 Ausblick

Das erstellte System hat noch keinen Produktstatus. Folgende Modifikationen und Erweiterungen sind notwendig, um das System in diesen Zustand überführen können:

- Das User-Interface der Analyseapplikation wurde ohne Rücksicht auf Anwender entwickelt, so daß es für entsprechende Tests ausreichend war. Es muß überarbeitet und die bestehenden Implementationsfehler müssen korrigiert werden
- Die Konvertierung und Extraktion der Daten sollte durch ein generisches System ausgeführt werden. Ein möglicher Ansatz hierfür ist die Benutzung von XSLT
- Der Zusammenhang von Analyseapplikation und Browser sollte komfortabler gestaltet werden. Der Prozeß der Identifikation entsprechender Daten über einen Index ist noch nicht benutzerfreundlich genug.
- Die Ablaufsteuerung ist derzeit nur ansatzweise implementiert. Hierzu sollte ebenfalls ein konfigurierbares System erstellt werden.
- Das Dateiformat der Konfiguration für eine Extraktion sollte in ein universelleres, stabilieres und menschenlesbares Format wie etwa XML überführt werden.

[2] Die Reihenfolge der Parameter wurde in diesem Fall mittels eines Paket-Analyse-Tools vorgenommen und dann in einer spezialisierten Abfrageklasse BahnDeRequestExecutor hart kodiert.

A Anhang

A.1 Veränderungen an HttpUnit

Die Veränderten HttpUnit Klassen finden sich im Source-Baum auf der beiligenden CD. Im einzelnen wurden folgende Änderungen durchgeführt:

- Klasse: `com.meterware.httpunit.javascript.JavaScript` Methode `isSupportedScript( QName element, XMLAttributes attrs )` wurde um eine Null-Abfrage erweitert. Dadurch konnte die Stabilität bei der Ausführung von JavaScript erhöht werden.
- Klasse: `com.meterware.httpunit.parsing.ScriptFilter`. Methode `private void handleScriptException( Exception e,`

  `String badScript ) {` wurde dahingehend erweitert, dass auch bei einem Fehlerhaften Script weitergearbeitet wird. Das war notwendig, um Seiten die fehlerhaftes JavaScript beinhalten, verarbeiten zu können.
- Klasse: `com.meterware.httpunit.UncheckedParameterHolder`. Die Instanzvariable `_parameters` wurde durch eine Instanz der Klasse `MyHashtable` (als innere Klasse implementiert) ersetzt. So ist es möglich, Parameter in einer expliziten Reihenfolge zu senden. Das war für die Abfrage von [AG] notwendig.

A.2 Bildschirmausdrucke der Analyseapplikation

Nachfolgend einige Bildschirmausdrucke der Analyseapplikationen. Es wurden einige Erläuterungen hinzugefügt, die in dem gelben Kästen dargestellt sind.

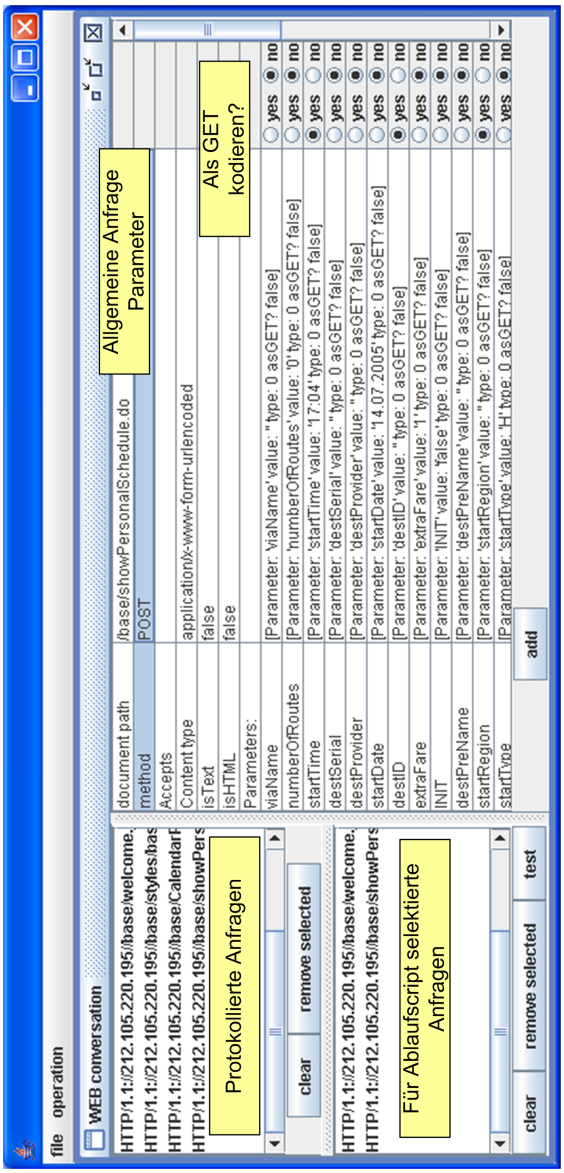

Abbildung A.1: Analyseapplikation protokolliert exemplarische Anfragen

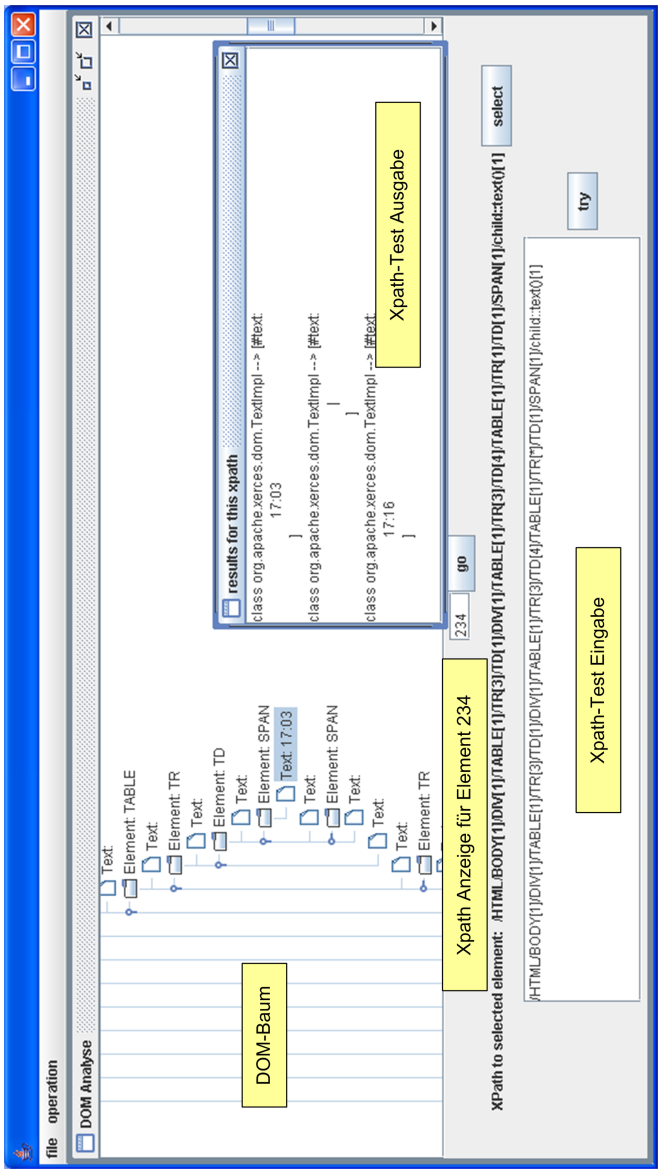

Abbildung A.2: Analyseapplikation stellt XPath-Analyse zur Verfügung

A.3 Bildschirmausdrucke der Wörterbuchapplikation

Nachfolgend einige Bildschirmausdrucke der Wörterbuchapplikation. Der ’Suchen’ Knopf startet die Suche nach Übersetzungen, der ’Speichern’ Knopf speichert in der Tabelle markierte Zeilen in eine Datei (’vocables.list’) im Verzeichnis des Benutzers.

Abbildung A.3: Wörterbuchapplikation sucht nach ”Urlaub”

A.4 Inhalt der CD

Auf der beigelegten CD befinden folgende Daten

- der Quellcode zur Analyseapplikation und den beiden beschriebenen Testtreibern
- eine ausführbare Datei zum Start der Analyseapplikation
- eine ausführbare Datei zum Start der Wörterbuchapplikation
- dieses Dokument im PDF
- die eingesetzten Bibliotheken und Tools als gepacktes Archiv und in der gegebenenfalls angepassten Form
- abgespeicherte Versionen der im Literaturverzeichnis referenzierten Web-Seiten

Glossar

CORBA CORBA is the acronym for Common Object Request Broker Architecture. CORBA is OMG's (siehe auch [OMGb]) open, vendor-independent specification for an architecture and infrastructure that computer applications use to work together over networks.

[OMGa]

DNS Domain Name System. System zur Auflösung von Namen in IP-Adressen im Internet

DTD Data Type Definition. Beschreibt die erlaubten Datenstrukturen in einem in einer Auszeichnungssprache, wie etwa HTML oder XML, verfaßten Dokument

GUI Graphical User Interface

HTML Hypertext Markup Language. Dokumenten-Beschreibungs-Sprache im WWW. Siehe auch Kapitel 2.1 und Kapitel 3.5

HTTP HyperText Transfer Protocol. Kommunikationsprotokoll im WWW

IDL Interface Definiton Language. Sprache zur Definition von Schnittstellen in CORBA

SGML Standard General Markup Language. SGML ist eine durch die ISO-Norm ISO 8879 definierte Metaprache mit der Auszeichnungssprachen (markup languages) wie etwa HTML beschreiben kann. [Gol91] beinhaltet eine genauere Beschreibung.

TCP Transmission Control Protocol. Protokoll für den verbindungsorientierten Dienst im Internet. Stellt unter anderem zuverlässigen Transport, Flusskontrolle und Überlastkontrolle zur Verfügung. Näheres in [Div81]

URL Unified Resource Locator

W3C World Wide Web consortium. Konsortium zur Entwicklung von Standards im WWW [conb]

WSDL Web Service Definition Language. Eine vom W3C spezifizierte Sprache zur Beschreibung von WebServices. Siehe auch [CCMW].

WysiWyg What you see is What you get. Steht für das allgemeine Prinzip ”Was du siehst ist was du kriegst”. Im allgemeinen werden so Editoren für Dokumente genannt, bei denen direkt in einer dem endgültigen Layout gleichenden Darstellungsform gearbeitet wird.

XML eXtensible Markup Language. Neuere, flexiblere und universellere Sprache zum allgemeinen Datenaustausch.

XPath XML Path Language. Sprache zur Adressierung von Teilen einer HTML-Seite. Siehe auch [CD] und Kapitel 4.2.3

XSLT eXtensible Stylesheet Language Transformations. Eine Sprache zur Transformation von XML-Dokumenten in andere XML-Dokumente. Siehe auch [Clab]

ÖPNV Öffentlicher Personennahverkehr

Literaturverzeichnis

[AG] AG, Deutsche B. *Die Bahn - Startseite Reiseportal*. http://www.bahn.de/, verifiziert am 08.07.2005

[Aho85] ALFRED V. AHO, Jeffrey D. U.: *Compilers - Principles, Techniques and Tools*. Addison-Wesley, 1985. – ISBN 0-2011-0194-7

[Akt] AKTIENGESELLSCHAFT, Deutsche L. *Lufthansa*. http://www.lufthansa.de/, verifiziert am 08.07.2005

[AS] ARNAUD SAHUGUET, Fabien A. *WysiWyg Web Wrapper Factory (W4F)*. http://db.cis.upenn.edu/DL/WWW8/, verifiziert am 25.06.2005

[Bal96] BALZERT, Helmut: *Lehrbuch der Softwaretechnik*. Spektrum Akademischer Verlag, 1996. – ISBN 3-8274-0042-2

[Bin] BINGHAM, Harvey. *SGML Syntax Summary*. http://xml.coverpages.org/sgmlsyn/sgmlsyn.htm#P2, verifiziert am 29.06.2005

[BL] BERNERS-LEE, Tim. *The World Wide Web project*. http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html, verifiziert am 05.05.2005

[CCMW] CHRISTENSEN, Erik ; CURBERA, Francisco ; MEREDITH, Greg ; WEERAWARANA, Sanjiva. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl, verifiziert am 11.07.2005

[CD] CLARK, James ; DEROSE, Steve. *XML Path Language (XPath) Version 1.0*. http://www.w3.org/TR/xpath, verifiziert am 30.06.2005

[Claa] CLARK, Andy. *CyberNeko HTML Parser*. http://people.apache.org/ãndyc/neko/doc/html/index.html, verifiziert am 29.06.2005

[Clab] CLARK, James. *XSL Transformations (XSLT) Version 1.0*. http://www.w3.org/TR/xslt#section-Introduction, verifiziert am 25.06.2005

[cona] CONSORTIUM, World Wide W. *Markup Validation Service v0.6.7*. http://validator.w3.org/, verifiziert am 08.07.2005

[conb] CONSORTIUM, World Wide W. *W3C World Wide Web Consortium*. http://www.w3.org/, verifiziert am 12.07.2005

[Coo] COOPERATION, Netscape. *PERSISTENT CLIENT STATE, HTTP COOKIES*. http://wp.netscape.com/newsref/std/cookie_spec.html, verifiziert am 26.06.2005

[Cor] CORPORATION, OneName. *Requirements for a Global Identity Management Service*. http://www.w3.org/2001/03/WSWS–popa/paper57, verifiziert am 18.07.2005

[DF] DAVID FALLSIDE, Yves L. *XML Protocol Working Group Charter*. http://www.w3.org/2000/09/XMLProtocolCharter, verifiziert am 15.04.2005

[Diva] DIVERSE. *RFC 2616 Fielding, et al.* http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html, verifiziert am 12.07.2005

[Divb] DIVERSE. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. http://www.w3.org/TR/xhtml1/#xhtml, verifiziert am 25.06.2005

[Div81] DIVERSE. *TRANSMISSION CONTROL PROTOCOL*. http://www.faqs.org/rfcs/rfc793.html, verifiziert am 12.07.2005. 1981

[Div02] DIVERSE: *HTML Web-Publishing Handbuch*. 2002. – ISBN 3-7723-7516-2

[Dum] DUMBILL, Edd. *The Semantic Web: A Primer*. http://www.xml.com/pub/a/2000/11/01/semanticweb/, verifiziert am 25.06.2005

[EG04] ERICH GAMMA, Ralph Johnson John V.: *Entwurfsmuster*. Addison-Wesley, 2004. – ISBN 3-8273-2199-9

[Fou] FOUNDATION, The Apache S. *Apache Tomcat*. http://jakarta.apache.org/tomcat/, verifiziert am 08.07.2005

[Giu] GIUSTINA, Fabrizio. *JTidy Home*. http://jtidy.sourceforge.net/, verifiziert am 29.06.2005

[Gmb] GMBH, HBT. *GEOFOX Persönlicher Fahrplan*. http://www.geofox.de, verifiziert am 08.07.2005

[Gol] GOLD, Russell. *HttpUnit*. http://httpunit.sourceforge.net/, verifiziert am 25.06.2005

[Gol91] GOLDFARB, Charles F.: *The Sgml Handbook*. Oxford University Press, 1991. – ISBN 0-1985-3737-9

[inc] INC., Google. *Google*. http://www.google.de/, verifiziert am 08.07.2005

[Kah98a] Definition 4.6.6 In: KAHLBRANDT, Bernd: *Software-Engineering*. 1998, S. 89. – ISBN 3-540-63309-X

[Kah98b] KAHLBRANDT, Bernd: *Software-Engineering*. 1998. – ISBN 3-540-63309-X

[Kou] KOUZOUBOV, Kirill. *JAVA SOCKS Server*. http://jsocks.sourceforge.net/index.html, verifiziert am 25.06.2005

[Lim] LIMITED, Opodo. *opodo Travel Your Way*. http://www.opodo.de, verifiziert am 08.07.2005

[Lyn] LYNCH, Kevin. *The Flash Platform*. http://www.macromedia.com/platform/whitepapers/platform_ov verifiziert am 12.07.2005

[Mün] MÜNZ, Stefan. *SELFHTML*. http://www.selfhtml.org/, verifiziert am 26.06.2005

[OMGa] OBJECT MANAGEMENT GROUP, Inc. *INTRODUCTION TO OMG'S SPECIFICATIONS*. http://www.omg.org/gettingstarted/specintro.htm, verifiziert am 10.07.2005

[OMGb] OBJECT MANAGEMENT GROUP, Inc. *Object Management Group*. http://www.omg.org/, verifiziert am 10.07.2005

[RHJa] RAGGETT, Dave ; HORS, Arnaud L. ; JACOBS, Ian. *Grouping elements: the DIV and SPAN elements*. http://www.w3.org/TR/html401/struct/global.html#h-7.5.4, verifiziert am 05.07.2005

[RHJb] RAGGETT, Dave ; HORS, Arnaud L. ; JACOBS, Ian. *HTML 4.01 Specification*. http://www.w3.org/TR/html401/, verifiziert am 25.06.2005

[RHJc] RAGGETT, Dave ; HORS, Arnaud L. ; JACOBS, Ian. *Introduction to SGML*. http://www.w3.org/TR/html401/intro/sgmltut.html#h-3.1, verifiziert am 25.06.2005

[SD] STEFAN DECKER, Michael S. *The Semantic Web Community Portal*. http://www.semanticweb.org/index_old.html, verifiziert am 25.06.2005

[SM] SUN MICROSYSTEMS, Inc. *Applets*. http://java.sun.com/applets/, verifiziert am 24.06.2005

[SSa] STRÜBIG, Joachim ; SCHÄFER, Mathias. *Allgemeines Block-Element*. http://de.selfhtml.org/html/text/bereiche.htm#block, verifiziert am 05.07.2005

[SSb] STRÜBIG, Joachim ; SCHÄFER, Mathias. *SELFHTML, cookie*. http://de.selfhtml.org/javascript/objekte/document.htm#cookie, verifiziert am 25.06.2005

[Ste04] STEFFEN, Gunnar: *Design einer Mitfahrerbörse auf Basis graphentheoretischer Verfahren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2004

[Ver] VERSCHIEDENE. *Durchgängige Elektronische FahrplanInformation*. http://www.delfi.de, verifiziert am 14.04.2005

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. Juli 2005	
Ort, Datum	Unterschrift