Universität Leipzig
Institut für Informatik
Lehrstuhl für Angewandte Telematik / e-Business
Prof. Dr. Volker Gruhn

Diplomarbeit

Spezifikation und Steuerung Ajax-basierter Nutzerinteraktion im Dialog Control Framework

Autor:	Michael Becker
Matrikelnummer:	9160486
Studiengang:	Informatik (Diplom) 13. Semester

Betreuer:	Dipl. Inf. Matthias Book
Erstgutachter:	Prof. Dr. Volker Gruhn

Eingereicht am:

Zusammenfassung

Benutzer moderner Webanwendungen verlangen von diesen Interaktionsmöglichkeiten, wie sie von Desktopanwendungen bekannt sind. Aufgrund der Limitierungen der zugrunde liegenden Technologien war es bisher nicht möglich, diese neuen Möglichkeiten zu implementieren. Mit dem Aufkommen Ajax-basierter Nutzerinteraktion können die Defizite der Technologien verdeckt und dynamischere Webanwendungen, die unmittelbar auf Aktionen des Benutzers reagieren entwickelt werden.

Die vorliegende Arbeit führt in die Grundlagen der Ajax-Kommunikation ein und zeigt, wie sich diese in das Dialog Control Framework integrieren lässt. Dazu wird eine Ajax-Komponente entwickelt und deren Verwendung vorgestellt.

Inhaltsverzeichnis

1 Einleitung

Die Anzahl an Anwendungen, welche über das World Wide Web aufgerufen und benutzt werden, ist in den letzten Jahren enorm angestiegen. Waren früher Seiten mit statischem Inhalt vorherrschend, werden diese heutzutage durch verteilte, dynamische Anwendungen abgelöst: e-Commerce, Online-Banking, webbasierte Buchungssysteme sowie Auktionsplattformen sind nur einige davon. Zur Benutzung dieser Anwendungen werden interaktive, bedienungsfreundliche Benutzeroberflächen (GUI) benötigt, deren Gestaltung sich an den Erfahrungen aus der Desktop-Welt anlehnt. Man spricht daher heute nicht mehr von Webseiten, sondern von Webanwendungen.

Einhergehend mit dem Anstieg der Dynamik von Webanwenwendungen ist auch deren Komplexität gewachsen. Es ist daher notwendig geworden, Methoden und Werkzeuge zu finden, mit denen sich auch größere Projekte umsetzen lassen. Um den Entwurf von Webanwendungen zu vereinfachen, hat sich die Aufteilung auf die Schichten Model (Anwendungsdaten und die eigentliche Geschäftslogik), View (Präsentation) sowie Controller (Anwendungssteuerung) bewährt; daraus ergibt sich das sogenannte MVC-Pattern.

Die Grundprinzipien des Internets und damit von Webanwendungen sind trotz der enormen Entwicklungen der letzten Jahre gleich geblieben. Es mussten also Wege gefunden werden, wie sowohl die gestiegenen Anforderungen an die Interaktionsmöglichkeiten als auch die immer komplexer werdende Geschäftslogik einer Anwendung gehandhabt werden können. Web-Frameworks wie z.B. Jakarta Struts oder Java Server Faces (JSF) unterstützen Entwickler bei der Aufteilung einer Webanwendung in die genannten Schichten des MVC-Patterns.

1.1 Dialogflusssteuerung

Bei der Umsetzung des MVC-Patterns hat sich die unabhängige Implementierung des Controllers als besondere Schwierigkeit herausgestellt. Der Controller umfasst die gesamte Dialogflusssteuerung, also die eigentliche Navigation, einer Webanwendung. Anhand eines vorher spezifizierten Ablaufs wird durch Eingaben des Benutzers entschieden, welche Seite als nächstes angezeigt wird.

Die meisten Web Frameworks unterstützen die Spezifikation von Dialogflüssen mit Hilfe einfacher Konfigurationsdateien. Allerdings erschöpfen sich die Möglichkeiten meist in der losen Kopplung einzelner Webseiten untereinander. Die Modellierung komplexerer Dialogflüsse, die z.B. auf Abhängigkeiten beruhen, ist dagegen nicht möglich.

Durch die Verwendung des Dialog Control Frameworks (DCF) kann dieser Defizit ausgeglichen werden. Das DCF unterstützt Entwickler bei der Behandlung von Prozessabläufen in Webanwendungen. Es dient der Spezifikation und Steuerung von Dialogflüssen [BG04]. In einer auf dem MVC-Pattern aufbauenden Webanwendung übernimmt das DCF einen Teilbereich der Aufgaben des Controllers - die Dialog-

flusssteuerung. Dazu übersetzt es Dialoggraphen, die mittels der Dialog Flow Notation (DFN) erstellt wurden, in die Dialog Flow Specification Language (DFSL), eine XML-basierte Spezifikationssprache. Das eigentliche Framework liest diese Dialogflussspezifikation ein und wandelt sie in ein objektorientiertes Dialogflussmodell um.

Die Diagramme der DFN sind gerichtete Graphen, welche sich aus Dialogelementen (Knoten des Graphen) und Ereignissen zwischen diesen Elementen (Transitionen im Graphen) zusammensetzen. Die Dialogelemente können dabei sowohl Masken sein, die als Seite im Browser des Anwenders angezeigt werden, oder auch Actions, welche die Anwendungslogik auf dem Server implementieren. Durch die Verknüfung von Masken, Aktionen und Ereignissen lassen sich die Interaktionsmöglichkeiten einer Anwendung spezifizieren. Um eine Wiederverwendbarkeit bereits spezifizierter Nutzerinteraktionen zu gewährleisten, können Dialogteile in Modulen, welche sich an beliebiger Stelle aufrufen lassen, gekapselt werden.

1.2 Interaktionsmöglichkeiten von Webanwendungen

Bei der Arbeit mit Webanwendungen kommt es in vielen Fällen (z.B. Absenden eines Formulars, Klick auf einen Link) zu Wartezeiten, da Daten verschickt und neue Seiten angezeigt werden müssen. Dies führt zu einer Störung des Arbeitsflusses des Benutzers und damit zu einer Senkung der Produktivität.

Es gibt mehrere Möglichkeiten, dieses Defizit zu beheben. So lassen sich komplexe Benutzeroberflächen mit Java Applets oder Macromedia Flash erstellen. Diese erlauben außerdem Interaktionsmöglichkeiten, die unabhängig von den Limitierungen der zugrunde liegenden Technologien sind. Allerdings haben sie einen entscheidenden Nachteil: Sie benötigen ein Plugin, was Benutzer der Anwendung erst installieren müssen.

Durch die Verwendung von Asynchronous Javascript and XML (Ajax) ist es möglich, ohne Plugins neue Interaktionsmöglichkeiten in Webanwendungen zu integrieren. Ajax beruht auf einer Reihe standardisierter Technologien, die in allen graphikfähigen Browsern[1] verfügbar sind.

Hinter Ajax steckt die Idee, einen flüssigeren Arbeitsablauf zu gewährleisten. Dies wird durch das bedarfsgerechte Nachladen benötigter Daten statt dem Aufruf neuer Seiten erreicht. Diese Daten werden dann zur Verfügung gestellt, indem die Seite, auf welcher sich der Benutzer gerade befindet, aktualisiert wird.

1.3 Ziele der Arbeit

Durch die Verwendung von Desktopanwendungen sind Benutzer Interaktionsmöglichkeiten gewöhnt, die in Webanwendungen aufgrund der technischen Limitierun-

[1] Hiermit sind im Allgemeinen Browser gemeint, die nach 1997 erschienen sind und auf dem Internet Explorer bzw. der Gecko- oder WebKit-Engine aufbauen. Zu Zweitem gehören unter anderem Safari, Firefox und Google Chrome.

gen nicht ohne weiteres realisierbar sind. Die Arbeit soll diese Defizite aufzeigen und erläutern, wie sie mittels Ajax aufgehoben werden können.

Dazu soll eine Komponente für das DCF entwickelt werden, die Ajax-basierte Interaktion im Framework ermöglicht. Zur Spezifikation der Ajax-Kommunikation wird die DFN und DFSL erweitert. Das eigentliche DCF wird angepasst, so dass es per Ajax initiierte Kommunikation verarbeiten kann.

Da das DCF nur einen Teilbereich des MVC-Patterns abdeckt, wird es mit JSF integriert, um komplexere graphische Benutzeroberflächen zu ermöglichen. Mittels JSF lässt sich die Präsentationsschicht einer Anwendung als Komponentenbaum modellieren, wodurch vom zugrunde liegenden Ausgabeformat abstrahiert wird. Aufgrund bereits vorhandener standardisierter Komponenten können komplexe Benutzeroberflächen mit geringem Zeitaufwand erstellt werden.

Die Integration mit JSF ermöglicht weiterhin die Verwendung von Frameworks, die reichhaltige JSF-Komponenten bereitstellen. Diese abstrahieren von den technischen Grundlagen der Ajax-Kommunikation und ermöglichen das Erstellen robusterer Webanwendungen.

1.4 Aufbau der Arbeit

Zunächst soll in Kapitel 2 in das Thema Webanwendungen eingeführt und der aktuelle Stand der Technik dargestellt werden. Es wird ein Überblick über die Technologien von Webanwendungen gegeben und auf Probleme, die sich durch den Einsatz dieser Technologien ergeben, eingegangen.

Kapitel 3 stellt die theoretischen Grundlagen von Ajax vor, welches verwendet wird um Defizite in der Nutzerinteraktion klassischer Webanwendungen zu umgehen. Der Aufbau und das Verhalten Ajax-basierter Anwendungen wird erläutert woran sich eine Übersicht über die Vorteile und Nachteile bei der Verwendung von Ajax anschließt.

Im vierten Kapitel werden Möglichkeiten diskutiert, wie das DCF mit Ajax kombiniert werden kann. Darin inbegriffen ist die Erweiterung der Notation der DFN und DFSL. Da das DCF mit dem JSF Framework kombiniert werden soll, werden Möglichkeiten vorgestellt, Ajax-basierte Nutzerinteraktionen in das JSF zu integrieren. Darauf folgt ein Überblick über existierende Ajax-Frameworks, die das Erstellen Ajax-basierter Webanwendungen vereinfachen.

Nachdem die drei Technologien DCF, JSF und Ajax miteinaner integriert wurden werden Ajax-Frameworks vorgestellt, die das Erstellen Ajax-basierter Webanwendungen vereinfachen.

Die eigentliche Implementierung der Ajax-Komponente wird in Kapitel 5 näher beleuchtet. Dabei wird auf Besonderheiten und Probleme eingegangen und dokumentiert, welche Teile des DCF angepasst wurden.

Um die Verwendung des DCF mit der Ajax-Komponente zu verdeutlichen, wurde eine Anwendung entwickelt, die auf dieser Komponente beruht und in Kapitel 6 vorgestellt wird. Es wird gezeigt, wie die neuen Interaktionsmöglichkeiten von der Spezifikation mittels der DFN und DFSL bis zur eigentlichen Implementierung genutzt werden.

Einen Überblick über verwandte Arbeiten und Ansätze sowie eine Diskussion über die Arbeit geben die letzten beiden Kapitel. Dabei soll die Arbeit noch einmal kritisch betrachtet und weitere Arbeitsmöglichkeiten auf dem Gebiet aufgezeigt werden.

2 Webanwendungen

In diesem Kapitel werden die technischen Grundlagen, die zur Entwicklung robuster Webanwendungen notwendig sind, vorgestellt. Aufgrund der verwendeten Technologien ergeben sich Probleme und Einschränkungen, welche dann dargestellt werden. Es werden Nutzerinteraktionen vorgestellt, die sich mit klassischen Webanwendungen nicht oder nur mit großem Aufwand umsetzen lassen.

Webanwendungen sind Programme, die auf einem Webserver ausgeführt werden. Die Interaktion zwischen dem Nutzer der Anwendung und der Anwendung selber erfolgt dabei durch einen Browser, der auf dem Rechner des Nutzers (Client) installiert ist, welcher sich mit einem Server verbindet. Ein Netzwerk (in der Regel das Internet) sorgt für die Verbindung dieser beiden Komponenten. Damit diese Verbindung korrekt funktioniert, müssen beide Punkte ein gemeinsames Protokoll implementieren, mit dem Daten ausgetauscht werden können[DLWZ03].

Durch die zunehmende Verfügbarkeit des Internets weltweit sowie dem stetigen Anstieg der vorhandenen Bandbreite hat die Verwendung von Webanwendungen enorm zugenommen. Teilweise ersetzen sie die Verwendung klassischer Desktopanwendungen[2]. Dabei bieten Webanwendungen diesen gegenbüber eine Reihe von Vorteilen, die unter anderem in [Con99] dargelegt sind:

- Zur Verwendung der Anwendung ist keine Installation und Konfiguration fremder Software nötig, da Webanwendungen im Browser des Benutzers ausgeführt werden.
- Es werden - aufgrund der Ausführung im Browser - verschiedene Plattformen und Betriebssysteme unterstützt.
- Benutzer einer Webanwendung arbeiten immer mit der neuesten Version, welche zentral auf einem Server liegt. Dadurch entfällt das dezentrale Einspielen von Updates.

Im Gegensatz zu klassischen Client-Server-Anwendungen gibt es bei Webanwendungen einige Besonderheiten. So ist unter anderem die Verbindung vom Browser zum Server durch die Verwendung des Hypertext Transfer Protocols (siehe unten) zustandslos. Bei der Entwicklung von Webanwendungen ist weiterhin darauf zu achten, dass diese im Browser des Benutzers dargestellt werden. Aufgrund der unterschiedlichen Browserversionen muss das Layout einer Anwendung daher flexibel sein.

2.1 Technische Grundlagen von Webanwendungen

Webanwendungen basieren auf einer Vielzahl unterschiedlicher - in der Regel standardisierter - Technologien und Komponenten. Einerseits gibt es die Softwarekom-

[2] Siehe auch: New Survey Shows Use of Web Applications Is Spreading Rapidly, http://www.prweb.com/pdfdownload/556690/pr.pdf.

ponenten Client (in der bekanntesten Form als Browser), Server und Proxy. Daneben gibt es auch die Hintergrundtechnologien wie unter anderem Kommunikationsprotokolle und Auszeichnungssprachen. Diese sind notwendig, damit das Web an sich überhaupt funktionieren kann und die Anwendungen auf allen Clients gleich aussehen und die gleiche Funktionalität besitzen. Im Folgenden soll zuerst das zugrunde liegende Kommunikationsprotokoll vorgestellt werden, woran sich eine kurze Übersicht über die beteiligten Softwarekomponenten anschließt. Danach werden die verschiedenen Sprachen vorgestellt, die notwendig sind, um Webanwendungen auf dem Client darzustellen und anzupassen.

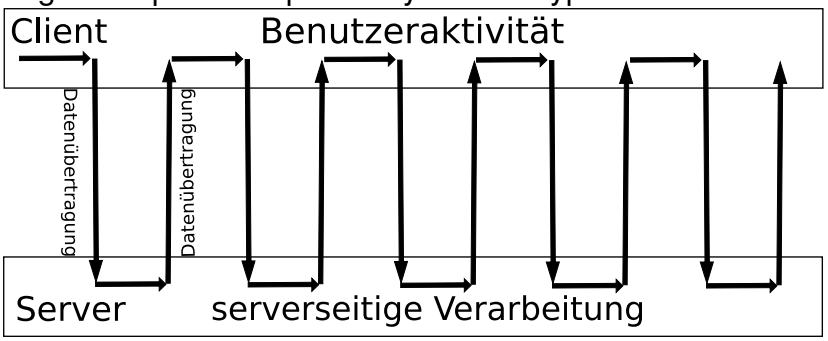
2.1.1 Kommunikation: Hypertext Transfer Protocol

Alle Aktionen, die Benutzer einer Webanwendung ausführen, setzen eine standardisierte, wohlgeformte Möglichkeit der Kommunikation zwischen Client und Server voraus. Das Hypertext Transfer Protocol (HTTP) ist das gebräuchlichste Protokoll, um Daten über das Internet zu verschicken. Es definiert die Syntax und die Semantik der Nachrichten, die zwischen zwei Endpunkten (Client und Server) ausgetauscht werden.

Das HTTP ist ein generisches Request-Response-Protokoll. Dies bedeutet, dass ein Client eine Anfrage (Request) an den Server sendet und dieser mit einer Antwort (Response) darauf reagiert. In der Regel werden die Anfragen durch eine Aktion des Nutzers, wie zum Beispiel dem Klick auf einen Link, ausgelöst. Die Kommunikation verläuft dabei immer synchron - auf eine Anfrage folgt immer eine Antwort. Jeder Nachrichtenaustausch wird durch die Zustandslosigkeit des Protokolls als eigenständige und unabhängige Transaktion behandelt [KR01].

Im Folgenden wird das Absenden einer Anfrage und das Erhalten der zugehörigen Antwort als Request-Response-Zyklus bezeichnet. Das Prinzip der Kommunikation mittels des HTTP in einem Zyklus ist in Abbildung 1 verdeutlicht.

Abbildung 1: Request-Response-Zyklus im Hypertext Transfer Protocol



HTTP wurde im März 1990 von Tim Berners-Lee am Cern konzipiert und ist das grundlegende Protokoll des World Wide Web. Es liegt aktuell in der Version HTTP/1.1[3] aus dem Jahr 1999 vor, in der technische Verbesserungen und Anpassungen

[3]Das Original-Proposol von Berners-Lee kann unter http://info.cern.ch/Proposal.html eingese-

des Protokolls an die Nutzung des Internets vorgenommen wurden. Eine ausführliche Übersicht über die Änderungen zwischen den Protokollspezifikationen findet sich in [KMK99].

Im dritten Kapitel wird gezeigt, wie man durch die Verwendung der Ajax-Technologie die synchrone Arbeitsweise des HTTP verdeckt und damit einen flüssigeren Anwendungsablauf ermöglicht. Kenntnise des Protokolls sind weiterhin für ein tiefer gehendes Verständnis der Funktionalität von und zur Fehlersuche in Ajax-Anwendungen von Bedeutung. So müssen die Statuscodes eines HTTP-Responses von der Anwendung ausgewertet werden, damit diese fehlerfrei funktionieren kann. Eine kurze Übersicht weiterer heutzutage genutzter Internetprotokolle findet sich in [SG03].

2.1.2 Softwarekomponenten: Client, Server, Proxy

Um Webanwendungen wirklich effektiv zu verwenden, benötigt der Benutzer Softwarekomponenten, welche die jeweilige Anwendung bereitstellen, anzeigen und den Zugriff auf diese ermöglichen. Im folgenden Abschnitt werden die Komponenten vorgestellt, mit welchen diese drei Aufgaben durchgeführt werden können.

2.1.2.1 Client

Nutzer erhalten Zugriff auf eine Webanwendung mittels eines Browsers, welcher die Daten der Anwendung mit Hilfe einer graphischen Benutzeroberfläche darstellt. Der Browser ist die bekannteste Form des Clients und verantwortlich für das Senden von Anfragen und Empfangen von Antworten mittels des HTTP (siehe 2.1.1) sowie der Darstellung des Inhalts durch die Interpretation von Dokumenten einer standardisierten Auszeichnungssprache (siehe 2.1.3).

Da Ajax-Anwendungen regen Gebrauch von clientseitigen Funktionalitäten machen, ist es wichtig, die Unterschiede zwischen den einzelnen Clients (im Besonderen zwischen verschiedenen Browsern) zu kennen. Auch wenn die eigentlichen Ajax-Aufrufe durch die Verwendung eines bereits bestehenden Frameworks eventuell gekapselt werden, kann es zu unterschiedlichen Verhaltensweisen kommen. Um diese zu erkennen und zu umgehen muss bei der Entwicklung von Anwendungen auf die Eigenheiten der einzelnen Browser eingegangen werden.

2.1.2.2 Server und Proxy

Bei der Arbeit mit einer Webanwendungen werden HTTP-Requests des Benutzers vom Browser aus an den Server gesendet. Der Server ist für die Auswertung des Requests und das Senden der Response zuständig. Bei den meisten heutzutage gebräuchlichen Webanwendungen ist die eigentliche Anwendungslogik auf dem

hen werden. Die Spezifikationen zu HTTP/1.0 und HTTP/1.1 finden sich als RFC 1945 unter http://tools.ietf.org/html/rfc1945 sowie als RFC 2616 unter http://tools.ietf.org/html/rfc2616.

Server implementiert. Dazu lassen sich eine Reihe von Programmiersprachen verwenden, mit denen dynamisch Dokumente erstellt werden, die an den Browser zurückgeschickt werden.

Bei der Integration von Ajax in bestendende Webanwendungen muss darauf geachtet werden, dass diese serverseitig korrekt erkannt und bearbeitet werden, da im Gegensatz zu normalen Anfragen keine vollständigen Webseiten sondern nur Daten zurückgeliefert werden. Der Server muss also die verschiedenen Request-Typen erkennen und dementsprechend behandeln.

In der Regel erfolgt die Kommunikation zwischen einem Client und einem Server im Internet nicht direkt. Stattdessen kommunizieren sie über so genannte Proxies, die als Bindeglied dienen. Sie übernehmen Aufgaben wie Caching oder Load Balancing [LA94].

2.1.3 Auszeichnung und Stil: Hypertext Markup Language / Cascading Style Sheets

Zur Auszeichnung von Webseiten wird die Hypertext Markup Language (HTML) verwendet. HTML-Dokumente sind die Grundlage des World Wide Web und werden in den jeweiligen Browsern dargestellt [Wil99].

HTML ist eine Anwendung der Standard Generalized Markup Language (SGML), deren erste Version 1989 von Tim-Berners Lee zusammen mit der Spezifikation des HTTP am Cern vorgestellt wurde. Aktuell liegt es in der Version 4.01[4] vor. Mittlerweile gibt es mit der Extensible Hypertext Markup Language (XHTML) einen Nachfolger, welcher nicht mehr direkt auf SGML aufsetzt sondern auf XML basiert.

Mit HTML lassen sich die logischen Bestandteile eines Textdokumentes beschreiben. Dazu gibt es sogenannte Tags, die bestimmte Elemente des Dokuments repräsentieren. Dies können unter anderem Tabellen, Listen oder auch Formulare sein, welche sich in einander verschachteln lassen, so dass es auch möglich ist, komplexe Dokumente zu beschreiben. Am Ende ergibt sich ein hierarchisch gegliedertes System verschiedener Tags, die von den Browsern ausgelesen werden.

Eine weitere Technik, die in Webanwendungen häufig zum Einsatz kommt, ist Cascading Style Sheets (CSS). Damit lassen sich HTML-Dokumente formatieren, wordurch die Präsentation vom eigentlichen logischen mittels HTML definierten Aufbau der Seite getrennt wird. Es ist möglich festzulegen, wie einzelne Elemente eines Dokuments am Bildschirm (oder auch an anderen Ausgabemedien) dargestellt werden sollen. Einen guten Überblick über die Formatierungsmöglichkeiten mit CSS bietet die Doktorarbeit von Håkon Wium Lie [Lie05], der CSS mitentwickelt hat. Auch CSS ist durch das W3C standardisiert[5], wodurch die Standardsprachbestandteile in allen Browsern gleich interpretiert werden.

Jede Webanwendung und damit auch jede Ajax-Anwendung benötigt als Grundge-

[4] Die zugehörige Spezifikation findet sich unter http://www.w3.org/TR/html4/.
[5] Die aktuelle Spezifikation findet sich unter http://www.w3.org/Style/CSS/.

rüst HTML-Dokumente. Sie sind daher essentielle Bestandteile dieser Anwendungen. Wichtig sind HTML- sowie CSS- Kenntnisse ferner, wenn Anwendungen auf unterschiedliche Endgeräte portiert werden sollen. So unterstützen die meisten mobilen Browser das Layouting mittels Tabellen nicht, wodurch Entwickler auf eine strikte Trennung zwischen Inhalt und Layout hinarbeiten müssen. Grundlegende Kenntnisse der beiden Techniken sind daher auch für die Verwendung des DCF notwendig, welches darauf ausgelegt ist, mehrere Endgerätetypen zu unterstützen. Auch bei der Kommunikation mittels Ajax muss durch die Verwendung des Document Object Models (siehe 3.2.1) auf die korrekte Verwendung der HTML-Spezifikation geachtet werden. Wird kein valides HTML verwendet, kann der Browser Inhalte unter Umständen nicht oder nur teilweise darstellen.

2.1.4 Dynamisierung: Javascript

Damit Ajax-Anwendungen überhaupt funktionieren, benötigt man auf der Clientseite eine Programmiersprache, mit der Anwendungslogik ausgeführt werden kann. Wie das Akronym Ajax bereits andeutet ist das Mittel der Wahl dabei Javascript.

Bei Javascript handelt es sich um eine dynamisch typisierte, universal einsetzbare, objektbasierte Sprache, die zur Laufzeit der Anwendung im Browser interpretiert wird. In Ajax-Anwendungen kommt Javascript unter anderem zum Einsatz, um das HTML-Gerüst einer Webseite dynamisch zu ändern, wenn neuer Inhalt verfügbar ist. Aber auch andere Aufgaben wie Validierung von Eingaben oder ähnlichem können durch Skripte übernommen werden. Durch die Standardisierung zentraler Sprachbestandteile unter dem Namen EcmaScript[6] ist die Sprache heutzutage in allen Browsern verfügbar.

Javascript wird in einem HTML-Dokument durch das Tag `script` eingebunden, welches Code enthält, der im Browser des Benutzers ausgeführt werden soll. Da es mittels eines Skriptes möglich ist den Aufbau des zugrunde liegenden HTML-Dokuments zu ändern, können auch sich selbst modifizierende Skripte entworfen werden, indem neuer Javascript-Code in das Dokument eingefügt wird.

Im Folgenden sollen einige besondere Sprachbestandteile von Javascript vorgestellt werden. Da die Ausführung der Skripte komplett clientseitig abläuft, ergeben sich natürlich einige besondere Sicherheitsbedenken, die erläutert werden. Für eine ausführliche Darstellung sowie tiefergehende Informationen ist hier auf [Zak05] verwiesen.

2.1.4.1 Besondere Sprachbestandteile

Im Gegensatz zu rein objektorientierten Sprachen ist Javascript prototypenbasiert. Objekte sind die Prototypen für neue Objekte; Klassen im herkömmlichen Sinne exi-

[6] EcmaScript ist standardisiert als ECMA-262: ECMAScript Language Specification; dies kann unter http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf eingesehen werden.

stieren nicht. Neue Objekte werden durch Kopieren bereits vorhandener statt durch Instantieren einer Klasse erstellt [DMC92] und [Bor86].

Objekte in Javascript sind dabei eine Zusammenfassung von Werten und Funktionen. In jeder Implementierung finden sich die folgenden Standardobjekte:

- Object: allgemeiner Prototyp, von dem alle anderen Objekte abgeleitet sind,
- Function: Prototyp für Funktionen,
- Array, String, Boolean, Number: Prototypen für die gleichnamigen Datentypen,
- Math, Date, RegExp: Prototypen für die Arbeit mit mathematischen Funktionen, Daten sowie regulären Ausdrücken.

Weiterhin implementiert Javascript das Konzept der Closures. Dabei handelt es sich - vereinfacht dargestellt - um Funktionen, die in anderen Funktionen notiert werden. Die innere Funktion hat dabei Zugriff auf die Variablen der äußeren Funktion, selbst wenn diese nicht mehr ausgeführt wird. Closures sind ein Konzept der funktionalen Programmierung und in [Loo95] näher erläutert.

Die Anwendung von Prototypen sowie Closures ist wichtig für die Erweiterungsmöglichkeiten von Javascript in Form von Frameworks, auf die in Kapitel 4 eingegangen wird. Erst durch diese Erweiterungen ist es möglich, komplexere Ajax-Anwendungen zu schreiben, die auf allen Browsern korrekt funktionieren.

2.1.4.2 Javascript und Sicherheit

Da Javascript im Browser des Benutzers ausgeführt wird, sind standardmäßig einige Sicherheitsfunktionen aktiviert, die auch bei der Verwendung im Ajax-Kontext zu beachten sind. Durch diese eingebauten Maßnahmen erhöht sich einerseits die Sicherheit für die Benutzer, allerdings können dadurch bestimmte Funktionalitäten nicht in Anwendungen integriert werden.

Eine wichtige Einschränkung bei der Verwendung von Javascript ist die Abarbeitung der Skripte in einer so genannten Sandbox. Dadurch beschränken sich die Möglichkeiten der Interaktion eines Skriptes auf die Objekte, die der Browser anbietet. Es kann nicht auf das lokale Dateisystem zugegriffen werden um z.B. Dateien automatisch zu speichern. Außerdem erfordern bestimmte Aktionen eine Bestätiung durch den Benutzer (Schließen des Fensters, Auslesen der Browser-History, usw.)

Eine weitere integrierte Sicherheitsfunktion ist die Same Origin Policy. Dadurch können Skripte nur mittels des gleichen Protokolls auf Ressourcen zugreifen, die auf der gleichen Domain liegen sowie den gleichen Port besitzen. Es ist also nicht möglich, in einer mittels HTTP übertrageneneWebseite auf Inhalte zuzugreifen, die mittels eines anderen Protokolls (wie z.B. File Transfer Protocol) erreichbar sind. Dieser Schutz wurde eingebaut, damit auf einer vom Anwender als sicher empfundenen Seite keine Inhalte einer von ihm als unsicher eingestuften Webseite eingebunden werden können.

Bei der Konzeption von Webanwendungen, die Ajax verwenden, muss besonders die Same Origin Policy beachtet werden. Sollen dynamisch Inhalte nachgeladen werden, müssen diese von der gleichen Quelle stammen, wie das eingebettete Javascript. Ist dies nicht der Fall, bricht das Script im Browser des Benutzers mit einer Fehlermeldung ab.

2.2 Technische und organisatorische Probleme von Webanwendungen

Aufgrund der Technologien, auf denen Webanwendungen basieren, ergeben sich verschiedene Probleme, auf welche in diesem Abschnitt eingegangen werden soll. Das grundlegende Problem von Webanwendungen ist die Beschränkung auf den synchronen Request-Response-Zyklus des HTTP. Dabei muss bei jeder Datenänderung die gesamte HTML-Seite neu geladen werden, was bei der Übertragung großer Datenmengen zu unerwünschten Wartezeiten führen kann. Gibt es bei der Datenübertragung Verzögerungen so verzögert sich dadurch auch der eigentliche Prozessablauf.

2.2.1 Souveräne und transiente Webanwendungen

Interaktionsmöglichkeiten von Anwendungen lassen sich wie folgt klassifizieren:

- Echtzeit: Die Anwendung muss ständig auf Eingaben des Benutzers reagieren und führt auch Aufgaben im Hintergrund aus.
- Verzögerungsresistent: Der Benutzer arbeitet mit der Anwendung und überführt diese durch eine definierte Aktion in den nächsten Zustand.

In [CR03] werden Webanwendungen in die beiden Kategorien souverän (Echtzeit) und transient (Verzögerungsresistent) unterteilt. Bei einer souveränen Anwendung fallen Verzögerungen bei der Arbeit mit der Anwendung sehr schnell auf. Beispielhaft sei hier nur eine Textverarbeitung erwähnt, bei der im Hintergrund eine Rechtschreibprüfung durchgeführt wird. Der Arbeitsablauf des Benutzers würde gestört, wenn er immer auf das vollständige Ergebnis der Prüfung warten müsste.

Bei transienten Anwendungen dagegen fallen Verzögerungen nicht so sehr ins Gewicht, da Benutzer nicht im gleichen Maß mit diesen interagieren wie dies bei souveränen Anwendungen der Fall ist. So müssen die Ergebnisse einer Suchanfrage nicht sofort verfügbar sein. Heutige Webanwendungen sind in der Regel transient; als Beispiele seien hier die Suche mittels einer Suchmaschine oder das Abschicken eines Bestellformulars eines Onlineshops angeführt.

Mit den in diesem Kapitel bisher vorgestellten Technologien lassen sich nur transiente Webanwendungen entwickeln. Die Entwicklung souveräner Anwendungen scheitert an der Synchronität des HTTP. Es ist mit dem durch das HTTP vorgegebenen

synchronen Request-Response-Zyklus nicht möglich, im Hintergrund Requests abzuschicken und zu verarbeiten.

2.2.2 Thin Clients

Heutige Browser verfügen über eine Vielzahl von Möglichkeiten, um clientseitig Anwendungslogik auszuführen. Dies ist unter anderem durch die Verwendung von Javascript (siehe Seite 13) sowie die Nutzung des Document Object Models, welches im dritten Kapitel näher beleuchtet wird, möglich.

Viele vorhandene Anwendungen profitieren nicht vom gestiegenen Funktionsumfang der Browser, da sie dem Prinzip folgen, dass der Browser nur den Inhalt einer Anwendung darstellen soll und keinerlei eigene Logik implementieren darf - in diesem Fall spricht man von Thin Clients. Dies erleichtert zwar die Verwendung der Anwendung auf unterschiedlichen Browsern, allerdings werden dadurch viele neue Interaktionsmöglichkeiten, die sich bieten, nicht in Anspruch genommen.

Nach dem serverseitigen Abarbeiten der Geschäftslogik müssen komplett neue Seiten zurückgeliefert werden, wodurch es oftmals zu einer Vermischung von Layout, Inhalt und eigentlicher Logik kommt.

Eine deutliche Verbesserung der Nutzerinteraktion würde sich durch die Verlagerung eines gewissen Teils der Anwendungslogik auf den Client ergeben. Da sich das Aussehen einer Anwendung häufig ändern kann, die dahinter liegende Logik allerdings gleich bleibt, ist der clientseitige Teil für die Darstellung der Anwendung zuständig. Auf dieser Ebene wird dann auf Benutzereingaben reagiert und es können bei Bedarf Anfragen im Hintergrund an den Server geschickt werden.

2.3 Anforderungen an moderne Webanwendungen

Durch die zunehmende Verbreitung und Steigerung der Komplexität von Webanwendungen ergeben sich neue Anforderungen an die Benutzerschnittstellen der Anwendungen. Diese orientieren sich an Desktopanwendungen, für die es inzwischen eine Reihe von Normen gibt[7].

Neben diesen vor allem die Ergonomie betreffenden Anforderungen gibt es noch eine Reihe weiterer Möglichkeiten, die Desktopanwendungen bieten, sich aber nicht ohne Weiteres in Webanwendungen umsetzen lassen. Auf einige dieser Möglichkeiten wird in den folgenden Abschnitten eingegangen.

[7] Die DIN EN ISO 9241 definiert in den Teilen 11 (Anforderungen an die Gebrauchstauglichkeit) sowie 110 (Grundsätze der Dialoggestaltung) die Richtlinien zum Design einer GUI.

2.3.1 Automatisches Speichern

Bei der Verwendung heutiger Desktopanwendungen - insbesondere bei Textverarbeitungen - werden die bisherigen Eingaben des Benutzers oftmals periodisch gespeichert. Dadurch kann er auch im Falle eines Programmabsturzes mit der Arbeit fortfahren, ohne wieder neu beginnen zu müssen.

Da auch Webanwendungen immer komplexer werden, wäre es wünschenswert, wenn die Zustände der Anwendung regelmäßig gespeichert werden. Dies ist in klassischen Webanwendungen durch das Aufteilen der Dateneingabe auf mehrere Seiten möglich, was allerdings aktives Eingreifen des Benutzers erfordert. Besonders bei der Eingabe großer Mengen an Daten auf einer Seite führt ein Verlust dieser Daten beim Programmabsturz zu einer enormen Produktivitätsminderung. Hinzu kommt bei Webanwendungen, dass zur Arbeit mit der Anwendung eine bestehende Internetverbindung benötigt wird. Bricht diese ab, sind die eingegebenen Daten verloren, wenn nicht automatisch gespeichert wurde.

2.3.2 Automatische Validierung und Eingabevorschläge

Ebenfalls aus dem Bereich der desktopbasierenden Textverarbeitung ist die Rechtschreibprüfung sowie das automatische Vervollständigen eingegebener Wörter bekannt. Diesen Eingabehilfen liegt immer eine Datenbankabfrage während der Eingabe zugrunde, die bekannte Wörter bzw. gültige Schreibweisen ermittelt. Dieses Prinzip kann auf Webanwendungen erweitert werden, indem dem Benutzer z.B. passende Kundennummern oder gültige Kennziffern während der Eingabe vorgeschlagen werden, ohne dass er diese in einem sehr großen Auswahhlmenü selektieren muss. Dadurch wird eine intuitivere Oberfläche geschaffen, die eine vereinfachte Datendarstellung ermöglicht.

2.3.3 Geschwindigkeit

Wie bereits beschrieben, basieren klassische Webanwendungen auf dem Request-Response-Paradigma des HTTP. Dadurch wird bei jeder Datenaktualisierung der vollständige Inhalt einer neuen Seite vom Server angefordert. Hat der Benutzer bereits große Datenmengen eingegeben, erzeugt er damit viel Netzwerkverkehr. Dies ist einerseits schlecht für den Benuzer, da die Übertragung großer Datenmengen länger dauert andererseits auch für den Server, da er eben diese Datenmengen immer wieder verarbeiten muss. Beim Neuladen kompletter Seiten wird außerdem der gesamte HTML- und CSS-Code, der nur der Formatierung dient, übertragen. Durch diesen Overhead vergrößert sich die Datenmenge weiter.

Besser wäre es, wenn der Server nur diejenigen Daten sendet, die tatsächlich zur korrekten Darstellung der Informationen notwendig sind. Damit würde die aktuell vorherrschende inhaltszentrierte Kommunikation durch eine datenzentrierte abgelöst werden, was zu einer Verringerung der zu übertragenden Datenmenge führt,

wodurch schnelleres - und damit produktiveres - Arbeiten ermöglicht wird.

2.4 Fazit

In diesem Kapitel wurden die Grundlagen von Webanwendungen dargestellt und auf verschiedene Probleme eingegangen. Weiterhin sind einige Anforderungen, die durch die zunehmende Komplexität der Anwendungen entstehen, dargestellt wurden.

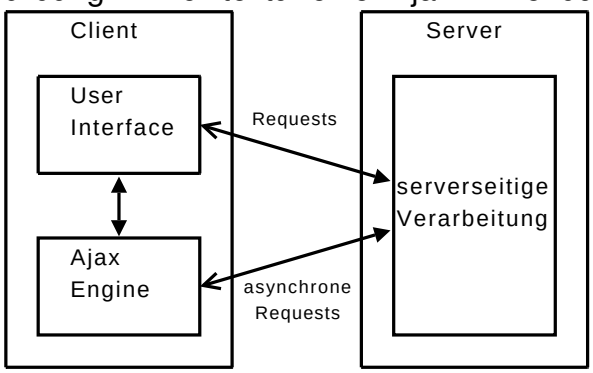
Webanwendungen würden durch die Integration der neuen Nutzerinteraktionen wie automatischem Speichern erheblich profitieren. Daher soll im nächsten Kapitel gezeigt werden, mit welchen Technologien die gezeigten Anforderungen implementiert werden können. Diese bauen auf den hier vorgestellten Grundlagen auf und ermöglichen durch die Erweiterung dieser soliden Basis neue Interaktions- und Kommunikationsmöglichkeiten, um auch einige der vorhanden Probleme von Webanwendungen zu lösen sowie die Modellierung souveräner Webanwendungen zu ermöglichen.

3 Ajax

Im vorangegangen Kapitel wurde gezeigt, welche Probleme es bei der Verwendung von Webanwendungen gibt und wie sich die Benutzbarkeit durch das Erfüllen verschiedener Anforderungen verbessern lässt. Da viele neue Interaktionsmöglichkeiten erst ermöglicht werden, wenn die Synchronität des HTTP verdeckt wird, werden in diesem Kapitel Techniken vorgestellt, mit denen dies möglich ist.

Allen voran sei hier die Technologie Asynchronous Javascript and XML (Ajax) genannt. Damit wird das Zusammenspiel verschiedener Standardtechnologien zur Realisierung asynchroner Kommunikation zwischen einem Client und einem Server bezeichnet, wodurch sich interaktive Webanwendungen entwickeln lassen, die im Gegensatz zu klassischen Webanwendungen ohne Ajax nicht mehr zustandslos scheinen. Dies wird ermöglicht durch das bedarfsgerechte (asynchrone) Nachladen von Daten. Obwohl die Techniken, die hinter Ajax stecken, keine Neuentwicklungen sind, wurde der Begriff selber erst Anfang 2005 in einem Artikel Jesse James Garrets [Gar05] geprägt.

Abbildung 2: Architektur einer Ajax-Anwendung



Durch die Grundideen hinter dem Modell Ajax - das teilweise Nachladen von Seiten sowie die asynchrone Kommunikation zwischen Client und Server - gibt es drei wesentliche Unterschiede Ajax-basierter Anwendungen zu klassischen Webanwendungen, wodurch sich eine Architektur ergibt, wie sie in Abbildung 2 dargestellt ist:

- Aktionen des Benutzers haben keinen direkten Request an den Server zur Folge sondern resultieren in einem Aufruf der clientseitigen Javascript-Komponente.
- Die Response wird nicht direkt dargestellt sondern von der Javascript-Engine entgegen genommen.
- Es werden keine kompletten Seiten zurückgeliefert sondern Daten, die in die aktuelle Seite eingearbeitet werden.

Im Folgenden sollen der Aufbau und die Schlüsseltechnologien von Webanwendungen, die Ajax verwenden, vorgestellt sowie ein Vergleich zu klassischen Webanwendungen gezogen werden.

3.1 Request-Response-Zyklus von Ajax-Anwendungen

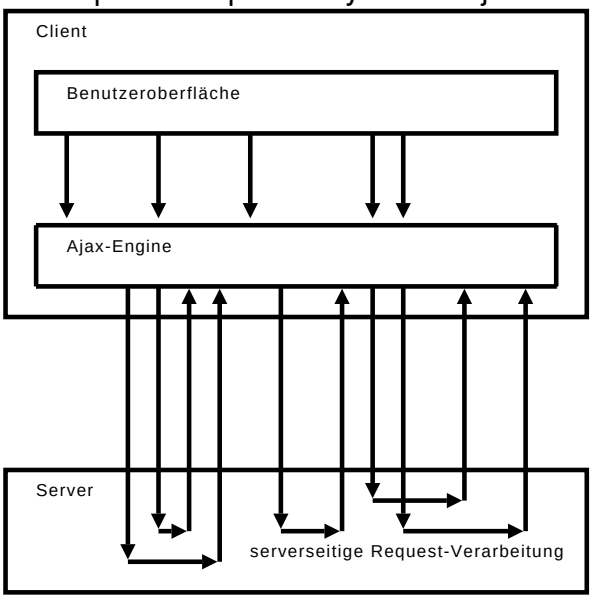
In klassischen Webanwendungen ohne Ajax-Funktionalitäten gibt es zwei grundsätzlich unterscheidbare Zustände der Anwendung:

- Der Browser stellt dem Benutzer Informationen dar.
- Der Benutzer wartet auf eine Antwort des Servers.

In der Zeit, in der Benutzer auf die Antwort des Servers warten, können diese nicht mit der Anwendung interagieren. Dadurch wird der Arbeitsfluss in klassischen Webanwendungen oft unterbrochen, da die Mehrzahl der Anwendungen häufig mit dem Server kommunizieren müssen.

Mit der Verwendung von Ajax können diese Wartezeiten im Idealfall vermieden, mindestens jedoch verkürzt werden. Obwohl die gleichen Anfragen an den Server gestellt werden, kann der Benutzer durch das Paradigma der asynchronen Kommunikation während dieser Zeit mit der Anwendung interagieren. Dadurch ergibt sich aus Benutzersicht in Ajax-Anwendungen ein anderer Request-Response-Zyklus als in klassischen Webanwendungen.

Abbildung 3: Request-Response-Zyklus in Ajax-Anwendungen



Wie in Abbildung 3 zu sehen, werden bei der Verwendung von Ajax im Vergleich zu klassischen Webanwendungen (vgl. Abbildung 1 auf Seite 10) mehrere kleinere Requests an den Server gesendet. Es ist weiterhin möglich, dass die verschiedenen Request-Response-Zyklen sich überlappen. Dies ist besonders vorteilhaft, wenn ein Request abgeschickt wird, dessen serverseitige Verarbeitung längere Zeit in Anspruch nimmt, da der Benutzer in diesem Fall trotzdem weiterarbeiten kann.

3.2 Technologien von Ajax-Anwendungen

Ajax-Anwendungen bestehen im Wesentlichen aus den folgenden Schichten:

- Präsentation: Die Darstellung der Anwendung erfolgt im Browser des Benutzers und wird mittels (X)HTML strukturiert und CSS formatiert (siehe 2.1.3).
- Datenaustausch: Nach einem asynchronen Request wird keine neue Seite angezeigt. Stattdessen werden aktualisierte Daten an den Client gesendet. Dazu wird eine strukturierte Sprache benötigt, die clientseitig verarbeitet werden kann, z.B. XML oder Javascript Object Notation (siehe 3.2.3).
- Aktualisierung: Zur Aktualisierung der angezeigten Seite wird mittels Javascript auf den HTML-Baum der Anwendung zugegriffen (siehe Document Object Modell 3.2.1).
- Kommunikation: Die Browser stellen das XMLHttpRequestObjekt (siehe 3.2.2) bereit um asynchrone Requests abzuschicken und zu verarbeiten.

Javascript ist das Bindeglied zwischen diesen Schichten und sorgt für deren clientseitige Zusammenarbeit. In den folgenden Abschnitten werden das Document Object Model sowie das Absenden asychroner Requests mittels des XMLHttpRequestObjects näher beleuchtet.

3.2.1 Document Object Model

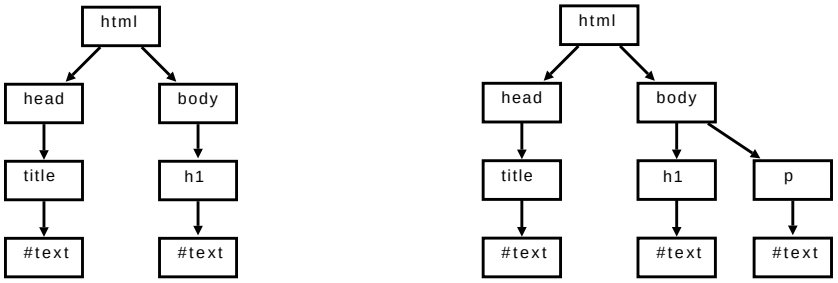
Das Document Object Model (DOM) ist eine plattform- und sprachunabhängige Programmierschnittstelle, um dynamisch auf Inhalt, Struktur und Layout von HTML-Dokumenten zuzugreifen und diese zu ändern. Durch das W3C ist auch das DOM standardisiert [BHH+04], so das sich die Verwendung in den einzelnen Browsern gleicht.

Aufbauend auf einem hierarchischen Baum, dessen Knoten die verschiedenen Elemente eines Dokuments repräsentieren, gibt es zwischen den einzelnen Knoten verschiedene Beziehungen: Child, Parent, Sibling. Analog dazu heißen die Operationen zur Navigation zwischen den einzelnen Knoten auch nextSibling, firstChild usw. Mittlels dieser Operationen sind ausgehend vom Wurzelknoten alle Knoten eines Dokuments erreichbar.

Listing 1: HTML-Dokument

```
1 <html>
2   <head>
3     <title>HTML-Dokument</title>
4   </head>
5   <body>
6     <h1>Ein kurzes Beispiel</h1>
7   </body>
8 </html>
```

Abbildung 4: Darstellung der Beispielseite als Baumstruktur vor und nach dem Einfügen neuer Elemente



In Listing 1 ist ein mit HTML ausgezeichnetes Dokument dargestellt, welches die Tags `html`, `head`, `title`, `body` sowie `h1` enthält, die ineinander verschachtelt sind. Dieses Dokument wird durch das DOM in den in Abbildung 4 (linker Teil) dargestellten Baum überführt. Durch die hierarchische Struktur des DOM unterteilen sich die Elementknoten in Parent-, Child- und Sibling-Beziehungen. So steht unter anderem `title` in Child-Beziehung zu `head`, welches wiederum in Child-Beziehung zu `html` steht.

Mit den Methoden des DOM lassen sich die einzelnen Elemente nun dynamisch verändern, neue Elemente einfügen sowie vorhandene löschen. Dies ist in Listing 2 verdeutlicht. In diesem Beispiel wird das Dokument aus Listing 1 bearbeitet und dynamisch ein neues Element eingefügt. Der resultierende DOM-Baum ist im rechten Teil der Abbildung 4 dargestellt. Für einen tiefergehenden Einblick in die Methoden und Attribute des DOM sei auf [Gam06] sowie auf die oben genannte Spezifikation verwiesen.

Listing 2: DOM-Methoden zur Bearbeitung eines Dokuments

```
1 function doDOM() {
2     var element = document.createElement("p");
3     element.appendChild
4         (document.createTextNode("Hallo Welt"));
5     document.childNodes[0].childNodes[1].
6         appendChild(element);
7 }
```

Da die Implementierungen des DOM in den verschiedenen Browsern mittlerweile relativ einheitlich sind und sich am Standard, ist es möglich, die Inhalte einer Website dynamisch zu ändern ohne browserspezifischen Code schreiben zu müssen. Dies ist eine der Voraussetzungen um Ajax-fähige Anwendungen zu erstellen. Alle Änderungen in der Benutzeroberfläche sind im Grunde nur Aufrufe der DOM-Implementierung des Clients. Dadurch muss der Client keine kompletten HTML-Seiten mehr erhalten. Stattdessen kann er selber dafür sorgen, die Daten, die er erhält, entsprechend darzustellen.

Damit ist es nun möglich, datenzentrierte Anwendungen zu entwickeln, die weitaus weniger Overhead produzieren als solche, bei denen komplette HTML-Dokumente als Response geschickt werden. Wie es mittels Ajax möglich ist dynamisch Daten nachzuladen, die dann in einer Änderung des Aussehens der Anwendung resultieren, soll im nächsten Abschnitt beschrieben werden.

3.2.2 XMLHttpRequest

Die Grundlage der Kommunikation einer jeden Ajax-Anwendung ist das Einleiten asynchroner HTTP-Request mittels des XMLHttpRequest-Objekts, welches von den jeweiligen Browsern bereitgestellt wird. Dieses kann von verschiedenen Skriptsprachen (im Browser in der Regel Javascript) verwendet werden, um Daten vom Client zum Server zu übertragen und die Antwort des Servers auszuwerten. Dazu wird eine neue HTTP-Verbindung aufgebaut, die unabhängig von der aktuellen ist, wodurch asynchrones Arbeiten ermöglicht wird. Es ist auch möglich, mehrere Requests gleichzeitig abzuschicken und dadurch mehrere parallele Verbindungen zu öffnen[8]. Auch auf dem Server werden diese dann unabhängig voneinander verarbeit.

Durch die Verwendung der Kommunikation über das XMLHttpRequest-Objekt wird es ermöglicht die Synchronität des HTTP zu verdecken. In klassischen Webanwendungen gibt der Benutzer Daten in ein Formular ein, sendet diese ab und wartet auf das Ergebnis. Die Anwendung ist während dieser Zeit nicht benutzbar. Durch asynchrone Verbindungen kann der Benutzer während der Verarbeitung der Daten auf dem Server weiterarbeiten. Außerdem verringert sich die zu übertragene Datenmenge, da nur tatsächliche Nutzdaten übertragen werden müssen.

Das XMLHttpRequest ist eine Schnittstelle einer Scriptsprache zur Implementierung von HTTP-Funktionalität, wie zum Beispiel dem Absenden eines Formulars oder dem Laden von Daten von einem Server. Der Name XMLHttpRequest ist dabei möglicherweise irreführend, da das Objekt nicht auf die drei Namensbestandteile beschränkt ist. So können Daten in jedem beliebigen Textformat übertragen und Requests über HTTP und HTTPS gestellt werden. Weiterhin ist das XMLHttpRequest-Objekt nicht auf das Absenden von Requests beschränkt, sondern kann auch HTTP-Responses empfangen und verarbeiten.

Das XMLHttpRequest-Objekt wurde in der Version 5 des Microsoft Internet Explorer zum ersten Mal bereitgestellt. Mittlerweile ist es auch in anderen Browsern vorhanden und wird von einer großen Anzahl populärer[9] Anwendungen verwendet. Da die verschiedenen Implementierungen teilweise nicht kompatibel zueinander sind, gibt es einen Standardisierungsvorschlag des W3C [Kes08].

Da jede Ajax-Anwendung auf der Benutzung des XMLHttpRequest-Objektes basiert, soll dessen Verwendung im Folgenden genauer erläutert werden. Durch tiefergehende Kenntnisse des Objektes ist es möglich, leistungsfähigere und robustere Anwendungen zu entwickeln.

[8] Hierbei ist allerdings zu beachten, dass in den meisten Browsern maximal zwei gleichzeitige Verbindungen zu einem Server aufgebaut werden können.

[9] Unter anderem GMail (mail.google.com) sowie Google Maps (maps.google.com).

Tabelle 1: Methoden des XMLHttpRequest-Objektes

Methode	Beschreibung
abort()	Beendet den aktuell laufenden Request.
getAllResponseHeaders()	Gibt alle HTTP-Header zurück.
getResponseHeader(header)	Gibt einen bestimmten HTTP-Header zurück.
open(method, url, [async, user, password])	Bereitet eine HTTP-Anfrage vor.
send(body)	Sendet den vorbereiteten Request an den Server.
setRequestHeader(header, value)	Setzt optionale Header-Werte.

3.2.2.1 Methoden und Attribute des XMLHttpRequest-Objektes

Zur Arbeit mit dem XMLHttpRequest-Objekt, welches in Ajax-Anwendungen verwendet wird, muss dieses zunächst erzeugt werden. Da in den verschiedenen Browsern dazu unterschiedliche Objekte notwendig sind, hat sich in der Praxis der in Listing 3 gezeigte, browserunabhängige Aufruf bewährt. Damit lässt sich ein neues XMLHttpRequest-Objekt erzeugen, das für die Implementierung der Ajax-Funktionalität verwendet werden kann. Tabelle 3.2.2.1 bietet einen Überblick über die Methoden des Objektes, die in allen Browsern gleich sind.

Listing 3: Instanziierung des XMLHttpRequest-Objektes

```
1  function getXMLHttpRequest() {
2    var request = null;
3
4    if (typeof XMLHttpRequest != "undefined") {
5      // Mozilla, Opera, Safarie, IE7
6      request = new XMLHttpRequest();
7    }
8    if (!request) {
9      // kleiner gleich IE6
10     try {
11       request = new ActiveXObject("Msxml2.XMLHTTP");
12     } catch (e) {
13       try {
14         request = new ActiveXObject("Microsoft.XMLHTTP");
15       } catch (e) {
16         request = null;
17       }
18     }
19   }
20   return request;
21 }
```

Soll nun ein Request an den Server gesendet werden, wird dies mit dem Ausführen der Methoden `open` und `send` erreicht. Dabei wird in der Methode `open` der Typ der Übertragung festgelegt.Der Standardfall - `true` - steht für eine asynchrone Übetragung; soll eine synchrone Kommunikation realisiert werden, muss als Wert `false` angegeben werden.

Für Ajax-Anwendungen verwendet man in der Regel eine asynchrone Kommunikation. Damit ist es möglich, den Status einer Anfrage zu überwachen und auf eventuel-

Tabelle 2: Attribute des XMLHttpRequest-Objektes

Eigenschaft	Beschreibung
readyState	Aktueller Status des Objekts • 0: nicht initialisiert • 1: Request-Objekt bereit, aber noch nicht abgeschickt • 2: Anfrage wurde per send() gesendet, der Server hat noch nicht geantwortet • 3: Daten werden empfangen • 4: Request vollständig ausgeführt, alle Daten empfangen
onreadystatechange	Referenz auf eine Callback-Funktion. Diese wird aufgerufen, wenn sich der Status des Objektes ändert.
status	HTTP-Statuscode der Antwort
statusText	HHTP-Statustext der Antwort
responseText	Antwort als Text
responseXML	Antwort im XML-Format

le Fehler zu reagieren. Im Gegensatz dazu wird bei der synchronen Kommunikation das Ergebnis einer Anfrage erst nach der kompletten Übertragung verfügbar. Bis dieser Fall eintritt wird die weitere Ausführung des Skriptes blockiert und die Anwendung kann nicht verwendet werden.

Neben den genannten Methoden bietet das XMLHttpRequest-Objekt auch Eigenschaften an, mit denen sich der Status einer Anfrage ermitteln lässt. Diese sind in Tabelle 3.2.2.1 zusammengefasst. Der wichtigste Status hat dabei die Nummer 4 Er signalisiert, dass ein Ajax-Request vollständig durchgeführt wurde.

3.2.2.2 Verwendung des XMLHttpRequest-Objektes

Nachdem die grundlegenden Methoden und Attribute des XMLHttpRequest-Objektes beschrieben wurden, soll im Folgenden beispielhaft ein asynchroner Aufruf dargestellt werden, der die Grundlage aller Ajax-Anwendungen ist. Dazu wird die bereits definierten Methode `getXMLHttpRequest()` aus Listing 3 verwendet.

Nach der Instanziierung des Objektes muss eine Callback-Funktion festgelegt werden, die beim Ändern des Status' aufgerufen werden soll. In dieser Callback-Funktion findet die Verarbeitung der Antwort des Servers statt. In Listing 4 wird diese Funktionalität implementiert. Dabei wird eine Datei vom Server angefordert

und deren Inhalt in einer Popupbox im Browser des Anwenders ausgegeben.

Listing 4: Ajax-Aufruf

```
1  request = getXMLHttpRequest();
2  request.onreadystatechange = callbackFunction;
3  request.open("GET", "/datei.txt");
4  request.send();
5
6  function callbackFunction() {
7    if ((request.readyState == 4) &&
8        (request.status == 200)) {
9      alert(request.responseText);
10   }
11 }
```

3.2.2.3 XMLHttpRequest in der Praxis

Die Kommunikation über das XMLHttpRequest-Objekt ist wie bereits erläutert die Grundlage einer jeden Ajax-basierenden Anwendung. Allerdings werden Entwickler von Webanwendungen eher selten mit der direkten Erzeugung und Verwendung des XMLHttpRequest-Objektes betreut sein, da es mittlerweile Frameworks gibt, die diese Aufgabe kapseln. Dies hat den Vorteil, dass man sich nicht um die genauen Unterschiede in der Implementierung der einzelnen Browser kümmern muss, wodurch schneller Ergebnisse erzielt werden. Dessen ungeachtet ist es gerade bei der Fehlersuche von Vorteil, die Eigenheiten des Objektes zu kennen.

Der Vollständigkeit halber sei hier noch erwähnt, dass es neben dem XMLHttpRequest-Objekt mit dem Document Object Model Level 3 Load and Save (DOM L&S) eine weitere Möglichkeit gibt, eine asynchrone Kommunikation zu ermöglichen. Mit dieser Schnittstelle können XML-Daten dynamisch in ein bestehendes DOM-Dokument eingelesen und das Dokument dann an den Server gesendet werden.

Da das DOM L&S durch das W3C standardisiert [SH04] ist, wäre es dem XMLHttpRequest-Objekt eigentlich vorzuziehen. Allerdings ist es in den aktuellen Browserversionen nicht oder nur sehr schlecht implementiert[10]. Außerdem unterstützt es nur XML-Daten und kann diese laut Spezifikation nur per HTTP-PUT übertragen. Daher wird im weiteren Verlauf der Arbeit auch nur das XMLHttpRequest-Objekt verwendet

3.2.3 Javascript Object Notation

Nachdem bereits gezeigt wurde, wie asynchrone Verbindungen aufgebaut und neue Daten dynamisch in ein HTML-Dokument eingefügt werden, wird im Folgenden umrissen, wie Daten formatiert werden, damit sie möglichst schnell mit den bereits bekannten Technologien erstellt und gelesen werden können. Zum Austauschen

[10] Einzig Opera unterstützt DOM L&S komplett, Mozilla unterstützt nur einen veralteten Teil und der Internet Explorer gar nicht.

strukturierter Daten zwischen dem Client und dem Server, benötigt man ein Datenaustauschformat, welches beide verarbeiten können. Neben dem bereits angesprochenen XML bietet sich zur Verwendung in Javascript auch die Javascript Object Notation (JSON) an. JSON basiert auf einer Untermenge von Javascript und ist ein sprachunabhängiges Format. Sie kann daher in allen gängigen Programmiersprachen generiert und geparst werden [Cro06].

Im Gegensatz zu XML werden Daten mit JSON kompakter dargestellt und lassen sich damit schneller übetragen. Weiterhin ist gerade bei der Integration mit Javascript die direkte Auwertung mittels der `eval()`-Funktion hervorzuheben, wodurch sehr schnell auf die Daten zugegriffen werden kann.

Listing 5: JSON-Darstellung eines Objekts

```
1 {
2     "Titel":    "Bohnenkaffe",
3     "Zutaten":  [
4         Bohnen,
5         Kaffee
6     ],
7     "machen":   function() {
8         alert("Ich mache einen " + this.Titel);
9     }
```

Grundbestandteile der JSON-Syntax sind Name-Wert-Paare, die in Programmiersprachen Objekte darstellen sowie eine geordnete Liste von Werten, die Arrays darstellen. Durch die Möglichkeit der Schachtelung dieser Typen lassen sich komplexe Datenstrukturen erzeugen. Listing 5 zeigt die Darstellung eines Objektes in JSON. Dort wird außerdem die Möglichkeit, eine Funktion zu integrieren verwendet. Auf diese Funktion kann dann mittels `Kaffe.machen()` zugegriffen werden.

3.3 Vorteile und Probleme von Ajax-Anwendungen

Nachdem der grundlegende Aufbau die Arbeitsweise von Ajax-Anwendungen bekannt ist, können davon ausgehend die Vorteile, die sich aus der Verwendung von Ajax ergeben, abgeleitet werden.

Der bei weitem größte Vorteil ist die Ermöglichung neuer, interaktiver Benutzeroberflächen. Mittels Ajax kann viel besser auf Aktionen des Benutzers reagiert werden, ohne dass neue Seiten geladen werden müssen. Wie jede Webanwendung muss auch eine Ajax-Anwendung nicht auf dem Clientrechner installiert werden, so dass dieser immer die aktuellste Version verwendet. Im Gegensatz zu anderen Lösungen wie z.B. Adobe Flash müssen Benutzer keine gesonderten Plugins installieren. Theoretisch reicht ein installierter Browser aus, um alle zur Verfügung stehenden Anwendungen zu nutzen. Durch die Verlagerung der Präsentationslogik auf den Client wird dessen Rechenleistung ausgenutzt, so dass die Serverseite entlastet wird.

Ajax bietet die Möglichkeit datenzentrierte Kommunikation zu implementieren. Dadurch werden nur kleinere inkrementelle Updates des Inhalts vorgenommen, was zu einer Verringerung des Datenverkehrs zwischen Client und Server führt. Gerade bei Endgeräten mit langsamer Internetverbindung wie zum Beispiel Handys lässt sich

die Arbeitsgeschwindigkeit damit erhöhen. In [Roo06] wird der Datenverkehr einer normalen mit dem einer Ajax-Anwendung verglichen.

Auch wenn Ajax-Anwendungen neue Interaktionsmöglichkeiten bereitstellen, verursacht die Verwendung der zugrunde liegenden Technologien neue Probleme, die gelöst werden müssen. Trotz des noch relativ jungen Alters der Ajax-Technologie gibt es für die meisten dieser Probleme bereits ausgereifte Lösungen.

Da Ajax-Anwendungen auf Javascript basieren, benötigen sie zwingend, dass dieses in den jeweiligen Browsern aktiviert ist. Beim Internet Explorer kommt (bis zur Version 7) aufgrund der nicht nativ vorhandenen Unterstützung des XMLHttpRequest-Objektes außerdem ActiveX als Voraussetzung hinzu. Dadurch kann es bei restriktiven Sicherheitsrichtlinien in Unternehmen zu Problemen bei der Verwendung Ajax-basierender Anwendungen kommen. Auf diese Probleme wird im Folgenden nicht eingegangen, da sie eher organisatorischer als technischer Natur sind und auch nicht programmatisch umgangen werden können.

3.3.1 Integration in vorhandenes Anwendungs- und Entwicklungsmodell

Die Integration von Ajax in eine bestehende Anwendung erfordert sowohl client- als auch serverseitig Änderungen am Anwendungsmodell. Ohne Ajax-Integration müssen Webseiten komplett auf dem Server vorgerendert werden und werden dann nur noch an den Client ausgeliefert, welcher die Dokumente anzeigt. Mit Ajax kann das Rendern einer Seite vollständig clientseitig erfolgen. Daher muss vor der eigentlichen Integration geklärt werden, welche Aufgaben auf dem Server verbleiben und welche Daten zwischen dem Client und dem Server ausgetauscht werden sollen.

Die Implementierung des MVC-Patterns kann durch die Verwendung von Ajax verbessert werden. Durch datenzentrierte Anwendungen ist es möglich die komplette View und auch Teile des Controllers auf den Client auszulagern, so dass der Server nur das Anwendungsmodell bereitstellt. Weitere Strategien zur Umsetzung des MVC-Patterns in Ajax-Anwendungen finden sich in [MCLPSF07].

Neben Änderungen im Anwendungsmodell ergeben sich zwangsläufig auch Änderungen in der Entwicklung einer Anwendung. Da Ajax auf Javascript basiert, können browserspezifische Probleme auftreten, die während der Entwicklung beachtet werden müssen. Durch die noch realtiv geringe Anzahl an Entwicklungswerkzeugen kann es zu einem Bruch zwischen der Implementierung von client- und serverseitigen Komponenten kommen.

3.3.2 Probleme mit der Browser-History

Ein großer Nachteil von Ajax-Anwendungen ist die fehlende Unterstützung der Browserhistory sowie der Navigation mittels der Vor- und Zurpck-Buttons der Browser. Da sich durch das Ausführen von Javascript-Ereignissen, bei denen keine neue Seite angezigt wird, die URL des Browsers nicht ändert, haben zwei verschiedene

Zustände die gleiche URL. Dadurch ist es nicht mehr möglich, mittels eines Lesezeichens einen bestimmten Anwendungszustand festzuhalten. Außerdem wird bei der Benutzung der Vor- und Zurück-Buttons die letzte Seite neu geladen und nicht der letzte Zustand. Dies kann vor allem bei der Eingabe großer Datenmengen (zum Beispiel in einem Rich Text Editor) zu Brüchen im Workflow des Anwenders führen.

Eine Möglichkeit, den Status einer Ajax-Anwendung zu speichern und wiederherzustellen, ist die Verwendung einer Sprungmarke. Normalerweise wird über das Gattersymbol # ein bestimmtes Ziel im Dokument angesprungen; exisitert dieses Ziel nicht, bleibt die Anzeige unverändert. Dadurch können Ajax-Anwendungen diese Sprungmarke verändern, ohne dass der Browser darauf reagiert. Nach jeder Aktion, die den Zustand der Anwendung ändert, wird eine Sprungmarke an die URL der aktuellen Seite angehängt. Wird dann diese Seite wieder per Lesezeichen aufgerufen, kann die Anwendung den entsprechenden Zustand wiederherstellen.

Um auch die Vor- und Zurück-Buttons funktionsfähig zu halten, sind weitere Anpassungen an der Anwendung nötig. In Browsern, die auf der Gecko-Engine (z.B. Mozilla) basieren, muss geprüft werden, ob sich die URL geändert hat und gegebenenfalls der Inhalt der Seite aktualisiert werden. Im Internet Explorer hingegen muss ein versteckter iframe eingebunden werden, der bei jeder Zustandsänderung der Anwendung aktualisiert und damit in der History des Browsers abgelegt wird.

Sowohl für die Probleme mit Lesezeichen als auch für diese mit den Vor- und Zurück-Buttons gibt es mittlerweile viele Lösungen, die in eine vorhandene Ajax-Anwendung eingebunden werden können. Eine ausgereifte Variante ist das Really Simple History Framework[11]. Weitere Informationen zur Umsetzung der History-Funktion finden sich in [CBS08].

3.3.3 Barrierearmut

Ein großes Problem aller auf Javascript basierenden Webanwendungen ist die Implementierung barrierearmer Benutzeroberflächen. Damit Webinhalte einem möglichst großen Benutzerkreis zugänglich gemacht werden können, sollten sie sich an den gängigen Richtlinien[12] zur Erstellung barrierearmer Webseiten orientieren. Dies gilt insbesondere für Internetauftritte der Behörden auf Bundesebene, die seit 31.12.2005 barrierefrei sein müssen[13].

Die Dynamisierung von Webseiten mittels Javascript kann dazu führen, dass diese barrierearme Gestaltungsrichtlinien verletzt werden. Dieses Problem wird hier nicht weiter im Detail betrachtet. Allerdings müssen entsprechende Lösungen gefunden werden, falls eine um Ajax erweiterte Version des Dialog Control Frameworks in öffentlichen Einrichtungen eingesetzt werden soll.

[11] Siehe http://codinginparadise.org/weblog/2005/09/ajax-history-libraries.html.

[12] Die Web Accessibility Initiative des W3C hat 1999 die Web Content Accessibility Guidlines veröffentlicht: http://www.w3.org/TR/WCAG10-HTML-TECHS/

[13] Maßgaben dazu sind in der ensprechenden Rechtsverordnung Barrierefreie Informationstechnik-Verordnung (BITV) geregelt.

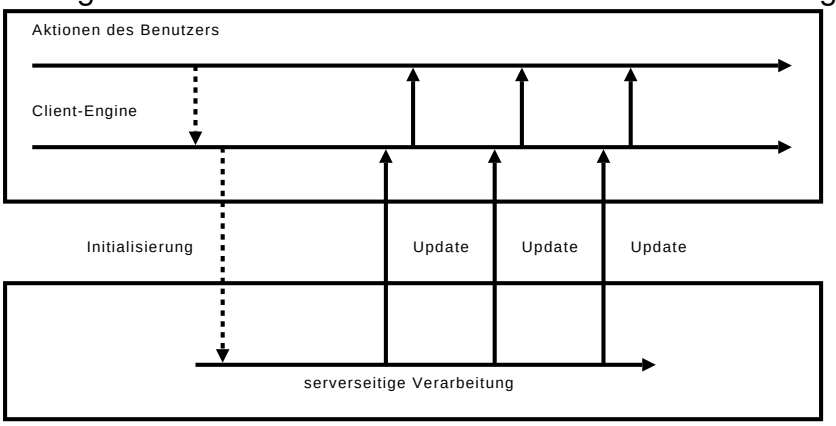
3.3.4 Serverseitige Aktualisierungen

Da auch bei der Verwendung Ajax-basierter Nutzerinteraktionen das HTTP verwendet wird muss die Kommunikation aufgrund des vorgegebenen Request-Response-Zyklus' vom Client ausgehen. Es ist daher nicht möglich in Echtzeit Daten zu aktualiseren, wenn diese sich serverseitig ändern. Zur Simulation echtzeitbasierender Kommunikation kann ein Polling-Mechanismus verwendet werden, bei dem der Client in vorgegebenen Intervallen Aktualisierungen vom Server anfragt.

Besser wäre es, wenn der Server direkt Daten an den Client senden könnte, womit keine Pull- sondern eine Push-basierende Kommunikation erreicht wird. In [BMD07] werden verschiedene Alternativen vorgestellt, wie die Kommunikationsmöglichkeiten von HTTP erweitert werden können; als Vertreter Push-basierender Kommunikation wird die Technik Reverse Ajax / Comet vorgestellt.

Dabei antwortet der Server auf eine Anfrage des Clients sehr langsam, so dass die Verbindung zwischen beiden Punkten offen bleibt. Dies wird ermöglicht, indem der Timeout, nach dem eine Verbindung getrennt wird auf einen sehr hohen Wert gesetzt wird; diese dauerhaften Verbindungen werden auch als long-lived HTTP connection bezeichet. Abbildung 5 stellt die Architektur einer Comet-basierenden Anwendung genauer dar. Der Server sendet Daten an den Client, welche dieser mittels einer internen Comet-Schnittstelle verarbeitet und die vorhandenen Daten aktualisiert.

Abbildung 5: Architektur einer Comet-basierenden Anwendungen



Durch den Einsatz von Comet kann der Server entscheiden wann neue Daten gesendet werden, was zu einer Verringerung der verwendeten Bandbreite führen kann. Weiterhin verringert sich der Overhead, welcher durch das Auf- und Abbauen einzelner HTTP-Verbindungen entsteht. Ein weiterer Vorteil des Comet-Ansatzes ist, dass dieser ebenso wie Ajax keine clientseitigen Plugins benötigt, sondern auf normalen HTTP-Requests aufsetzt.

Obwohl Daten an den Client gesendet werden können, muss auf diesem eine Methodik implementiert werden, mit der die neuen Daten abgefragt werden, da Browser keine automatische Aktualisierung unterstützen. Es muss daher ein clientseitiger Puffer angelegt werden, der regelmäßig geprüft wird. Die langlebige HTTP-Verbindung, auf der Comet beruht, kann außerdem dazu führen, dass der Server

weniger Anfragen gleichzeitig bearbeiten kann. Da die Verbindung nur zum Empfang genutzt werden kann, muss eine neue Verbindung aufgebaut werden, wenn der Client Daten an den Server sendet.

3.4 Zusammenfassung

In diesem Kapitel wurde eine Möglichkeit aufgezeigt, wie Limitierungen klassischer Webanwendungen, die sich durch die Verwendung des HTTP ergeben, umgangen werden können. Mittels Ajax lassen sich durch die Verdeckung der Synchronität Webanwendungen mit interaktiveren und reaktionsschnelleren Benutzeroberflächen entwickeln. Es ist daher möglich, komplexere GUIs zu entwickeln, die fast verzögerungsfrei verwendet werden können.

Auch die Beschränkungen und Probleme bei der Verwendung von Ajax wurden aufgezeigt. Da es heutzutage für einen Großteil der Probleme bereits Lösungen gibt, soll Ajax im Rahmen des Dialog Control Frameworks eingesetzt werden, um eine reichhaltigere Dialoggestaltung zu ermöglichen. Dazu ist im nächsten Kapitel zu prüfen, welche Anpassungen am Framework vorgenommen werden müssen und wie die Dialogflussnotation zu erweitern ist.

4 Integration von Ajax in das Dialog Control Framework

In den vorangegangen Kapiteln wurden Webanwendungen sowie deren Defizite in der Nutzerinteraktion vorgestellt. Durch die Verwendung Ajax-basierter Kommunikation lassen sich einige dieser Defizite umgehen. In diesem Kapitel soll gezeigt werden, in welcher Form das DCF erweitert werden muss um Ajax zu integrieren. Dazu muss zuerst die Spezifikation der neuen Interaktionen mittels der DFN / DFSL ermöglicht werden. Diese enthält momentan nur Sprachelemente, um synchrone Dialogflüsse zwischen einzelnen Dialogelementen darzustellen. Daran anschließend muss das DCF selber so angepasst werden, dass es asynchrone Requests verarbeiten kann und bei Bedarf keine neuen Seiten anzeigt.

Bei der Ajax-Kommunikation werden keine kompletten HTML-Dokumente vom Server an den Client gesendet, sondern nur Daten bereitgestellt; dies muss sich auch in der DFN widerspiegeln, so dass Anwender erkennen, welcher Kommunikationsweg verwendet wird. Darauf aufbauend muss ein entsprechendes Gegenstück in die DFSL implementiert werden. Diese wird aus der DFN generiert und vom DCF als Spezifikation der Dialogflusssteuerung verwendet. Die eigentliche Implementierung der DCF-Komponente wird im Kapitel 5 dargestellt.

Bevor das DCF um Ajax erweitert wird, soll es in das JSF Framework integriert werden. JSF hat sich zur einfacheren Entwicklung komplexer J2EE-Anwendungen etabliert. Es dient dem Erstellen graphischer Benutzeroberflächen aus standardisierten Komponenten. In diesem Kapitel wird gezeigt, welche strukturellen Änderungen am DCF und am JSF Framework zur Kombination der beiden vorzunehmen sind. Daran schließt sich die Vorstellung von Möglichkeiten zur Integration Ajax-basierter Nutzerinteraktion in das JSF Framework an.

Im letzten Kapitel wurden bereits einige Probleme Ajax-basierter Anwendungen angesprochen. Vor allem clientseitig ergeben sich durch die Verwendung von Javascript neue Anforderungen an Entwickler von Webanwendungen. Damit beim Gebrauch der Ajax-Komponente möglichst wenig clientseitiger Javascript-Code geschrieben werden muss, wird die Verwendung eines Frameworks geprüft. Dies ermöglicht die Entwicklung robusterer Anwendungen, da von den eigentlichen technischen Grundlagen abstrahiert wird. Zur Verdeutlichung der Auswahl werden verschiedene Frameworks vorgestellt und miteinander verglichen.

4.1 Integrationsschritte

Das Dialog Control Framework übernimmt in einer auf dem MVC-Pattern basierenden Webanwendung die Controller-Komponente der Applikation. Jeder Request, den ein Client absendet, enthält einen Parameter `Event`, der vom DCF ausgewertet wird. Der `DialogController` ermittelt dann anhand des aktuellen Anwendungsstatus', welche Dialogflüsse aufgerufen werden müssen.

Abbildung 6: Elemente einer Dialogflussbeschreibung

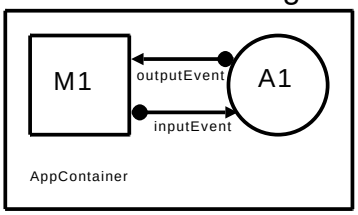


Abbildung 6 sowie Listing 6 zeigen einen beispielhaften Ausschnitt einer Dialogflussbeschreibung: Java Server Pages Masken bilden dabei die Präsentationsschicht, die Actions dagegen implementieren die eigentliche Geschäftslogik.

Als zentrale Klasse fungiert der `DialogController`, welcher immer dann aufgerufen wird, wenn der Client einen Request sendet. Ausgehend davon wird durch die Actions die Geschäftslogik der Anwendung ausgeführt. Ein vollständiger Dialogschritt beginnt und endet immer an einer Maske und kann beliebig viele andere Dialogelemente in sich enthalten.

Listing 6: Implementierungen Maske M1 sowie Action A1

```
1  Maske M1:
2  <html>
3  <head><title>M1</title></head>
4  <body>
5  <a href="index?event=inputEvent">rufe inputEvent auf</a>
6  </body>
7  </html>
8
9  Action A1:
10 class A1 extends DCFActionImpl
11 {
12   public String performOperation(DialogEvent event)
13   {
14     return "outputEvent";
15   }
16 }
```

Da am Ende eines jeden Dialogschrittes eine neue Maske geladen und damit ein vollständiges HTML-Dokument an den Client übertragen wird, muss eine Möglichkeit gefunden werden, diesen Punkt des DCF-Lebenszyklus so zu ändern, damit bei der Verarbeitung eines Ajax-Requests nur die angeforderten Daten zur Verfügung gestellt werden.

4.1.1 Integration des DCF mit JSF

JSF ist ein Framework zur komponentenbasierten Erstellung graphischer Benutzeroberflächen. Es liefert erprobte und funktionierende Konzepte zur Umsetzung einer GUI in Webanwendungen [Bos04]:

- MVC-Pattern: JSF unterteilt eine Anwendung in die Teile Model (Managed Beans), View (die UI-Komponenten) sowie Controller (das FacesServlet). Dadurch werden Anwender bei der Entwicklung strukturierter Webanwendungen unterstützt.

- Navigationskonzept: Durch Spezifizierung der Navigationsmöglichkeiten einer JSF-Anwendung innerhalb der `faces-config` lässt sich die Navigationslogik auslagern. Dadurch entfällt die Angabe direkter Zielseiten in den JSF-Seiten, so dass auf die Navigation auf einem höheren Abstraktionsniveau erfolgen kann.
- Bean Management: Mittels der sogenannten Expression Language des JSF-Frameworks lassen sich Managed Beans direkt von einer Webseite aus ansprechen, wodurch diese automatisch die aktuellen Werte der jeweiligen Bean besitzen.
- Eventhandling: JSF erlaubt das Hinzufügen von Listenern, die auf bestimmte Events einer Komponente reagieren. Dadurch lässt sich eine dem GUI Framework Swing ähnliche Event-Verarbeitung implementieren.
- Datenvalidierung und -konvertierung: Daten, die in Formularen auf Webseiten eingegeben werden, werden von JSF automatisch in ihr korrektes Format konvertiert. Dadurch können die eingegebenen Daten leicht automatisch validiert werden.

Da das DCF als Framework zur Steuerung von Dialogflüssen in Webanwendungen konzipiert ist, setzt es am Navigationskonzept des JSF Frameworks an. Es ist möglich, die verschiedenen JSF-Komponenten zu ersetzen, so dass die Grundidee zur Integration ist, diejenige Komponenten zu ersetzen, die für die Navigation zuständig ist. Das DCF übernimmt dann die Rolle dieses sogenannten NavigationHandlers.

Sowohl das FSF Framework als auch das DCF basieren auf der Grundlage benannter Transitionen zwischen den Navigationselementen; daher kann das DCF den Standard-JSF-NavigationHandler ersetzen. Die Action-Namen des JSF werden dabei zu den Event-Namen des DCF. Durch die größere Mächtigkeit der DFN sind weitaus komplexere Dialogflüsse möglich als JSF bereitstellt.

4.1.2 Integration von Ajax-Requests in den JSF-Lifecycle

Bei jedem Request an eine JSF-basierende Anwendung wird der JSF-Lifecycle durchlaufen. Dessen Hauptaufgaben sind die Bearbeitung des Komponentenbaums einer JSP, die Validierung gesendeter Daten sowie die Aktualisierung des Geschäftsmodells durch Java Beans. Beim Durchlauf eines kompletten Lifecycles werden also umfangreiche Berechnungen auf dem Server ausgelöst. Da bei der Verwendung von Ajax mehr Requests als in klassischen Webanwendungen gesendet werden, führt dies zu einer erhöhten Belastung des Servers, die es zu vermeiden gilt. Wird der Ajax-Request allerdings nicht im Rahmen des JSF-Lifecycles behandelt, wird auch der Komponentenbaum nicht aktualisiert, wodurch es zu Inkonsistenzen zwischen dem server- und dem clientseitigen Anwendungszustand kommt.

JSF selber bietet in der aktuellen Version keine direkte Ajax-Integration an. Da die Architektur allerdings erweiterbar ist, ist es möglich, eigene Komponenten mit Ajax-Unterstützung zu entwickeln. Es muss also eine Möglichkeit gefunden werden, Ajax-

Requests in den Lifecycle zu integrieren und von solchen Requests, die eine komplett neue Seite anfordern, zu unterscheiden. Außerdem soll gezeigt werden, welche Möglichkeiten es gibt, den Response so anzupassen, dass keine unnötigen Daten gesendet werden.

4.1.2.1 Abänderung des JSF-Lifecycles

Damit der Lifecycle überhaupt abgeändert werden kann, muss dem Server bekannt gegeben werden, ob es sich um eine Ajax- oder eine normale Anfrage handelt. Dies kann durch die Erweiterung des HTTP-Request-Headers geschehen. Dadurch kann anhand der verschiedenen Request-Typen definiert werden, welche Aktionen ausgeführt werden sollen. JSF bietet mehrere Möglichkeiten, in den vorhandenen Lifecycle einzugreifen und diesen gegebenenfalls abzuändern.

Es ist - unabhängig von JSF - möglich, Ajax-Requests an ein eigenes Servlet weiterzuleiten, welches nicht in JSF integriert ist. Dadurch wird das zentrale `FacesServlet` nicht aufgerufen und der komplette Lifecycle umgangen, wodurch diese Methode sehr performant ist. Allerdings ist es nicht möglich, auf Validatoren und andere Elemente des JSF Frameworks zuzugreifen, so dass diese eigenständig implementiert werden müssen. Es ergeben sich außerdem Probleme, wenn es zu Änderungen im Anwendungsmodell kommt. Diese müssen dann sowohl im JSF-Teil der Anwendung als auch im Servlet für die Ajax-Requests integriert werden. Eine weitere Möglichkeit ist der Einsatz eines Servlet Filters[14], der vor jedem Request und nach jedem Response aufgerufen wird. Auch dieser leitet den Ajax-Request auf ein eigenes Servlet weiter, so dass zwar auf JSF-Komponenten zugegriffen werden kann, diese allerdings eigenhändig implementiert werden müssen. Der JSF-Lifecycle wird auch mit dieser Methode nicht angestoßen, wodurch sich die gleichen Probleme wie bei einem extern definierten Servlet ergeben.

Die beiden vorgestellten Möglichkeiten genügen den Anforderungen, sich innerhalb des JSF-Lifecycles zu bewegen nicht, daher muss direkt in diesen eingegriffen werden. Dazu bietet sich das Konzept der PhaseListener an; diese werden im Rahmen des Lifecycles vor und nach jeder Phase aufgerufen. Ein PhaseListener wird dazu für diejenige Phase registriert, in deren Rahmen er benachrichtigt werden soll. Innerhalb der PhaseListener ist der Zugriff auf den vollen JSF-Komponentenbaum möglich, wodurch auch serverseitige Validatoren angestoßen werden können sowie der Zugriff auf Managed Beans möglich ist.

4.1.2.2 Verarbeitung des Responses

Nachdem der Request serverseitig verarbeitet worden ist, muss der Response an den Client zurückgeschickt werden. Dazu gibt es mehrere Methoden, die unterschiedlich komplex sind. Die einfachste Methode, Ajax-Kommunikation zu integrieren, ist die Verwendung einer Technik namens Partial Page Rendering (PPR). Dabei

[14] Servlet Filter wurden in der Java Sevlet Spezifikation 2.3 eingeführt. Weitere Informationen zu Filtern finden sich unter http://java.sun.com/products/servlet/Filters.html

schickt der Server die komplette Response zurück und der Client sucht die zu aktualisierenden Daten aus dem Dokument. Es sind daher keinerlei serverseitige Anpassungen notwendig, so dass diese Methode leicht in bestehende Anwendungen integriert werden kann. Allerdings hat diese Methode den Nachteil, dass der Server unnötige Daten überträgt, die der Client nicht benötigt.

Ein anderer Ansatz beinhaltet die Verwendung des sogenannten Delta DOM Renderings (D^2R) [MD08], welches darauf basiert, Unterschiede zwischen zwei DOM-Strukturen zu ermitteln; diese werden als Delta bezeichnet. Die Berechnung der Differenz kann dabei entweder client- oder serverseitig erfolgen. Bei der Berechnung auf dem Client wird wie beim PPR die komplette Response übertragen. Wird die Differenz auf dem Server berechnet, speichert dieser die aktuell vorhandene Struktur des DOM zwischen und berechnet die Deltas. Dadurch müssen nur noch diese Deltas an den Client gesendet werden, so dass sich die Menge der zu versendenden Daten reduziert. Auch hier ist clientseitig Logik vorhanden, die dafür sogt, dass der DOM-Baum der aktuellen Seite mit den geänderten Daten aktualisiert wird.

4.1.3 Erweiterung der Dialog Flow Notation

Da wie oben bereits beschrieben bei der Ajax-Kommunikation keine Seiten neu geladen werden sollen, muss die DFN um eine neue Kante erweitert werden, die diesen Kommunikationsweg visualisiert. In der dieser Arbeit zugrundeliegenden Version des DCF sind bereits die folgenden Kanten vorhanden:

- Dialogflusskanten spezifizieren die eigentlichen Events zwischen zwei Dialogelementen und werden als gerichtete Kante vom Sender- zum Empfängerelement dargestellt.
- Datenflusskanten spezifizieren die Datenflüsse, welche vom jeweiligen Event ausgelöst werden. Sie werden ebenfalls als gerichtete Kante dargestellt, die im Falle synchroner Datenflüsse auf der Dialogflusskante liegt und im Falle asynchroner als gebogener, gestrichelter Pfeil die beiden Dialogelemente miteinander verknüpfen.

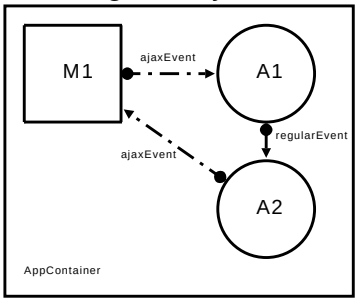
Damit sich Ajax-basierende Zustandsübergänge visuell von normalen Events unterscheiden, müssen diese gesondert dargestellt werden. Dazu gibt es die folgenden beiden Möglichkeiten

- Die Kanten für reguläre Events werden verwendet und Ajax-Events erhalten ein Präfix `ajax`.
- Es wird eine neue Kante eingeführt, die Ajax-Events repräsentiert.

Die erste Variante hat den Vorteil, dass keine neuen Elemente in die DFN aufgenommen werden müssen und sich deren Komplexität dadurch nicht erhöht. Demgegenüber stehen allerdings erhebliche Nachteile. So muss bei der Erstellung sowie Änderungen des Modells darauf geachtet werden, dass Ajax-Events korrekt bezeichnet werden. Außerdem gleicht sich die Darstellung beider Eventarten, so dass sie

erst nach eingehenderer Untersuchung der Spezifikation voneinander unterschieden werden können. Daher wird eine eigene Kante für Ajax-Events eingeführt, die sich an den Eigenschaften regulärer Event-Kanten orientiert, sich aber visuell von dieser abhebt.

Abbildung 7: Graphische Darstellung von Ajax-Events in der Dialog Flow Notation



Die Kante wird als gerader Strichpunktpfeil dargestellt, so dass eine Abgrenzung zu regulären Dialogevents möglich ist. Da in der DFN asynchrone Datenflüsse bereits als gestrichelte Pfeile dargestellt werden, wurde zur besseren Unterscheidung die Strichpunktnotation gewählt. Abbildung 7 zeigt diese Darstellungsform anhand von Ajax-Events zwischen Dialogelementen.

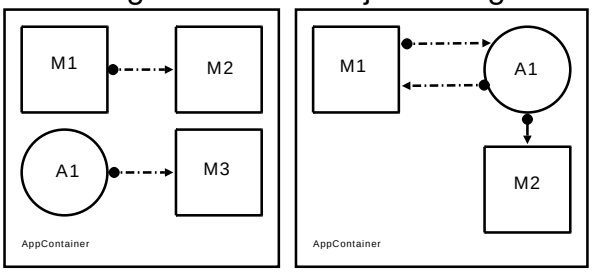
4.1.3.1 Regeln für Ajax-Events

Reguläre Events im DCF können zwischen zwei beliebigen Dialogelementen ausgetauscht werden. Im Gegensatz dazu sind bei der Verwendung von Ajax-Events folgende Regeln zu beachten:

- Ajax-Events werden als Reaktion auf eine durch den Benutzer initiierte asynchrone Browser-Anfrage ausgelöst und können daher nur von Masken ausgehen. Actions können keine Ajax-Events auslösen, da sie nicht auf Benutzereingaben reagieren können.
- Nach der Abarbeitung des Ajax-Events muss wieder die auslösende Maske angezeigt werden. Die Abarbeitungskette zwischen ausgehendem und eingehendem Ajax-Event darf also keine weitere Maske enthalten. Da nach Abarbeitung der Event-Kette ein Update des DOM-Baums erfolgt, macht es keinen Sinn, eine andere Maske anzuzeigen. Dies würde einem Seitenwechsel gleichkommen und daher die Asynchronität der Ajax-Kommunikation hinfällig machen.
- Dialogelemente innerhalb der Abarbeitungskette dürfen keine Events enthalten, die auf andere Masken zeigen. Eine Ausnahme dieser Regel ergibt sich bei der Verwendung von Error-Events. Diese unterbrechen den eigentlichen Dialogfluss und können daher auch innerhalb der Ausführung eines Ajax-Flusses aufgerufen werden. Falls diese eine Maske enthalten, so darf diese aufgerufen werden.

Abbildung 8 zeigt nicht erlaubte Dialogflüsse mit Ajax-Kommunikation. Links oben sendet eine Maske ein Ajax-Event zu einer anderen Maske; dadurch wird am Ende der Abarbeitung nicht die ursprüngliche Maske angezeigt. Darunter initiiert eine Action Ajax-Kommunikation, was aber dem Grundgedanken widerspricht, dass Ajax-Events durch eine Benutzeraktion ausgelöst werden. Auf der rechten Seite ist eine formal korrekte Ajax-Eventkette zu sehen; die Action `A1` verfügt allerding über einen Dialogfluss zu einer anderen Maske als der ursprünglichen, wodurch auch dieser Dialogfluss nicht erlaubt ist.

Abbildung 8: Verbotene Ajax-Dialogflüsse



Unter Beachtung dieser Regeln können Ajax-Events wie reguläre Events verwendet werden. Das bedeutet im Besonderen, dass in der Abarbeitungskette des Events auch Module und Container eingeschachtelt werden können. Dadurch behält die DFN auch bei der Verwendung von Ajax-Events einen Großteil ihrer Mächtigkeit.

4.1.4 Erweiterung der DFSL

Die mit der DFN spezifizieren Dialogflüsse werden in die DFSL übertragen. Anhand der DFSL Dokumente erzeugt das DCF das Dialogflussmodell der Anwendung. Aus diesem Grund muss auch die DFSL um Spezifikationsmöglichkeiten für Ajax-Events erweitert werden.

Listing 7: DFSL Dokument mit Ajax-Events

```
1  <ex-mask name="M1">
2      <on-ajax name="ajaxEvent">
3          <call-action>A1</call-action>
4      </on-ajax>
5  </ex-mask>
6
7  <ex-action name="A1">
8      <on-event name="regularEvent">
9          <call-action>A2</call-action>
10     </on-event>
11 </ex-action>
12
13 <ex-action name="A2">
14     <on-ajax name="ajaxEvent">
15         <call-mask>M1</call-mask>
16     </on-ajax>
17 </ex-action>
```

Dazu wird zur Identifikation von Ajax-Events das neue Element `on-ajax` eingeführt. Ausgehend von einer Maske wird damit die Initiierung eines XMLHttpRequests angezeigt. Am Ende der Verarbeitungskette des Events wird das `on-ajax`-Event an

eine Maske gesendet, was dahingehend interpretiert wird, dass diese Maske nicht neu aufgerufen, sondern die clientseitige Framework-Logik angestoßen wird. Wie auch in der DFN lassen sich die neuen Events wie reguläre Events verwenden, so dass sie in den gleichen Kontexten eingesetzt werden können. Listing 7 zeigt das in die DFSL umgewandelte Beispiel der DFN aus Abbildung 7. Dabei fungiert das `on-ajax`-Event als aus- bzw. eingehendes Event zur Maske `M1`.

4.2 Frameworks für die Ajax-Integration in JSF

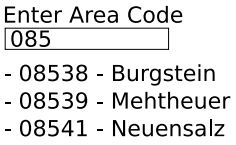
Es wurde bereits gezeigt, wie sich Ajax in den Lebenszyklus JSF-basierender Anwendungen einbinden lässt. Würde man diese reine Integration verwenden, ist allerdings mit den vorhandenen JSF-Komponenten keine Ajax-Kommunikation möglich, da diese keine Requests über das XMLHttpRequest-Objekt absenden können. Daher müssen neue JSF-Komponenten erstellt und in einer eigenen Taglib zusammengefasst werden. Damit diese Ajax-fähig sind, muss weiterhin viel clientseitiger Javascript-Code geschrieben werden. Um diesen Aufwand zu reduzieren empfiehlt sich die Verwendung eines Frameworks, was die genannten Arbeiten kapselt und innerhalb des JSF-Lifecycles verwendet werden kann.

Mittlerweile gibt es eine Vielzahl bereits vorhandenenr Ajax-Frameworks, die das Erstellen Ajax-basierter Anwendungen drastisch beschleunigen und vereinfachen können. Zu unterscheiden ist dabei zwischen reinen Javascript-basierenden Frameworks, die direkt eingebunden werden können und keine Vorgaben an die Anwendung stellen. Die Frameworks machen sich die integrierten Möglichkeiten von Javascript zunutze, die Sprache selbst mittels Prototypen oder durch die Verwendung von Closures zu erweitern. Dies ist im Kapitel Dynamisierung: Javascript auf Seite 13 näher erläutert.

Die Javascript-Frameworks bieten in der Regel ein Wrapper-Objekt für das XmlHttpRequest-Objekt an. Dieses kapselt die Unterschiede zwischen den einzelnen Browsern und die benötigten HTTP-Grundlagen, wie das Setzen der Header. Allerdings müssen bei clientseitigen Frameworks die Webseiten manuell geschrieben und die Ajax-Funktionalität per Hand eingebaut werden. Dafür lassen sich die Frameworks relativ leicht in eine bereits bestehende Anwendung einbinden und diese Anwendung Schritt für Schritt um Ajax-Funktionalität erweitern.

Auf der anderen Seite gibt es serverseitige Ajax-Frameworks, die eine komponentenbasierte Entwicklung ermöglichen. Dazu werden die Komponenten mittels einer serverseitig eingesetzten Programmiersprache entwickelt und automatisch in HTML-Code umgewandelt. Jegliche Aktionen des Benutzers werden per Ajax-Request an den Server gesendet und dort ausgewertet. Deshalb sind diese Anwendungen meist etwas träger als Ajax-Anwendungen, in denen die Präsentationslogik komplett clientseitig gehandhabt wird. Dafür sind diese Frameworks zugänglicher, da keine neue Programmiersprache eingesetzt werden muss, und bieten in der Regel ein solides Fundament an Basiskomponenten an. Da das DCF auf Java basiert, werden im Verlauf nur serverseitige Frameworks betrachtet, die ebenfalls Java-basierend sind.

Abbildung 9: Ausschnitt des Postleitzahlen Beispiels



Zur Verdeutlichung der Unterschiede zwischen den einzelnen Frameworks wird ein kleines Beispiel erstellt, welches sich an der Funktionalität von Google Suggest[15] anlehnt. Während der Eingabe eines Textes in ein Eingabefeld wird dynamisch auf dem Server eine Liste mit vorhandenen Daten geprüft und dem Benutzer zur Auswahl gestellt. Dies kann - neben der Benutzung bei der Suche via Google Suggest - auch zum Einsatz kommen, wenn Kundennummern, Postleitzahlen oder ähnliches in einem Formular eingegeben werden. Abbildung 9 zeigt die Ausgabe der Anwendung. Im Beispiel existiert auf dem Server eine Tabelle, die Postleitzahlen zu Städten zuordnet. Während der Eingabe einer Postleitzahl wird auf dem Server die Tabelle geprüft und dem Benutzer mögliche Postleitzahlen angezeigt.

Im Folgenden werden einige bekannte Vertreter client- und serverseitiger Frameworks vorgestellt, woran sich ein Vergleich dieser einzelnen Frameworks anschließt. Anhand dieser Daten wird dann eines gewählt, welches bei der Implementierung der Ajax-Komponente verwendet werden soll.

4.2.1 prototype / script.aculo.us

Prototype[16] ist ein rein auf Javascript basierendes Framework und bietet eine große Anzahl an Javascript-Erweiterungen zum Umgang mit dynamischem HTML-Code, der als Grundlage für Ajax-Anwendungen dient. Basierend auf prototype wurde scipt.aculo.us[17] entwickelt, welches das Framework um Unterstützung für visuelle Effekte von GUI Elementen erweitert sowie unter anderem eine Drag & Drop Bibliothek implementiert. Da die zu entwickelnde Komponente keine graphischen Besonderheiten aufweisen muss, werden diese Erweiterungen allerdings nicht näher beleuchtet.

Neben der Möglichkeit, asynchrone Anfragen über Ajax durchzuführen, bietet prototype weitere nützliche Javascript-Funktionen an, die Entwickler bei der Verwirklichung häufiger Aufgaben unterstützen. Auf diese soll im Folgenden allerdings nicht eingegangen werden, vielmehr soll erläutert werden, wie sich Ajax-Kommunikation mittels prototype realisieren lässt. Eine Übersicht über die weitergehende Funktionalität des Frameworks findet sich in [PS07].

Zur Realisierung einer asynchronen Anfrage wird das XmlHttpRequest-Objekt bei Prototype durch das Objekt `Ajax.Request` gekapselt. Dieses vereinfacht die Semantik der benötigten Callback-Funktion wesentlich, da ihm nur eine Funktion an-

[15] http://www.google.com/webhp?complete=1&hl=en
[16] www.prototypejs.org
[17] http://script.aculo.us/

gegeben werden muss, die automatisch aufgerufen wird.

4.2.1.1 Verwendung

Listing 8 zeigt die benötigten Schritte, um einen Ajax-Request mittels prototype anzustoßen und den Response auszuwerten. Bei der Eingabe in das angegebene Feld wird automatisch das Backend-Servlet aufgerufen. In `onComplete` wird eine Callback-Funktion angebenen, die aufgerufen wird, wenn der Ajax-Request abgeschlossen ist. Diese aktualisiert den Inhalt des Elements mit der Id `result`. Zur Fehlerbehandlung bietet prototype außerdem noch die Funktionen `onFailure` sowie `onException` an.

Listing 8: PLZ-Suggest mit prototype

```
1 <input type="text" name="areacode" id="areacode" onkeyup="sendRequest()"/>
2 <div id="result"/>
3
4 function sendRequest() {
5  var code = $("areacode").value;
6  new Ajax.Request( "/example/backend?code=" + encodeURI(code), {
7   method: "get",
8   onComplete:function(xhr) {
9    $("result").innerHTML = xhr.responseText;
10  }
11 });
12 }
```

Neben dem Request-Objekt ist es mit den Objekten `Ajax.Updater` und `Ajax.PeriodicUpdater` möglich, Elemente eines HTML-Dokuments automatisch zu akualisieren bzw. periodische Ajax-Requests zu initiieren.

4.2.1.2 Kurzbewertung

Die Verwendung von prototype erlaubt den einfachen Zugriff auf die Methoden des XMLHttpRequest-Objektes. Durch die Kapselung der Unterschiede und Eigenheiten der verschiedenen Browser kann sich die Ajax-Entwicklung auf die eigentliche Funktionalität konzentrieren. Allerdings sind zur Verwendung von prototype in komplexen Applikationen tiefer gehende Kenntnisse sowohl von Javascript als auch dem DOM nötig. Dies hat zur Folge, dass alle Nachteile, die bei der Enwicklung von Ajax-Anwendungen (siehe dazu auch 3.3) vorhanden sind, auch beim Erstellen von Anwendungen mittels prototype auftreten.

4.2.2 Yahoo! User Interface Library

Die Yahoo! User Interface Library (YUI)[18] ist ebenso wie prototype ein clientseitiges Framework zur vereinfachten Integration von Ajax in Webanwendungen. Der Hauptfokus der YUI! liegt dabei - wie der Name schon vermuten lässt - auf der Entwicklung graphischer Benutzeroberflächen. Die Unterstützung der Ajax-Kommunikation ist daher nur ein Teil der Bibliothek, die sich in die folgenden Komponenten unterteilt:

[18] http://developer.yahoo.com/yui/

- Die `Core`-Komponente bietet Zugriff auf die DOM-Elemente eines Dokuments mittels der Verwendung einer einheitlichen Schnittstelle, die von den einzelnen Browser-Spezifika abstrahiert. Sie bietet weiterhin ein vereinheitliches Javascript Event-Handling der Komponenten.
- In der Komponente `Utilities` werden diverse Entwicklungstools bereitgestellt; diese enthalten unter anderem einen Browser-History-Manager, mittels welchem die Navigation innerhalb einer Webanwendung nachgebildet werden kann. Außerdem findet sich hier mit dem Connection Manager die Schnittstelle für den Zugriff auf das XMLHttpRequest-Objekt.
- Weitere Komponenten sind die CSS Tools, Developer Tools für Unit Tests sowie Controls und Widgets, die eine Auswahl vorgefertigter Benutzerelemente wie Kalender, Rich Text Editor oder Autocomplete-Elemente enthält.

4.2.2.1 Verwendung

Listing 9: PLZ-Suggest mit YUI!

```
1 <input type="text" name="areacode" id="areacode" onkeyup="sendRequest()"/>
2 <div id="result"/>
3
4 function sendRequest() {
5  var callback = {
6   success: function(o) {
7    document.getElementById("result").innerHTML = o.responseText;
8   },
9   failure: function(o) {}
10  }
11 var code = document.getElementById("areacode").value;
12 var url = "/example/backend?code=" + code;
13 var transaction = YAHOO.util.Connect.asyncRequest("GET", url, callback, null);
14 }
```

Listing 9 zeigt die Implementierung des Postleitzahlenbeispiels unter Verwendung der Yahoo! User Interface Library. Die Methode `asyncRequest` leitet einen XMLHttpRequest ein und gibt ein Transaktionsobjekt zurück. Als dritter Parameter wird eine Menge von Funktionen übergeben, welche die Verarbeitung der Antwort des Servers übernehmen. Diese enthalten die beiden Methoden `success` sowie `failure`, in welchen die jeweilige Logik implementiert wird.

4.2.2.2 Kurzbewertung

Ebenso wie bei prototype ist man bei der Verwendung der YUI an Javascript gebunden; daher gelten auch die oben genannten Nachteile. Im Vergleich zu anderen clientseitigen Ajax-Frameworks ist die YUI eher eine Sammlung verschiedener Werkzeuge als eine einheitliche Umgebung. Aus diesem Grund sind die Abhängigkeiten zwischen den einzelnen Komponenten sehr gering, so dass nur die wirklich benötigten Teile der Bibliothek eingebunden werden müssen, wodurch sich die Gesamtgröße der zu übertragenden Daten verringert.

4.2.3 RichFaces

RichFaces[19] ermöglich es, Ajax in bestehende JSF Anwendungen zu integrieren, ohne Javascript verwenden zu müssen. Dies wird durch die Erweiterung bestehender JSF-Komponenten um Ajax-Funktionalität realisiert. Durch die Integration von Rich Faces in den JSF Lebenszyklus lässt sich auf serverseitige Validatoren und Konverter ebenso zugreifen wie auf Managed Beans.

Das Framework selbst besteht aus zwei Teilen: der Ajax4JSF-Bibliothek, mit der bestehende JSF-Komponenten erweitert werden können sowie dem RichFaces-Teil, der vorgefertigte Komponenten mit Ajax-Unterstützung bietet. Dadurch ist einerseits komponentenzentrierte (RichFaces) als auch seitenweite (Ajax4JSF) Ajax-Unterstützung möglich. Es ist daher leicht möglch, bereits bestehende Anwendungen um Ajax-Funktionalität zu erweitern. Clientseitig basiert RichFaces auf einer integrierten prototype / script.aculo.us sowie der eigenen Ajax4JSF Ajax Bibliothek.

4.2.3.1 Verwendung

Um die Funktionalität des Frameworks zu nutzen, müssen einer bestehenden JSF-Seite die RichFaces Taglibs `rich` sowie `a4j` hinzugefügt werden. Daraufhin können diese sofort genutzt werden.

Die Tags der Taglib `aj4` ermöglichen es, bestehende Komponenten um Ajax-Fähigkeiten zu erweitern. Mittels der Komponente `a4j:support` wird angegeben, auf welches Javascript-Ereignis eine Komponente reagieren und welcher Teil der Seite beim Eintreten dieses Ereignisses aktualisiert werden soll. Listing 10 zeigt, wie das Postleitzahlenbeispiel unter Verwendung von RichFaces realisiert werden kann.

Listing 10: PLZ-Suggest mit RichFaces

```
1  <%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
2  <%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
3  ...
4  <f:view>
5   <h:form>
6    <rich:panel header="Enter AreaCode">
7     <h:inputText size="50" value="#{bean.areaCode}">
8      <a4j:support event="onkeyup" reRender="areacodes"></a4j:support>
9     </h:inputText>
10    <rich:dataList id="areacodes" var="areaCode" value="#{bean.areaCodes}">
11     <h:outputText value="#{areaCode.code} - #{areaCode.city}"></h:outputText>
12    </rich:dataList>
13   </rich:panel>
14  </h:form>
15 </f:view>
16 ...
```

Das Eingabefeld wird um Ajax-Funktionalität erweitert, die auf Tastendruck (angegeben durch `event="‘onkeyup"’`) reagiert. Dabei wird die Komponente mit der Id `areacodes` mit den Werten der Bean neu gerendert. In dieser Komponente wird das Ergebnis der Ajax-Anfrage in einer Liste angezeigt.

[19] http://www.jboss.org/jbossrichfaces/

4.2.3.2 RichFaces Architektur

Bei der Verwendung asynchroner Kommunikation mit dem RichFaces Framework reagiert die clientseitige Ajax-Engine auf Javascript-Ereignisse und sendet dazu gehörige Requests mittels des XMLHttpRequest-Objektes an den Server. Zur Unterscheidung der verschiedenen Request-Typen muss serverseitig ein Ajax-Filter registriert werden.

Im Rahmen der Verarbeitung von Requests kommen im RichFaces Framework die folgenden Kernbestandteile zum Einsatz:

- Ajax Filter: Dieser ServletFilter wird beim Aufruf des zentralen FacesServlets ausgeführt und ist für die Unterscheidung der verschiedenen Request-Typen zuständig. Handelt es sich um einen Ajax-Request, parst der Filter den Response, woraufhin nur die notwendigen Teile des DOM-Baums an den Client gesendet werden.
- Ajax Action Components: Die Action Components sind JSF-Komponenten, die zur Initialisierung von Ajax-Requests verwendet werden. Werden diese statt der normalen `commandButton` bzw. `commandLink` verwendet, sendet die Javascript Engine des RichFaces Framework einen XMLHttpRequest ab.
- Ajax Container: Innerhalb eines Ajax-Containers werden auf einer JSF-Seite diejenigen Bereiche angegeben, die während eines Ajax-Requests verarbeitet werden sollen. Dadurch kann vermieden werden, dass der JSF-Lifecycle aufgrund von Fehlern während der Validierung von Komponenten, die für den aktuellen Request nicht benötigt werden, abbricht.
- Javascript Engine: Die clientseitige Javascript Engine stellt die Low Level API des RichFaces Frameworks bereit, mit der asynchrone Request initiiert werden können.

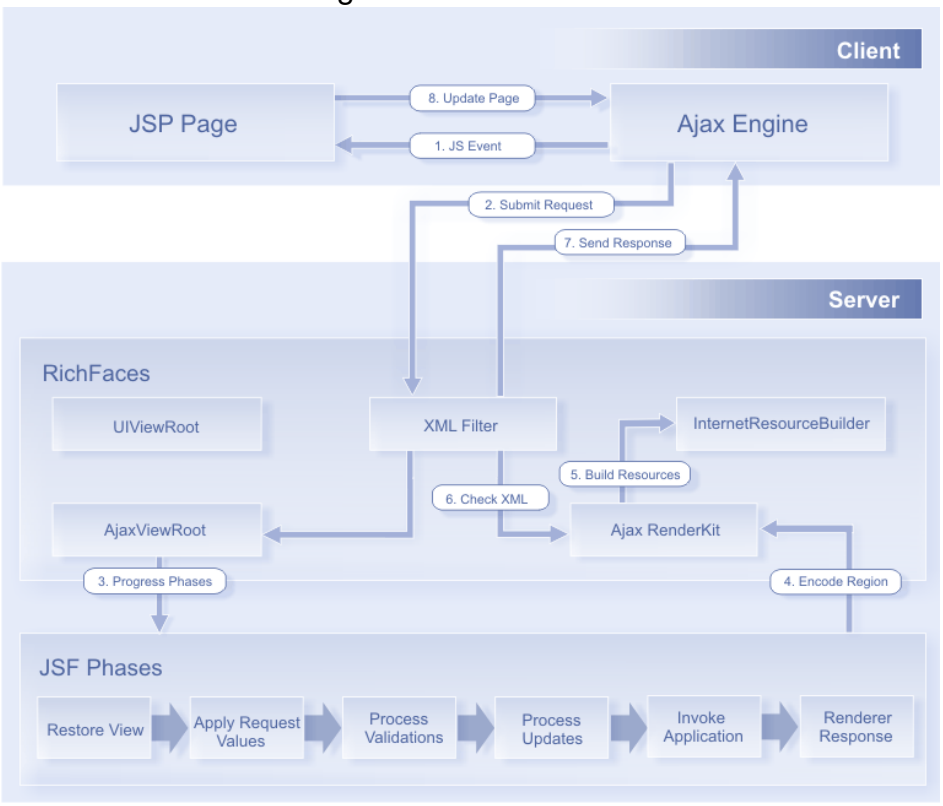
Abbildung 10[20] fasst die Architektur des Frameworks zusammen und zeigt an welche Schritte beim Absenden eines Requests durchlaufen werden.

Wie in allen Ajax-Anwendungen wird der Request hier nicht direkt von der graphischen Benutzeroberfläche ausgelöst sondern von der clientseitigen Ajax Engine. Nachdem der Request dekodiert und auf eventuelle Ajax-Bestandteile durchsucht wurde, wird der reguläre JSF-Lifecycle durchlaufen. RichFaces macht also keinen Gebrauch von der Möglichkeit, PhaseListener zu registrieren und dadurch Rechenzeit auf dem Server zu sparen.

Am Ende des JSF-Lifecycles werden wieder RichFaces-Komponenten aufgerufen um die Response zu erstellen. Der `XMLFilter` extrahiert dabei die benötigten Teile des Komponentenbaums, die aktualisiert wurden und sendet den DOM-Baum an die clientseitige Engine, die daraus ein Delta-DOM erstellt und die entsprechenden Elemente neu rendert.

[20] Quelle: JBoss RichFaces Handbuch, Chapter 5. Basic concepts of the RichFaces Framework: http://www.jboss.org/jbossrichfaces/docs/

Abbildung 10: RichFaces Architektur



4.2.3.3 Kurzbewertung

Mittels des RichFaces Frameworks lassen sich bestehende JSF-Anwendungen sehr leicht um Ajax-Funktionalität erweitern. Dies wird durch die Möglichkeit, bestimmten Komponenten Ajax-Fähigkeiten zu verleihen, erreicht. Dadurch ist es bei der Integration in bestehende Anwendungen nicht erforderlich, alle JSF-Seiten auf einmal zu ändern; stattdessen kann dies iterativ erfolgen. Begünstigt wird die Integration weiterhin durch einfache Konfiguration des Frameworks.

Bei jedem Ajax-Request wird der komplette JSF-Lifecycle durchlaufen, was bei einer hohen Zahl an Requests zu Lastspitzen führen kann. Es wäre möglich, dieses zu umgehen, indem ein eigener PhaseListener registriert wird, der bei Ajax-Request entsprechende Verkürzungen des Lifecycles vornimmt.

4.2.4 ICEFaces

Das ICEFaces Framework[21] ist ebenso wie RichFaces eine Erweiterung des JSF Frameworks, mit der sich Ajax-Anwendungen vollständig in Java erstellen lassen. Es basiert auf einer Bibliothek von JSF-Komponenten, die um Ajax-Funktionalitäten erweitert wurden.

Bei der Verwendung von ICEFaces wird der Standard JSF Renderer durch einen

[21] http://www.icefaces.org

Direct-to-DOM Renderer (D2D) ersetzt, welcher statt einer kompletten Seite nur ein Delta DOM (siehe 4.1.2.2) an den Client zurücksendet. Mittels dem D2D Renderer ist es möglich, einen JSF Komponentenbaum direkt in einen DOM-Baum zu rendern.

4.2.4.1 Verwendung

Die Umsetzung des Beispiels erfolgt unter Verwendung von Komponenten der ICEFaces Taglib, was in Listing 11 dokumentiert ist. Mit `selectInputText` wird eine Komponente bereitgestellt, die die Auswahl eines Wertes aus einer Liste ermöglicht. Dieser wird ein valueChangeListener angegeben, welcher aufgerufen wird, wenn sich der eingegebene Wert ändert. Die Listener-Methode gibt dann eine Liste passender Postleitzahlen zurück.

Listing 11: PLZ-Suggest mit ICEFaces

```
1  <f:view xmlns:f="http://java.sun.com/jsf/core"
2   xmlns:h="http://java.sun.com/jsf/html"
3   xmlns:ice="http://www.icesoft.com/icefaces/component">
4  ...
5  <ice:form>
6   <ice:selectInputText valueChangeListener="#{bean.updateAreaCode}">
7    <f:selectItems value="#{bean.areaCodeItems}" />
8   </ice:selectInputText>
9  </ice:form>
10 ...
```

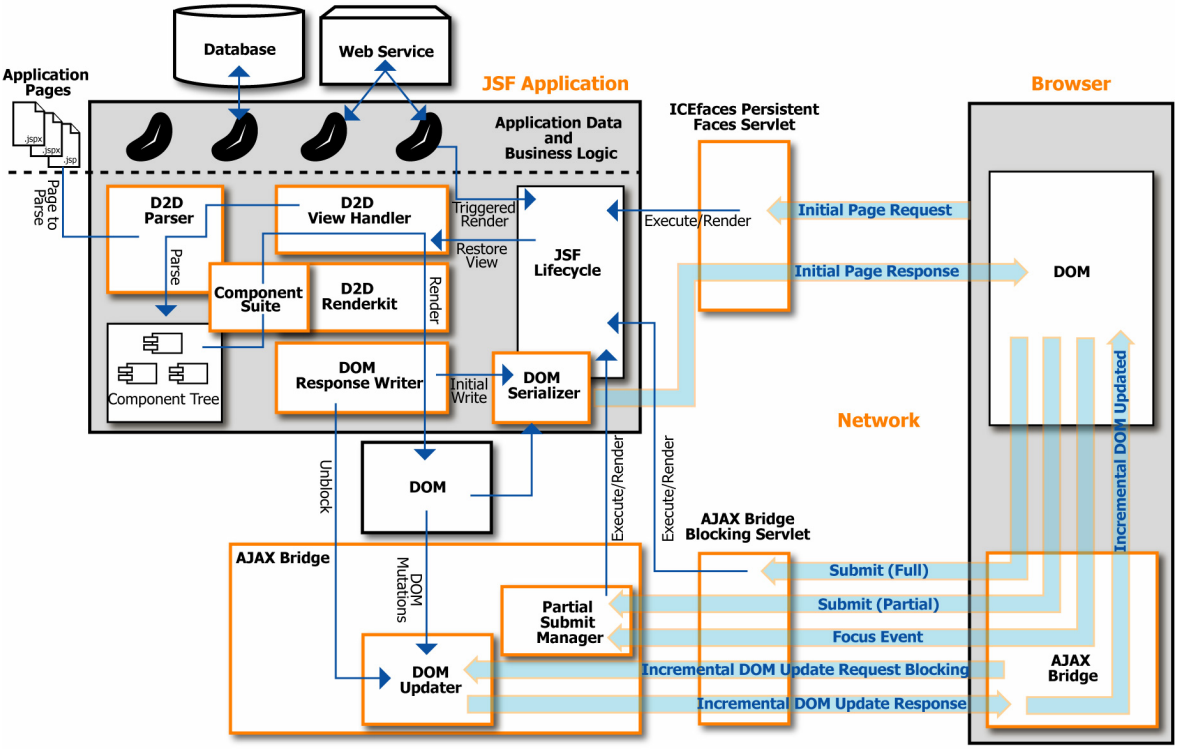
4.2.4.2 ICEFaces Architektur

Das ICEFaces Framework ändert den JSF-Lifecycle ab, so dass individuelle Komponenten zum Einsatz kommen, die statt HTML einen DOM-Baum zurückliefern. Dabei ersetzt das Persistent Faces Servlet das Standard Faces Servlet und übernimmt die Bearbeitung des initialen Requests eines Clients und sendet den zugehörigen Response an diesen. Nachdem die Einstiegsseite an den Client übertragen wurde, übernimmt das Blocking Servlet die weitere Bearbeitung. Es unterscheidet zwischen blockierenden (synchronen) Requests, die direkt an den JSF Lifecycle weitergeleitet werden sowie nicht blockierenden (asynchronen) Requests, die von der ICEFaces Ajax Bridge verarbeitet werden. In der RestoreView Phase des JSF Lifecycles ruft der D2D View Handler den D2D Parser auf, der einen JSF Komponentenbaum aus einer JSP erstellt. Dieser Komponentenbaum wird im Rahmen der RenderResponse Phase mittels des D2D RenderKits in einen DOM-Baum überführt.

Ist die initiale Seite an den Client ausgeliefert, übernimmt die clientseitige Ajax Bridge die weitere Bearbeitung von Requests. Diese werden dadurch an das Blocking Servlet gesendet, welches bei asynchronen Requests die serverseitige Ajax Bridge aufruft. Innerhalb der Bridge verarbeitet der Partial Submit Manager den Request und initiiert den JSF-Lifecycle, nach dessen Abschluss der DOM Updater aufgerufen wird. Dieser vergleicht den bisherigen DOM-Baum mit dem zurückgelieferten und ermittelt die Unterschiede zwischen beiden. Daraus wird ein Delta DOM-Baum generiert, welcher an den Client gesendet wird, wo die clientseitige Ajax

Bridge für die Einarbeitung der Änderungen an der Seite verantwortlich ist. Da nur die geänderten Daten übertragen werden, verringert sich das zu übertragende Datenvolumen und damit auch die Wartezeit beim Verwenden der Anwendung. Einen Überblick über die Architektur sowie das Zusammenspiel der ICEFaces Komponenten gibt Abbildung 11[22].

Abbildung 11: ICEFaces Architektur



Die meisten Komponenten des ICEFaces Frameworks sind so eingestellt, dass sie bei Änderungen automatisch die zugehörigen Backing Beans aktualisieren. Diese Partial Submit genannte Technik sorgt dafür, dass nur das jeweils aktive Formularfeld validiert wird, wodurch es keine Fehlermeldungen für abhängige Eingabefelder gibt.

4.2.4.3 Serverseitige Aktualisierungen

In ICEFaces integriert ist die Möglichkeit mittels der API des `PersistentFacesState`, Änderungen am Zustand der Anwendung direkt an den Client weiterzureichen, ohne auf eine Benutzerinteraktion angewiesen zu sein. Ermöglicht durch die Verwendung langlebiger HTTP-Verbindungen kann damit eine Push-basierte Kommunikation (siehe 3.3.4, Seite 30) verwendet werden. In ICEFaces ist dies als Erweiterung des JSF-Komponentenmodells implementiert, wodurch Ajax-Push komplett in Java realisiert werden kann. Da durch die langlebige HTTP-Verbindung bereits eine Verbindung je Client verwendet wird, verwendet ICEFaces auf der Clientseite Connection Sharing für Push-Updates.

[22] Quelle: ICEfaces Developer's Guide Community Edition, Chapter 2 ICEfaces System Architecture http://www.icesoft.com/developer_guides/icefaces/htmlguide/devguide/sys_architecture.html

Auf der Serverseite wird im Rahmen des Java Specification Request 315[23] an einer neuen Spezifikation für Java Servlets mit asynchronen Verbindungen gearbeitet. ICEFaces integriert einen asynchronen HTTP Server, der neben einem vorhandenen Application Server für die Behandlung der Push-Requests zum Einsatz kommt.

4.2.4.4 Kurzbewertung

Die Komplexität von ICEFaces ist sowohl Vor- wie Nachteil. Durch die tiefe Integration in den JSF-Lifecycle werden unnötige Arbeitsschritte bei der Verwendung von Ajax-Requests vermieden. Weiterhin steht aufgrund der einfachen Integration Push-basierter Kommunikation eine weitere mächtige Interaktionsmöglichkeit zur Verfügung. Demgegenüber stehen Inkompatibilitäten mit anderen JSF-Komponenten, die auftreten können, wodurch sich die Integration in bereits bestehende Anwendungen erschwert.

4.2.5 J4Fry

Die beiden bisher vorgestellten JSF Ajax-Frameworks greifen tief in den JSF-Lifecycle ein und liefern eigene Ajax-fähige Komponenten, welche die Standardkomponenten ersetzen. Die Grundidee hinter J4Fry[24] dagegen ist es, die vorhandenen Komponenten nicht zu verändern, so dass Anwendungen auch mit neueren JSF-Versionen verwendet werden können, ohne das eigentliche Framework zu aktualisieren.

Listing 12: PLZ-Suggest mit J4Fry

```
1 <%@ taglib prefix="fry" uri="http://j4fry.org/jsf"%>
2 ...
3 <h:form>
4  <h:inputText value="#{bean.areaCode}" id="text">
5   <fry:ajax event="onkeyup" reRender="output"/>
6  </h:inputText>
7 </h:form>
8
9 <h:outputText id="output" value="#{bean.areaCode}"></h:outputText>
```

Ausgehend von den Standardkomponenten kann diesen mittels einer Ajax-Komponenten zusätzliche Funktionalität hinzugefügt werden - ähnlich wie beim ajax4JSF-Teil des RichFaces Frameworks. Dazu gibt es die `j4fry:ajax`-Komponente, deren Verwendung in Listing 12 gezeigt ist. In ihr ist sowohl eine JSF-Action definierbar, welche beim Auslösen bestimmter Ereignisse aufgerufen wird als auch die Id einer Komponente, die aktualisiert werden soll.

Durch einen integrierten PhaseListener wird bei J4Fry der JSF-Lifecycle vor der RenderResponse-Phase abgebrochen. Dieser Listener übernimmt das Rendern der neu darzustellenden Komponenten mittels JSON. Durch das Ersetzen der komplexen Renderphase wird der Server bei Ajax-Requests weitaus weniger belastet. Ebenso wie ICEFaces unterstützt J4Fry Partial Submit mittels eines PhaseListeners.

[23] http://jcp.org/en/jsr/detail?id=315
[24] http://www.j4fry.org

4.2.6 Weitere Frameworks

Zur Vervollständigung werden an dieser Stelle noch zwei weitere Frameworks, denen ganz andere Herangehensweisen zugrunde liegen, vorgestellt. Sie erscheinen allerdings aufgrund ihrer Natur nicht dazu geeignet, als DCF Erweiterung zu fungieren.

4.2.6.1 Direct Web Remoting

Direct Web Remoting (DWR)[25] ist ein serverseitiges Ajax-Framework, welches darauf basiert, automatisch Javascript aus bestehenden Java-Klassen zu erzeugen. Die beiden Hauptbestandteile sind das DWR-Servlet, welches Requests verarbeitet und Responses an den Client zurück sendet sowie Javascript auf der Clientseite, welches die Requests auslöst und den Inhalt der Website automatisch aktualisieren kann. Dabei findet keine Ausführung direkt im Browser statt, sondern es werden nur Daten zwischen Client und Server hin- und hergeschickt, die auf dem Server verarbeitet werden.

Der clientseitige Javascript-Code wird aus bestehenden Java Beans erzeugt, deren Methoden ganz oder teilweise offen gelegt werden. Dazu werden die Beans in einer XML-Datei referenziert und DWR erzeugt den daraus resultierenden Javascript-Code, der die Beans auf dem Client emuliert, automatisch.

4.2.6.2 Google Web Toolkit

Das Google Web Toolkit (GWT)[26] bietet eine ganz andere Herangehensweise als die bisher vorgestellten Frameworks an. Anwendungen unterteilen sich beim GWT in Client- und Serverkomponenten, wobei beide Teile komplett in Java entwickelt werden.

Zum Erstellen von graphischen Oberflächen wird eine Bibliothek dynamischer, wiederverwendbarer Komponenten - genannt Widgets - angeboten, die den Komponenten der Swing-Bibliothek ähneln. Durch Kombination einzelner Widgets lassen sich somit individuelle Benutzeroberflächen erstellen. Weiterhin wird clientseitig eine Teilmenge der Java Klassenbibliothek emuliert, die wichtige Klassen aus den Paketen `java.lang` sowie `java.util` enthält. Damit sind auch anspruchsvollere Anwendungen möglich, bei denen der Schwerpunkt der Logik auf der Clientseite liegen kann.

Im Gegensatz zu den übrigen serverseitigen Frameworks wie RichFaces oder ICEFaces werden die Komponenten nicht auf dem Server belassen. Im GWT ist ein Compiler enthalten, der Java Code direkt in Javascript, HTML und CSS Dateien übersetzt. Diese reagieren wie statische Dokumente, so dass Rechenzeit auf dem

[25] http://directwebremoting.org/
[26] http://code.google.com/intl/de-DE/webtoolkit/

Server gespart wird. Außerdem wird dadurch auch die Anzahl an Requests im Gegensatz zu komplett serverseitigen Frameworks verringert, da nur kommuniziert wird, wenn tatsächlich neue Daten benötigt werden.

Die Interaktion zwischen Client und Server wird über sogenannte Remote Procedure Calls ermöglicht. Damit ist es möglich, Daten zu serialisieren und Java-Objekte zwischen dem Client und dem Server auszutauschen. Auf der Serverseite wird dazu ein vom GWT bereit gestellter `RemoteService` implementiert, der einem einfachen Java Servlet entspricht. Clientseitig werden ein synchrones sowie ein asynchrones Service-Interface definiert. Der Client verwendet die asynchrone Schnittstelle zur Kommunikation mit dem Server.

4.3 Vergleich der Frameworks

Im Folgenden soll ein Vergleich der vorgestellten Frameworks anhand diverser Kriterien erfolgen. Dabei wird zuerst auf den Unterschied zwischen client- und serverseitigen Frameworks eingegangen. Da sich serverseitige im Test als praktikablere Möglichkeit zur Erweiterung des DCF erwiesen haben, wird sich die Gegenüberstellung auf die vorgestellten Vertreter ICEFaces, j4fry und RichFaces konzentrieren.

4.3.1 Vergleich client- und serverseitiger Lösungen

Bei der Verwendung clientseitiger Javascript-Frameworks werden keine speziellen Anforderungen an den Server gestellt. Das heißt also, dass jede serverseitige Programmiersprache verwendet werden kann. Die Minmalanforderung, einfache Daten in Textform an den Client zu senden, erfüllen alle Programmiersprachen. Daher lassen sich eigenständige Javascript-Frameworks ohne weitere serverseitige Anpassungen in bestehende Anwendungen integrieren.

Weiterhin sind clientseitige Frameworks in der Regel weitaus effizienter und vor allem kleiner als serverseitige. Der eigentliche Framework-Code muss nur einmal auf den Client übertragen werden und braucht nicht nachgeladen zu werden. Dadurch verringert sich die zu versendende Datenmenge und das Zeitverhalten der gesamten Anwendung wird besser. Da Teile der Anwendungslogik auf den Client ausgelagert werden, verringert sich weiterhin die Serverlast, wodurch dessen Kapazitäten für andere Aufgaben zur Verfügung stehen.

Außerdem kann die Anzahl der Requests drastisch gesenkt werden, da viele Aufgaben direkt vom Client übernommen werden können. So ist es ohne Probleme möglich, große Datenbestände clientseitig zu sortieren; im Falle serverseitiger Frameworks wird zwar auch Datenvolumen im Vergleich zu einer Ajax-freien Lösung gespart, allerdings muss der Server alle sortierten Daten wieder an den Client senden. Dieser Schritt entfällt, wenn der Client diese Aufgabe übernimmt.

Nachteile ergeben sich durch die Verwendung von Javascript. Zur Benutzung der Frameworks sind in der Regel detaillierte Kenntnisse in Javascript sowie in

HTML/CSS nötig. Neben dem Code auf der Serverseite muss nun auch clientseitig Code gewartet werden. Da auf dem Server eine andere Programmiersprache verwendet wird, kommt es auch zu Brüchen in der Objektserialisierung (eine Ausnahme hiervon bildet das GWT), da Daten zuerst in XML, JSON oder andere textbasierte Formate umgewandelt werden müssen. Dadurch erhöht sich der Wartungsaufwand der gesamten Anwendung. Zudem gilt Javascript aufgrund seiner Typenlosigkeit als fehleranfällig.

Wenn Javascript verwendet wird, um komplexere Dialoge zu erstellen, muss deren Präsentationslogik komplett clientseitig erstellt werden. Daher muss die bestehende serverseitige Architektur aufgeteilt und in Javascript neu implementiert werden, wodurch sich eine weitaus komplexere verteilte Architektur ergibt.

Serverseitige Ajax-Lösungen dagegen lassen sich ohne das Erlernen einer neuen Programmiersprache in Anwendungen integrieren, da sie komplett mit derjenigen Sprache, die auf dem Server verwendet wird, entwickelt werden. Dadurch sind mächtigere Sprach- und Implementierungskonstrukte wie automatisierte Unit-Tests, Typisierung sowie generische Datentypen verfügbar. Der Ajax-Komponente steht die gesamte bereits vorhandene Anwendungsarchitektur auf dem Server zur Verfügung, so dass keine redundanten Implementierungen vorgenommen werden müssen. Durch diese Integration lässt sich der Wartungsaufwand von Ajax-Anwendungen beträchtlich senken, wodurch Entwicklungskosten gespart werden.

Wie oben bereits beschrieben, liegt der größte Vorteil serverseiter Lösungen in der Kommunikation zwischen Client und Server. Da sie die vorhandenen Datentypen verwenden können, ist keine Umwandlung in XML- oder JSON-Strukturen (also textbasierende Datenformate) notwendig. Gerade das GWT bietet mit dem `IsSerializable`-Interface einen interessanten Ansatz, mit dem vollständige Objekte serialisiert werden können. Dies ist vor allem bei der Verwendung vieler benutzerdefinierter Datentypen von Vorteil.

Da serverseitige Frameworks zur Ausführung im Browser in HTML- und Javascript-Code umgewandelt werden, enthalten sie selber auch ein clientseitiges Javascript-Framework. Dieses kann direkt genutzt werden. Es können aber auch andere clientseitige Frameworks integriert werden. Dadurch ist beispielsweise die Verwendung von ICEFaces möglich, welches um script.aculo.us erweitert wurde, um individuellere Benutzeroberflächen zu erzeugen. Die Erweiterung des DCF soll sich allerdings vorerst auf die Anbindung auf Serverseite konzentrieren.

4.3.2 Vergleich serverseitiger Frameworks

Der DialogController als zentrale Steuerungsinstanz des DCF muss auch bei der Verwendung von Ajax-Kommunikation mit einem Framework angesteuert werden. Daher muss es möglich sein, die Abarbeitung auf dem Server anzupassen, so dass der `DialogController` verwendet wird. Nur dann kann die Anwendung auch von den erweiterten Dialogsteuerungsmöglichkeiten des DCF profitieren. Wie oben bereits gezeigt wurde, wird dies erreicht, indem das DCF als JSF NavigationHandler verwendet wird. Dies ist bei allen drei vorgestellten serverseitigen Lösungen mög-

lich, da sie keinen eigenen NavigationHandler mit sich bringen.

4.3.2.1 Auswirkungen auf den JSF-Lifecycle

ICEFaces greift am tiefsten in den JSF-Lifecycle ein und ersetzt viele der Standardkomponenten durch eigene. Dadurch sind umfangreiche Änderungen an bereits bestehenden Applikationen nötig. Außerdem besteht die Gefahr, dass andere Komponentenframeworks wie Apache Trinidad oder MyFaces Tobago mit ICEFaces nicht mehr funktionieren. Demgegenüber steht bei ICEFaces ein an Ajax-Interaktion angepasster Lifecycle. Durch den Einsatz des D2D Renderers wird die benötigte Zeit zum Rendern einer Seite verkürzt; mittels Partial Submit lassen sich ausgereifte Formularvalidierungen implementieren.

RichFaces dagegen verwendet einen ServletFilter für Ajax-Requests und greift daher kaum in den JSF-Lifecycle selber ein. Der Filter ist verantwortlich für die Unterscheidung der Request-Typen und das darauf aufbauende Rendern der Response. Daher wird in der RenderResponse-Phase der komplette Komponentenbaum gerendert und in das jeweilige Ausgabeformat umgewandelt, bevor im Filter mittels eines Parsers die benötigten Teile extrahiert werden. Dadurch entsteht eine Menge Overhead bei der Verarbeitung vieler kleiner Ajax-Requests, was sich negativ auf die Performance beim Einsatz von RichFaces auswirkt.

Einen Mittelweg durch die Verwendung eines PhaseListeners verfolgt J4Fry. Hier wird im Falle eines Ajax-Requests vor der RenderResponse-Phase abgebrochen und ein eigener Renderer eingesetzt. Alle anderen JSF-Bestandteile werden nicht berührt und können normal genutzt werden. Dadurch lässt sich J4Fry problemlos mit anderen Komponentenframeworks zusammen einsetzen. Dies ist für die Entwicklung ansprechender graphischer Benutzeroberflächen allerdings auch notwendig, da mit J4Fry selber keine Ajax-fähigen Komponenten wie bei RichFaces oder ICEFaces mitgeliefert werden.

4.3.2.2 JSF-Komponenten der Frameworks

Sowohl RichFaces als auch ICEFaces bieten eine große Anzahl vorgefertigter Ajax-fähiger Komponenten an. Dadurch ist es schnell möglich, ansprechende graphische Benutzeroberflächen zu entwickeln. Bei der Verwendung von RichFaces wie auch bei J4Fry ist es zusätzlich möglich, bereits bestehende Komponenten um Ajax-Funktionalitäten zu erweitern. Dies ist insbesondere notwendig, wenn vorhandene Anwendungen nicht komplett umgestellt, sondern sukkzessive um Ajax-basierte Nutzerinteraktionen erweitert werden sollen.

Bei der Verwendung vorgefertigter Komponenten muss allerdings beachtet werden, dass diese JSF-Events erzeugen können, so dass das DCF als NavigationHandler verwendet wird. Ist dies nicht der Fall würde ein statischer Dialogfluss erzeugt, der nicht mittels der DFN spezifiziert wurde.

4.3.3 Auswahl eines Frameworks

Es ist prinzipiell möglich, alle vorgestellten Frameworks zur Anwendungsentwicklung mit dem DCF zu verwenden, da keines einen eigenen NavigationHandler verwendet und damit die Rolle des DCF verdeckt. Um einen möglichtst einfachen Übergang von klassischen zu Ajax-Anwendungen zu gewährleisten, wurde im Rahmen dieser Arbeit das RichFaces Framework ausgewählt. Es bietet durch die integrierte Ajax4JSF-Basis eine unkomplizierte Möglichkeit, bestehende Anwendungen um Ajax zu erweitern. Dies passt sich in die ebenfalls rückwärtskompatible DCF-Erweiterung ein.

Werden vorgefertigte Ajax-fähige Komponenten verwendet, muss im Rahmen der Einbindung in eine DCF-basierende Anwendung geprüft werden, ob diese so angepasst werden können bzw. müssen, dass sie das DCF verwenden und keine Action-Methoden per JSF Expression Language aufrufen. In diesem Fall würde die Komponente einen statischen Dialogfluss vorgeben und nicht die Navigationsmöglichkeiten des DCF verwenden.

4.4 Zusammenfassung

In diesem Kapitel wurden die beteiligten Techniken vorgestellt, mit denen das DCF um Ajax-basierte Nutzerinteraktionen erweitert wird. Dazu wurde gezeigt, wie das DCF mit dem JSF Framework kombiniert werden kann. so dass eine weitere Abstraktionsebene geschaffen wurde, die es ermöglicht, wiederverwendbare GUI-Komponenten zu erstellen. Mit der Integration in das JSF Framework ist durch die JSF Expression Language auch ein vereinfachter Zugriff auf Request-Daten über Managed Beans möglich. Da die Frameworks auf unterschiedliche Teile einer MVC-basierenden Anwendung fokusiert sind, profitieren beide Teile von einer Verknüpfung.

Nachdem dann gezeigt wurde, dass es auch möglich ist, Ajax in den JSF-Lifecycle zu integrieren, wurden Möglichkeiten vorgestellt, mit denen das DCF um Ajax-Interaktionen erweitert werden kann. Dazu sind Änderungen beim Parsen der DFSL-Dokumente sowie beim eigentlichen Verarbeiten von Events notwendig.

Da zur Verwendung von Ajax-Interaktionen Requests asynchron gesendet werden müssen, müssen JSF-Komponenten entwickelt werden, die solche Requests abschicken können. Um auf eine bereits vorhandene Basis solcher Komponenten aufzubauen, ist es sinnvoller diese nicht von Hand zu entwickeln, sondern auf ein Framework zurückzugreifen, welches Komponenten anbietet. Es wurden verschiedene Frameworks, die unterschiedliche Ansätze verfolgen, vorgestellt und verglichen. Im Rahmen des DCF hat sich RichFaces als eine gute Wahl herausgestellt, da es einerseits fertige Ajax-fähige Komponenten bereitstellt, andererseits aber auch die Möglichkeit offen lässt, vorhandene Komponenten um Ajax zu erweitern.

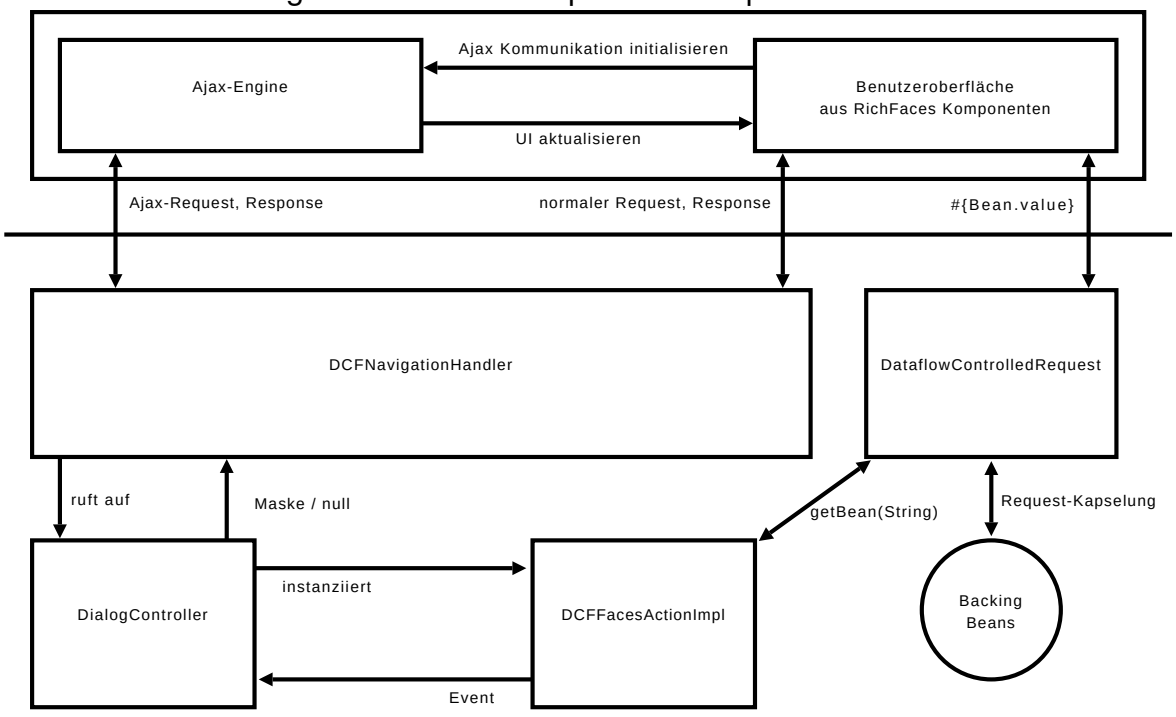
Im nachfolgenden Kapitel soll nun ausgehend von den hier vorgestellten theoretischen Ansätzen dargelegt werden, wie diese praktisch umgesetzt wurden.

5 Implementierung der Ajax-Komponente

Nachdem im vorangegangenen Kapitel erläutert wurde, wie sich die Technologien JSF, Ajax und DCF kombinieren lassen, soll an dieser Stelle beschrieben werden, wie die eigentliche Integration vollzogen wird. Dazu werden zuerst die Änderungen am DCF und JSF Framework vorgestellt und gezeigt, wie mit dem DCF auf JSF Bestandteile zugegriffen werden kann. Daran anschließend wird erläutert, welche Bereiche des DCF angepasst wurden, um Ajax-basierte Nutzerinteraktionen zu ermöglichen und wie das JSF-Ajax-Framework RichFaces im Rahmen des DCF zu verwenden ist.

Aufgrund der Natur des DCF ergeben sich einige Einschränkungen hinsichtlich der Komplexität der zu erstellenden Komponente, welche sich an den Einsatzbereichen des DCF orientieren soll. Die Erstellung komplexer graphischer Benutzeroberflächen oder Verwaltung größerer Multimediainhalte ist hier nicht das Ziel. Stattdessen liegt der Fokus auf der Behebung von Defiziten in der Nutzerinteraktion.

Abbildung 12: Zusammmenspiel der Komponenten des DCF



Der erste Schritt der Implementierung ist die Integration des DCF mit JSF, wodurch Benutzeroberflächen mit einem weit verbreiteten Framework erstellt werden können. Aufbauend darauf wurde das DCF um die Möglichkeit der Generierung und Verarbeitung von Ajax-Events erweitert, so dass diese eingesetzt werden können.

Damit clientseitig von der Verwendung Javascripts abstrahiert wird, wurde am Ende das RichFaces Framework eingebunden, welches Ajax-fähige JSF-Komponenten bereitstellt. Abbildung 12 bietet einen Überblick über das Zusammenspiel der einzelnen Komponenten, welches im Folgenden näher erläutert werden soll.

5.1 Anpassungen am DCF zur Integration in JSF

Da das DCF selber nur die Dialogsteuerung einer Anwendung übernimmt, muss es mit anderen Techniken kombiniert werden, die die Anzeige der Masken übernehmen. Bisher war dies nur durch den Einsatz von JSPs mit integrierten Scriptlets möglich. In Kapitel 4 wurden bereits Vorteile von JSF genannt; daher wurde das DCF so angepasst, dass es in den JSF-Lifecycle integriert werden kann.

Dazu wird der Navigationsmechanismus des JSF Frameworks durch das DCF ersetzt und es kann direkt aus dem DCF heraus auf Beans zugegriffen werden. Die nächsten Abschnitte sollen die Änderungen am DCF verdeutlichen.

5.1.1 DCFNavigationHandler

Wie in Kapitel 4 beschrieben, soll das DCF als NavigationHandler im JSF-Lifecycle eingebunden werden. Der vom JSF Framework bereitgestellte Navigationsmechanismus arbeitet nach dem gleichen Prinzip wie das DCF. Beide erwarten als Eingabe einen String (im DCF der Eventname, im JSF eine Action) und geben den Namen der Seite zurück, welche als nächste angezeigt werden soll. Mittels einer Konfigurationseinstellung in der `faces-config` (siehe Listing 13) ist es möglich, den `DCFNavigationHandler` einzubinden und den Standard-JSF-Handler zu ersetzen.

Listing 13: Konfiguration des DCFNavigationHandlers innerhalb der faces-config

```
1 <faces-config>
2     <application>
3         <navigation-handler>org.dialogcontrol.dcf.faces.DCFNavigationHandler</navigation-handler>
4     </application>
5 </faces-config>
```

Der `DCFNavigationHandler` ist verantwortlich für das Laden des DialogControllers für einen spezifischen Präsentationskanal sowie das Entgegennehmen des Eventnamens und das Weiterleiten an den DialogController. Um das DCF korrekt zu initialisieren wurde weiterhin mittels eines PhaseListeners in den JSF-Lifecycle eingegriffen, so dass beim erstmaligen Zugriff auf die Anwendung der DialogController initialisiert wird.

Da im JSF-Kontext das `FacesServlet` als zentrales Servlet eingesetzt wird, ist es nicht mehr notwendig, den `PresentationChannel` des DCF direkt aufzurufen. Dieses muss deshalb aus der Konfiguration der Webanwendung entfernt werden.

5.1.2 DCFFacesActionImpl, Zugriff auf Managed Beans

Die Actions im DCF müssen das Interface `Action` implementieren, in dem Methoden bereitgestellt werden, mit denen auf vorhandene Daten zugegriffen werden kann. Zur besseren Integration in den JSF-Kontext wurde eine eigene Implementierung `DCFFacesActionImp` erstellt, die mittels der Methode `getBean` einen einfacheren Zugriff auf die Beans des JSF Frameworks ermöglicht.

Die im DCF enthaltene Datenflusssteuerung des DataflowControllers (DFC) verwaltet den Zugriff auf Attribute des Requests [Ric06]. Befindet sich der DFC im strikten Modus, müssen alle Zugriffe innerhalb der Dialogflussspezifikation definiert werden. Dadurch wird auch automatisch der Zugriff auf diejenigen Beans gekapselt, die sich im Request-Scope befinden. Bei der Definition von Beans innerhalb der `faces-config` muss beachtet werden, dass der Name der Bean dem Namen des Datums aus dem DFC entspricht. Auch beim Zugriff auf Beans mittels der JSF-eigenen Expression Language wird die Datenflusssteuerung verwendet, so dass diese auch innerhalb von Masken zum Tragen kommt.

5.1.3 Verwendung von JSF-Komponenten im Rahmen des DCF

Es gibt JSF-Komponenten (z.B. `commandButton` oder `commandLink`), denen mittels Expression Language im Parameter action angegeben wird, dass sie die Methode einer registrierten Bean aufrufen sollen. Bei der Verwendung des DCF sollte auf diese Möglichkeit allerdings verzichtet werden, da dies einem statischen Dialogfluss entspräche, der nicht durch die Dialogflussspezifikation vorgegeben ist. Äquivalent dazu sollte auf den Einsatz von Validatoren verzichtet werden, die mittels JSF registriert werden, da auch diese statische Dialogflüsse vorschreiben.

Das DCF bietet selber Actions an, welche die genannten Aufgaben übernehmen können. So kann ein Validator nachgebildet werden, indem beim Klick auf den Sende-Button eines Formulares eine entsprechende Action aufgerufen wird, die die Eingabe validiert. Im Dialogfluss wird dabei spezifiziert welches Event aufgerufen werden soll, wenn alle erforderlichen Daten korrekt eingegeben wurden bzw. welches, wenn fehlerhafte Daten vorliegen.

5.2 Anpassungen am DCF zur Integration von Ajax-Events

In der Dialog Flow Notation wurde ein neues Event eingeführt, welches auf Ajax-Ereignisse reagieren soll. Dieses Event muss ein entsprechendes Gegenstück in der Implementierung finden. Dazu wird eine neue Klasse `AjaxEvent` erstellt, welche dieses repräsentiert. Diese wird von der Klasse `DialogEvent` abgeleitet und ist äquivalent zur Klasse `RegularEvent` implementiert.

Durch die Trennung normaler von Ajax-Events kann der DialogController entscheiden, welche Aktionen auszuführen sind. So wird am Ende einer Kette von Ajax-Events keine neue Maske aufgerufen, sondern die benötigten Daten werden (durch die Verwendung der Datenflusssemantik) der jeweiligen Maske zur Verfügung gestellt.

5.2.1 Einlesen der Dialogflussspezifikation

Da die DFSL um ein Konstrukt erweitert wurde, mit dem Ajax-Events spezifiziert werden, muss auch das DCF so erweitert werden, dass es diese Events korrekt einliest und an den jeweiligen Dialogelementen bereitstellt.

Beim Einlesen der XML-basierten Spezifikation wird geprüft, ob `on-ajax` Events ausgelöst werden. Ist dies der Fall, wird statt eines regulären Events ein Ajax-Event in das Dialogflussmodell eingefügt. Anhand des Eventtypen prüft das DCF dann, ob es nach der Abarbeitung eines Dialogschrittes eine neue Seite anzeigen muss oder nur Daten zu aktualisieren hat.

5.2.2 Verarbeitung von Ajax-Events

Clientseitig muss dafür gesorgt werden, dass Ajax-Events per XMLHttpRequest-Objekt ausgelöst werden, damit sie wirklich asynchron ablaufen. Die Serverkomponente des DCF kann die Request-Art im Nachhinein nicht mehr ändern. Dazu wird später beschrieben, wie der Request mit Hilfe des integrierten Frameworks ausgelöst werden kann.

Das DCF dient nur dazu, Eventnamen zu verarbeiten und davon ausgehend Actions aufzurufen bzw. Masken zu definieren, die angezeigt werden sollen. Daher unterscheidet sich die eigentliche Verarbeitung von Ajax-Events nicht wesentlich von derjenigen, die bei regulären Events durchgeführt wird. Dadurch ist es leicht möglich, bereits bestehende Applikationen um Ajax-Kommunikationswege zu erweitern bzw. vorhandene Dialogflüsse zu ersetzen. Nur der `DCFNavigationHandler` musste angepasst werden[27], so dass im Falle eines Ajax-Events keine neue Maske angezeigt wird.

5.3 Integration von RichFaces

Um unabhängig von Browserimplementierungen des XMLHttpRequest-Objektes zu sein, wurden im vorangegangen Kapitel Frameworks vorgestellt, die unter anderem die API des Objektes kapseln. RichFaces soll dabei so eingebunden werden, dass es im DCF-Kontext verwendet werden kann. Durch die Verwendung von RichFaces ist es nicht mehr notwendig, den JSF-Lifecycle manuell anzupassen und eigene Ajax-fähige JSF-Komponenten zu entwickeln, wodurch bei der eigentlichen Anwendungsentwicklung eine große Zeitersparnis möglich ist.

[27] Durch die Einführung eines neuen Eventtypen mussten darüber hinaus einige Anpassungen am eigentlichen DialogController vorgenommen werden, die hier allerdings nicht näher beleuchtet werden sollen.

5.3.1 Anpassungen an der Anwendungskonfiguration

Damit RichFaces Ajax-Requests verarbeiten sowie die Response daraus erstellen kann, muss ein ServletFilter registriert werden. Im Falle eines Requests leitet dieser die korrekte Anfrage an das FacesServlet weiter, welches wiederum den DCF-NavigationHandler aufruft. Während der Response-Verarbeitung wird ein Filter, der den vorhandenen HTML-Response in XML umwandelt, angesprochen. Mittels eines Parser wird der so erzeugte Baum durchsucht, damit Referenzen auf CSS- und Javascript-Dateien eingefügt werden können, die für die korrekte Darstellung notwendig sind.

Listing 14: Anpassungen web.xml zur Integration von RichFaces

```
1  <!-- RichFaces Filter -->
2  <filter>
3      <display-name>RichFaces Filter</display-name>
4      <filter-name>richfaces</filter-name>
5      <filter-class>org.ajax4jsf.Filter</filter-class>
6  </filter>
7  <filter-mapping>
8      <filter-name>richfaces</filter-name>
9      <servlet-name>Faces Servlet</servlet-name>
10     <dispatcher>REQUEST</dispatcher>
11     <dispatcher>FORWARD</dispatcher>
12     <dispatcher>INCLUDE</dispatcher>
13 </filter-mapping>
```

Listing 14 zeigt die Anwendungskonfiguration der Demoanwendung. Bei jedem Zugriff auf das JSF Faces Servlet wird der Filter aufgerufen. Um korrekt zu arbeiten, muss der RichFaces Filter als erster Filter definiert werden, damit er vor der weiteren Verarbeitung des Requests durch die DCF spezifischen Filter ausgeführt wird.

5.3.2 Verwendung von RichFaces im Rahmen des DCF

Das RichFaces Framework bietet eine Reihe von Komponenten an, mit denen Ajax-basierte Nutzerinteraktionen realisiert werden können. Zwei der wichtigsten Vertreter sind die beiden Komponenten `a4j:commandButton` sowie `a4j:commandLink`. Diese können wie ihre JSF-Äquivalente verwendet werden, initiieren allerdings statt eines normalen einen asynchronen Request per XMLHttpRequest-Objekt.

Listing 15: Verwendung von a4j:commandButton sowie a4j:commandLink

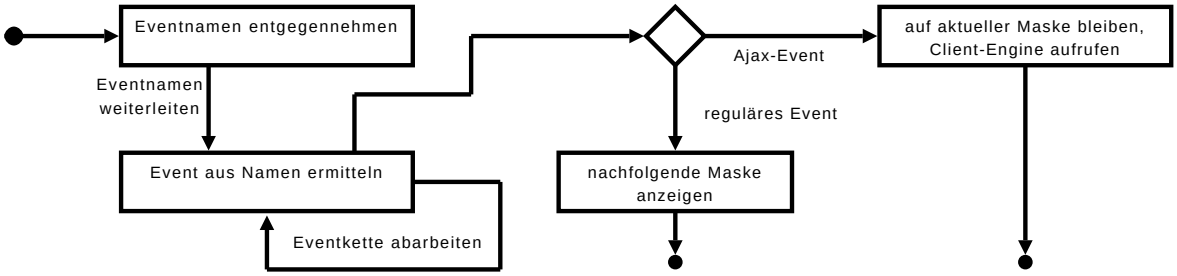
```
1  <h:form>
2      <a4j:commandButton action="eventName" value="invoke Event"/>
3      <a4j:commandLink action="eventName" value="invoke Event"/>
4  </h:form>
```

Die Verwendung der beiden Komponenten ist in Listing 15 dargestellt. Den Komponenten wird ein Parameter `action` hinzugefügt, der den Namen des auszuführenden DCF Events beinhaltet und vom NavigationHandler entgegen genommen wird. Mit der Verwendung dieser beiden ist es möglich, Ajax-Kommunikation zu initiieren, ohne dass clientseitiger Javascript Code geschrieben werden muss. Es ist wie bei den originalen JSF-Komponenten auch möglich, direkt Bean-Actions anzusteuern, was allerdings außerhalb der Spezifikation des DCF-Lebenszyklus' liegen würde und deshalb nicht verwendet werden sollte.

5.4 Zusammenfassung

In Abbildung 13 ist zusammenfassend dargelegt, wie Events im DCF verarbeitet werden. Erst am Ende der Abarbeitung wird geprüft, ob tatsächlich eine neue Seite angezeigt wird oder auf der aktuellen Seite verblieben werden soll.

Abbildung 13: Ablauf der Event-Verarbeitung des DCF



Die Hauptschritte der Abarbeitung sollen hier nocheinmal kurz zusammengefasst werden:

- Eventnamen entgegennehmen / weiterleiten: Der `DCFNavigationHandler` ermittelt aus dem aktuellen Request das Event, welches aufgerufen werden soll und gibt dieses an das DCF weiter.
- Eventkette abarbeiten: Das DCF selbst sorgt dann dafür, dass das jeweilige Event korrekt verarbeitet wird. Dazu wird das Event aus dem Dialogflussmodell extrahiert und eventuell Actions aufgerufen bzw. Compounds eingeschachtelt. Ist die Eventkette vollständig abgearbeitet, das heißt also das nächste aufzurufende Dialogelement ist eine Maske, wird wieder der DCFNavigationHandler aufgerufen.
- Reguläres Event: Im Falle eines regulären Events wird auf die mit der Maske verknüpften Seite weitergeleitet und diese angezeigt.
- Ajax-Event: Handelt es sich um ein Ajax-Event, wird keine neue Seite angezeigt. Stattdessen wird die clientseitige Javascript-Engine aufgerufen (bereitgestellt durch RichFaces). Diese sorgt dafür, dass notwendige Änderungen an der aktuellen Seite vorgenommen worden; d.h. Komponenten, die aktualisiert werden müssen (durch die Angabe des `reRender`-Attributes), werden neu gerendert. Dadurch werden Änderungen durch die Geschäftslogik visualisert.

Da sich an der internen Abarbeitung eines Events keine Änderungen ergeben haben, können bereits bestehende Dialogflussspezifikationen auch mit der um Ajax-Fähigkeiten erweiterten DCF-Variante verwendet werden.

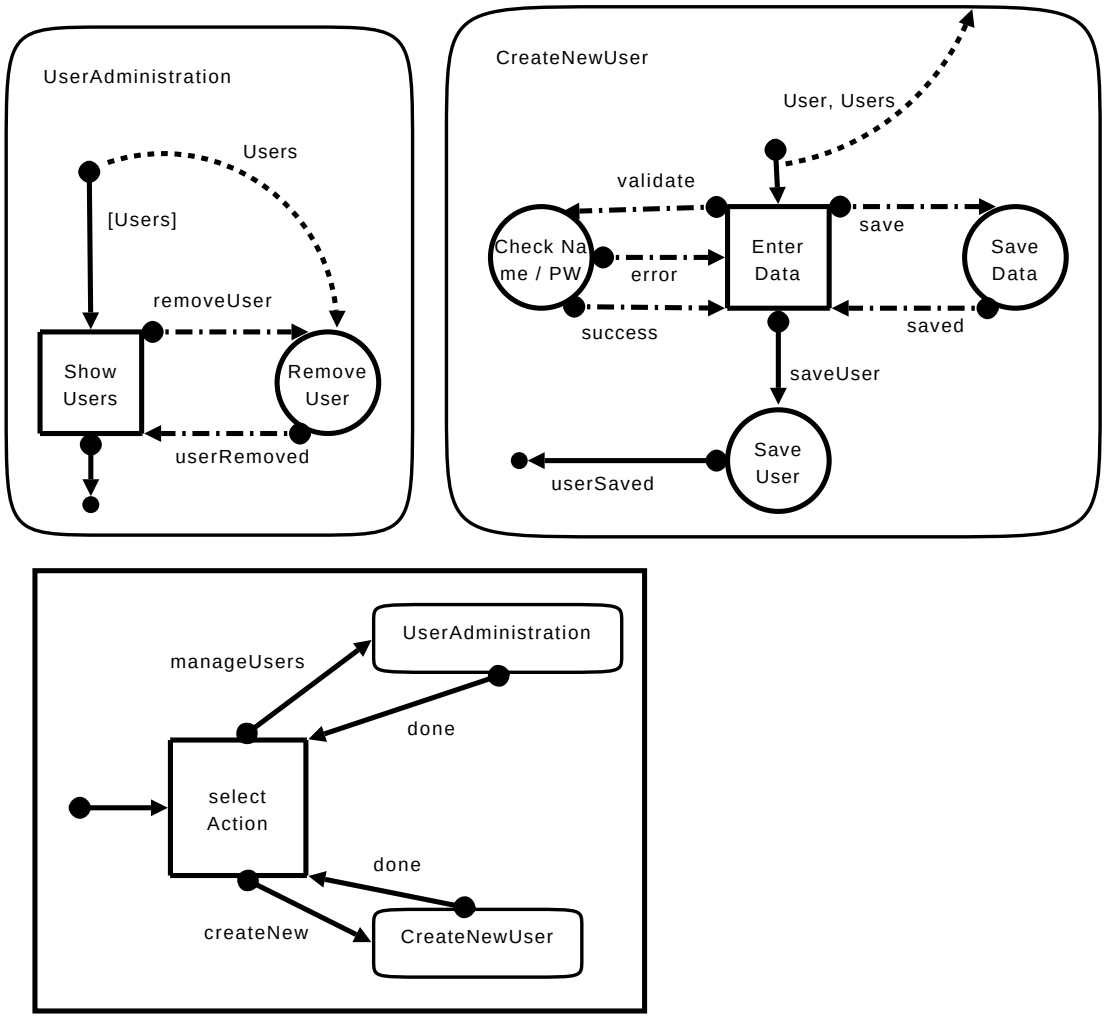
6 Lösungen durch die Ajax-Komponente

Im Rahmen der Arbeit wurde eine Anwendung entwickelt um zu zeigen, wie Ajax-basierte Nutzerinteraktionen unter Verwendung des DCF eingesetzt werden können, um komplexere Benutzeroberflächen zu erstellen. Diese wird in diesem Kapitel kurz vorgestellt, womit die Integration der entwickelten Ajax-Komponente in DCF-basierende Anwendungen beginnend bei der Dialogflussspezifikation bis hin zur Implementierung verdeutlicht werden soll.

6.1 Dialogflussspezifikation der Anwendung

Die Anwendung besteht aus den Modulen UserAdministration und CreateNewUser, in denen Ajax-fähige Komponenten eingesetzt werden. Neue Benutzer werden mittels Ajax-Kommunikation angelegt und gelöscht. Abbildung 14 zeigt die Dialogflussspezifikation der Anwendung.

Abbildung 14: Dialogflussspezifikation der Demoanwendung



6.2 Ajax-basierte Nutzerinteraktionen der Anwendung

Wie bereits in Kapitel 5.3.2 auf Seite 58 erläutert, stellt RichFaces Komponenten, mit denen asynchrone Requests abgesendet werden können, bereit. Dabei ist zu beachten, dass diese beim Absenden der Requests den Lebenszyklus des DCF nicht verlassen. Es bietet sich daher an, vorhandene JSF-Komponenten mit Hilfe der Ajax4JSF-Taglib zu erweitern.

Mit Hilfe der erstellten Anwendung können neue Arten der Nutzerinteraktion, die bisher nur in Desktopanwendungen möglich waren (siehe 2.3 auf Seite 16), umgesetzt werden, was im Folgenden vorgestellt werden soll:

- Automatisches Speichern: Die Komponente `pull` des RichFaces Frameworks ermöglicht das periodische Absenden asynchroner Requests. Der Komponente wird der DCF-Eventname und die neu zu rendernden JSF-Komponenten angegeben. Dadurch ist es möglich, in einem vorher definierten (oder vom Benutzer festzulegenden) Intervall ein Event aufzurufen, welches dafür sorgt, die aktuell vorhandenen Formulardaten temporär zu speichern.
- Automatische Validierung der Eingabevorschläge: In der Maske `EnterData` wird schon bei der Eingabe der Daten eines neu zu erstellenden Benutzers geprüft, ob ein Benutzer mit diesem Namen bereits existiert. Dies wird durch das `support`-Tag ermöglicht, welches auf das Javascript-Event `onChange` reagiert. Dadurch wird ein DCF-Event aufgerufen, welches eine Action instanziiert, die den Benutzernamen prüft. Schlägt die Prüfung fehl, wird eine Fehlermeldung bereits beim Anlegen des Benutzers ausgegeben. Ohne Ajax könnte die Prüfung erst erfolgen, wenn bereits alle Daten eingegeben sind, wodurch der Benutzer erst viel später Feedback erhält.
- Geschwindigkeit: Durch die automatische Validierung des Benutzernamens muss im Falle eines Fehlers auch nicht die komplette Seite zusammen mit der Fehlermeldung neu geladen werden. Es muss nur noch die eigentliche Meldung vom Server zum Client übertragen werden, was einerseits die Datenmenge und daraus folgend die benötigte Übertragungszeit verringert.

7 Verwandte Arbeiten

Im Rahmen dieser Arbeit wurde eine Komponente für das DCF entwickelt, mit der Ajax-basierte Nutzerinteraktionen verarbeitet werden können. Zur Verdeutlichung weiterer ähnlich gelagerter Ansätze sollen in diesem Kapitel Arbeiten vorgestellt werden, die sich mit dem Einsatz von Ajax in JSF-basierten Anwendungen beschäftigen. Es werden Frameworks vorgestellt, die es wie das DCF erlauben Dialogflüsse zu spezifizieren. Auch diese implementieren mittlerweile Ajax, so dass sie sich als Inspiration für weitere Untersuchungen eignen.

Durch den Einsatz des DCF ist es mit der Verwendung der DFN möglich, Dialogflüsse von Webanwendungen anhand von Diagrammen und damit auf konzeptueller Ebene zu spezifizieren. Da Dialogflüsse nur ein Teil von Webanwendungen sind, sollen hier auch weitere Modellierungsansätze vorgestellt werden, die einen theoretischen Überbau zur Entwicklung von Webanwendungen bieten.

7.1 Ajax in JSF-basierten Anwendungen

7.1.1 Vergleich von Ajax-Frameworks

Im Rahmen dieser Arbeit konnten nur einige wenige Ajax-Frameworks vorgestellt werden. Es lohnt sich allerdings den wachsenden Markt im Auge zu behalten, da es noch sehr viele andere Frameworks gibt, die eventuell benötigte Features implementieren.

In [Lee08] wird eine Reihe verschiedener JSF-Ajax-Frameworks miteinander verglichen und nach Kriterien wie Reichhaltigkeit, angebotenten UI-Elementen, etc. kategorisiert. Außerdem wird verglichen, welche Komponenten die Frameworks anbieten sowie auf Probleme bei der Integration von Ajax und JSF eingegangen.

Da die DCF-Erweiterung so entwickelt wurde, dass das Ajax-Framework relativ leicht ersetzt werden kann, bietet sich eine größere Übersicht an, so dass es möglich ist, ein Framework zu verwenden, was am besten zur Natur der jeweiligen Anwendung passt.

7.1.2 JSF 2.0 und Ajax

Derzeit ist noch die Verwendung eines zusätzlichen Frameworks mit eigenen an Ajax angepassten Komponenten notwendig, um Ajax-Kommunikation zu ermöglichen. Dazu muss unter Umständen tief in den JSF-Lifecycle eingegriffen werden, wodurch (wie zum Beispiel bei ICEFaces) die Integrationsmöglichkeiten mit anderen Komponentenframeworks ausgeschlossen oder verringert werden können.

Dem soll im Rahmen der JSF Version 2.0 Abhilfe geschaffen werden [BK08]. Es ist geplant, eine Ajax-Engine zu integrieren, mit der es möglich ist, einen Ajax-Request

an den Server zu senden sowie einen Partial Page Respone zu erhalten und mit diesem die Elemente des DOM-Baums zu aktualisieren. Damit orientiert sich das Vorgehen am J4Fry Framework, welches keine neuen UI-Elemente anbietet, sondern vorhandene um Ajax erweitert.

Weiterhin angedacht ist die einfachere Unterscheidung zwischen Ajax- und normalen Requests sowie das daraus resultierende Abbrechen des Lifecycles mittels eines PhaseListeners. Dies würde den Einsatz reiner Ajax-Frameworks überflüssig machen, womit die Komponentenentwicklung näher am Standard wäre und weitere Fehlerquellen ausgeschlossen werden können.

Die Spezifikation von JSF 2.0 befindet sich momentan im Status Public Review. Erfahrungen mit der Entwicklung vorheriger JSF-Spezifikationen haben gezeigt, dass sie etwa ein Jahr nach dieser Phase einen finalen Status erreicht.

7.2 Modellierung von Webanwendungen

Im Bereich des Softwareengineering hat sich zur Modellierung von Anwendungen die Unified Modeling Language (UML) im Rahmen der Erstellung objektorientierter Systeme durchgesetzt. Diese genügt allerdings in einigen Kernpunkten den spezifischen Anforderungen zur Modellierung von Webanwendungen nicht:

- Es können keine Links innerhalb der Anwendung modelliert werden.
- Die Oberflächen von Webanwendungen lassen sich nicht modellieren.

Aus diesem Grund wurden an das Web Engineering angepasste Modellierungswerkzeuge entwickelt, von denen zwei Vertreter vorgestellt werden sollen. Im Anschluss daran wird gezeigt, wie die neuen Interaktionsmöglichkeiten, die sich durch die Verwendung Ajax-basierter Kommunikation ergeben, in die Modellierungsansätze integriert werden können.

7.2.1 UML-based Web Engineering

Mittels des UML-based Web Engineering (UWE) Ansatzes ist es möglich, Webanwendungen modellgetrieben zu entwickeln [Koc00]. Da die Notation auf UML basiert und nur um ein UML-Profil erweitert wurde, können bestehende Modellierungswerkzeuge weiterhin verwendet werden. Im Rahmen der UWE-Modellierung werden dabei Stereotypen für die spezifischen Bestandteile von Webanwendungen, die in klassischen Anwendungen nicht existieren, verwendet.

Bei der Verwendung des UWE-Ansatzes werden die folgenden 5 Modelle unterschieden:

- Use Case Model: Hierbei wird die eigentliche Funktionalität einer Anwendung sowie die Interaktionen zwischen den verschiedenen Akteuren und dem Sy-

stem an sich dargestellt. UWE erweiterteitert das bekannte UML Use Case Model durch den Stereotypen `navigation`. Damit wird ausgedrückt, dass ein Akteur zum jeweiligen Anwendungsfall navigieren kann.

- Conceptual Model: Basierend auf dem UML Klassendiagramm werden die Klassen und Beziehungen zwischen diesen modelliert.
- Navigation Model: Dies ist ein UWE spezifischer Stereotype, mit dem die eigentliche Navigationsstruktur der Anwendung, basierend auf dem Conceptual Model, dargestellt wird. Das Navigation Model unterteilt sich in das Navigation Space Model zur Darstellung der eigentlichen Navigationsmöglichkeiten sowie dem Navigation Structure Model zur Darstellung unterschiedlicher Zugriffsstrukturen, wie z.B. einem Index, Menu oder einer Guided Tour.
- Presentation Model: Der strukturelle Aufbau der Benutzeroberflächen wird mittels eines Abstract User Interfaces modelliert.
- Process Model: Die im Navigation Model eingeführten Prozesse werden im Rahmen des Process Models modelliert. Dazu werden die Verhaltensdiagramme der UML (u.a. Sequenzdiagramme, Aktivitätsdiagramme, etc.) verwendet.

Es ist möglich, aus den UWE-Modellen automatisch JSPs und Java Beans für die Präsentation sowie das Datenmodell zu erstellen. Die Geschäftslogik, die mittels der Verhaltensdiagramme modelliert wurde, wird ergänzend dazu zur Laufzeit interpretiert und ausgeführt.

Durch die Erweiterung von UWE mittels des Werkzeuges UWE4JSF lassen sich automatisch JSF-basierende Anwendungen erzeugen, die augrund der verwendeten Komponentenbibliothek auch Ajax-Kommunikation einsetzen können [Kro08].

7.2.2 Web Modeling Language

Eine weitere Methode, Webanwendungen anhand von Modellen zu entwickeln, ist die Verwendung der Web Modeling Language (WebML). Diese UML-ähnliche Notation dient der Beschreibung einer Webseite anhand der unabhängige Dimensionen Daten, Seiten, Links zwischen Seiten, graphisches Erscheinungsbild sowie Individualisierung [CFB00]. Dazu werden im Rahmen des WebML-Prozesses folgende Modelle verwendet:

- Structural Model: Hierbei werden anhand von Entities und Relationships die Daten der Anwendung spezifiziert. Die verwendeten Diagramme sind kompatibel zu E/R-Modellen sowie UML-Klassendiagrammen.
- Hypertext Model: Die Beschreibung der Hypertext-Topologie einer Webseite unterteilt sich in das Composition Model, worin festgelegt wird, wie Links dargestellt werden sollen (u.a. Index, Filter) sowie das Navigation Model, in welchem angegeben wird, wie die eigentlichen Seiten miteinander verknüpft sind.

- Presentation Model: Unabhängig von der Anzeige auf verschiedenen Endgeräten wird hier das eigentliche graphische Erscheinungsbild sowie die Anordnung der Oberflächenelemente beschrieben.
- Personalization Model: Individualisierte Strukturen und Inhalte sowie Zugriffskontrollen lassen sich durch das Einführen von Personen und Rollen darstellen.

Aus den graphischen Modellen erstellt WebML XML-Dokumente, die von verschiedenen Frameworks eingelesen werden und aus denen Anwendungsgerüste erstellt werden können.

7.2.3 Modellierung von Ajax-Kommunikation

Sowohl UWE als auch WebML ermöglichen die Modellierung klassischer Webanwendungen, bieten allerdings keine Unterstützung für Ajax-basierte Anwendungen mit der Möglichkeit asynchrone Requests zu initiieren und nur Daten statt kompletter Seiten zu senden. Um diese zu integrieren, werden Erweiterungen benötigt, die im Folgenden vorgestellt werden.

In [BPLSF08] wird ein Ansatz vorgestellt, bei dem aus einer Modellierung in der Business Process Management Notation (BPMN, [Whi04]) automatisch WebML-Modelle erstellt werden, die Ajax-Interaktionen spezifizieren. Die BPMN wird dabei so interpretiert, dass Daten, Geschäftslogik, Kommunikation und Präsentation modelliert werden können.

Das Datenmodell einer Anwendung wird durch die Datenobjekte der BPMN dargestellt. Zur Spezifikation des Speicherorts werden sie einem der Bereiche (Swimlane) Client oder Server zugeordnet. Der Zustand (State) des Datenobjektes wird als Persistenzverhalten (persistent oder flüchtig) interpretiert. Durch Verknüpfungen der BPMN-Flussobjekte wird die Geschäftslogik spezifiziert.

Die Darstellung der verschiedenen Kommunikationsmöglichkeiten erfolgt durch die Verwendung der BPMN-Sequenzflusspfeile. Mittels Annotierungen lassen sich sowohl synchrone (s) als auch asynchrone (a) Interaktionen modellieren. Die Richtung des Pfeils gibt dabei vor, ob die Kommunikation vom Client oder vom Server ausgeht, wodurch sich auch Ajax-Push modellieren lässt.

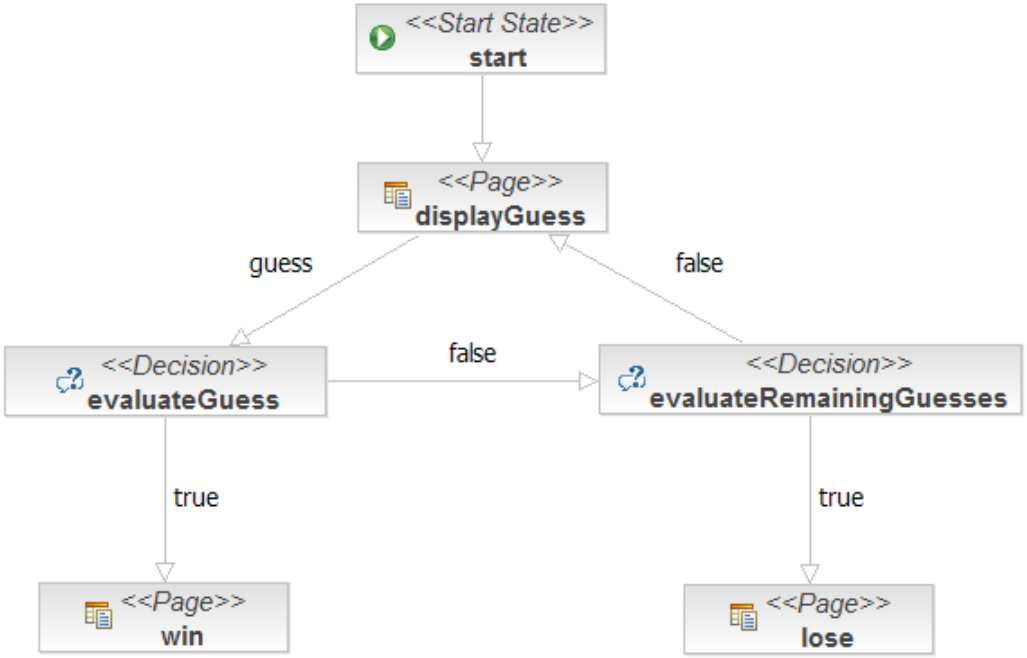
Die Diagramme der WebML wurden so erweitert, dass sich auch nebenläufige Abarbeitungen, wie sie durch asynchrone Requests erforderlich werden, darstellen lassen. Da bei der Verwendung von Ajax-Kommunikation keine neue Seite geladen wird, musste auch das Hypertext Model angepasst werden. Eine einzelne Seite kann nun auch verschiedene Unterseiten (activity subpage) enthalten.

7.3 Verwandte Frameworks

7.3.1 Seam

Das JBoss Seam Framework dient der Erstellung webbasierter Geschäftsanwendungen auf Basis der J2EE Technologie und JSF. Es dient der Verbindung der verschiedenen Schichten einer Anwendung miteinander, wodurch technische Verknüpfungen im Code, die nicht zur eigentlichen Geschäftslogik gehören, vermieden werden.

Abbildung 15: Seam Pageflows



Zur Spezifikation komplexerer Navigationsflüsse verwendet Seam sogenannte Pageflows. Wie in Abbildung 15 zu sehen, bestehen diese aus Page-Elementen, die zur Anzeige einer bestimmten JSF-Seite dienen sowie Decision-Elementen, welche die weitere Verzweigung spezifizieren. Die Übergänge zwischen den Elementen nennen sich Transitions und können mit einer JSF-Action verknüpft werden. Wie auch im DCF können Pageflows durch die Verwendung von Unterprozessen verschachtelt werden, so dass komplexere Dialogflüsse möglich sind.

Seam ermöglicht die Verwendung Ajax-basierter Nutzerinteraktionen durch die Integration eines JSF-Ajax-Frameworks wie z.B. RichFaces. Dabei ist ein nebenläufiger Zugriff auf den Session- sowie den Anwendungskontext erlaubt. Außerdem werden Ajax-Anfragen zwischengespeichert, so dass gleiche Anfragen serverseitig weniger Rechenaufwand verursachen.

7.3.2 Spring Web Flow

Mittels Spring Web Flow (SWF), einem Teilprojekt des Spring Frameworks[28], ist es möglich den Dialogfluss einer Webanwendung zu spezifizieren. Dazu werden wie bei der Verwendung des DCF XML-Dokumente genutzt, die im Rahmen des SWF mittels der Flow Definition Language eine Menge von Zuständen (States) definieren. In jedem State wird eine Aktion ausgelöst: Dies kann entweder das Anzeigen einer Seite oder aber das Ausführen einer Action zum Abarbeiten der Geschäftslogik sein.

Listing 16: Spezifikation eines Dialogflusses mittels Spring Web Flow

```
1  <flow>
2   <view-state id="enterBookingDetails">
3    <transition on="submit" to="reviewBooking" />
4   </view-state>
5   <view-state id="reviewBooking">
6    <transition on="confirm" to="bookingConfirmed" />
7    <transition on="revise" to="enterBookingDetails" />
8    <transition on="cancel" to="bookingCancelled">
9     <render fragments="hotels:searchResultsFragment" />
10   </transition>
11  </view-state>
12  <end-state id="bookingConfirmed" />
13  <end-state id="bookingCancelled" />
14 </flow>
```

Um zwischen den einzelnen States zu wechseln, werden Transitions verwendet, welche von einem Event ausgelöst werden. Abbildung 16 zeigt einen vereinfachten Dialogfluss zur Buchung eines Hotels. Um den Dialogfluss einer Anwendung auf mehrere Module aufzuteilen, bietet SWF sogenannte Subflows, die in andere Flows eingeschachtelt werden können.

Wird Spring zusammen mit JSF verwendet, bietet es Ajax-fähige Komponenten an, mit denen asynchrone Requests initiiert werden können. Diese erzeugen ein Event, welches wie normale Events durch eine Transition verarbeitet wird. Um Spring anzugeben, dass nicht die komplette Seite neu geladen werden soll, werden innerhalb der Transition noch diejenigen Komponenten angegeben, die nach Abarbeitung des Requests neu gerendert werden. Dies ist in Listing 16 in Zeile 9 durch die Angabe des Tags `render` verdeutlicht. Falls Javascript clientseitig nicht verfügbar ist, wird ein normaler Request-Response-Zyklus durchlaufen und die komplette Seite neu gerendert.

7.4 Zusammenfassung

Die modellbasierte Entwicklung von Webanwendungen, die auf Techniken des klassischen Softwareengineering zurückgreift, wird auch in den nächten Jahren noch Gegenstand weiterer Forschungen sein. Mit UWE sowie WebML wurden zwei Ansätze vorgestellt. Es ist denkbar, dass Navigation Space Model (UWE) bzw. das Navigation Model (WebML) durch die Diagramme der DFN zu ersetzen, so dass komplexere Dialogflüsse möglich sind.

[28] http://www.springsource.org/

Aus technischer Sicht erscheint das Seam Framework mit seinen nebenläufigen Zugriffsmöglichkeiten als interessanter Untersuchungsgegenstand für Ajax-basierte Nutzerinteraktionen.

8 Fazit und weitergehende Arbeiten

Im Rahmen der Arbeit wurden Konzepte vorgestellt, mit denen sich die Interaktionsmöglichkeiten von Webanwendungen verbessern lassen. Dazu wurden asynchrone Kommunikationsmöglichkeiten vorgestellt, welche mittels Ajax realisierbar sind. Das Dialog Control Framework wurde dabei um eine Ajax-Komponente erweitert, so dass asynchrone Dialogflüsse schon im Rahmen der Spezifikation visualisiert werden können.

Ein weiterer Teil der Arbeit war die Integration des DCF in das Java Server Faces Framework, in welches sich das DCF als NavigationHandler einfügen lässt. Dadurch wird eine noch bessere Trennung der drei Schichten einer Anwendung, die auf dem MVC-Pattern basiert, erreicht.

Im folgenden Kapitel soll die Arbeit noch einmal kurz zusammengefasst und aufgetretene Probleme und Besonderheiten diskutiert sowie ein Ausblick auf weitere Arbeitsfelder gegeben werden.

8.1 Zusammenfassung

Die Ajax-Komponente für das DCF, welche in dieser Arbeit entwickelt wurde, unterstützt Benutzer bei der Spezifikation und Implementierung Ajax-basierter Nutzerinteraktion im DCF. Durch die Erweiterung der DFN und DFSL kann schon während des Anwendungsdesigns bestimmt werden, wann asynchrone Requests abgeschickt werden sollen, was eine genauere Spezifikation erlaubt. Bisher war das DCF auf synchrone Requests und damit das Seitenparadigma beschränkt. Mit der Ajax-Komponente können nun reichhaltigere und interaktivere Benutzeroberflächen erstellt werden, die die Einschränkungen der zugrunde liegenden Technologien verdecken.

Es ist leicht möglich RichFaces, welches Ajax-fähige JSF-Komponenten bereitstellt, zu verwenden. Dies hat den Vorteil, dass einerseits keine eigenen Komponenten, die speziell an das DCF angepasst sind, entwickelt werden müssen. Andererseits kann auf einer soliden Javascript-Engine aufgebaut werden, die unabhängig vom DCF weiterentwickelt und verbessert wird, was zu einer Verringerung möglicher Fehler führt.

8.2 Diskussion

Auch wenn die Ajax-Komponente Verbesserungen in den Interaktionsmöglichkeiten mit sich bringt, entstehen dadurch neue Probleme bei der Verwendung des DCF. So ist besonders beim Einsatz vorgefertigter Framework-Element darauf zu achten, dass der eigentliche Dialogfluss des DCF nicht umgangen wird und stattdessen statische Navigationswege verwendet werden. Hier sei als Beispiel die SuggestionBox von RichFaces genannt, der mittels einer `suggestionAction` definiert werden kann,

welche Aktionen sie durchführen soll. Diese Action muss an eine Bean-Methode gebunden sein und eine Liste möglicher Werte zurückgeben. Beim Aufruf wird allerdings der JSF NavigationHandler nicht angesteuert, so dass das DCF gar nicht zum Tragen kommt. Hier ist in zukünftigen Versionen noch zu untersuchen, wie die Komponenten des Frameworks angepasst werden müssen.

Der aktuellen Implementierung der Ajax-Komponente müssen bei einem Ajax-Request zwei Ajax-Events angegeben werden: Das erste von der Maske aus, die den eigentlichen Request sendet und das zweite, wenn eine Action wieder zurück auf die Maske zeigt. Es wäre denkbar, dies so anzupassen, dass nur noch die Maske ein Ajax-Event auslösen muss und das DCF dann selbst entscheidet, dass keine neue Maske angezeigt wird, so dass von der Action ein reguläres Event ausgehen kann. Dadurch können die Regeln für Ajax-Events konsequenter umgesetzt werden, so dass sie wirklich nur von Masken ausgehen dürfen, was das Validieren der Dialogflussspezifikation vereinfachen würden.

Durch die praktische Nebenläufigkeit von Ajax-Requests ergeben sich einige Fragen hinsichtlich des Framework-Verhaltens. Ungeklärt ist, wie das Framework sich verhalten soll, wenn ein Ajax-Request gestellt wird, dessen Abarbeitung auf dem Server sehr lange dauert. In diesem Fall kann es passieren, dass durch weitere Aktionen des Benutzers bereits andere Compounds geladen wurden; beim Abarbeiten des Ajax-Events kann es daher zu Inkonsistenzen im DCF kommen, die aufgelöst werden müssen.

8.3 Ausblick

Neben den vorgestellten und bearbeiteten Themenfelder konnten während der Bearbeitung weitere Ansätze ermittelt werden, die allerdings über den Rahmen der Arbeit hinausgehen. Zur weiteren Verbesserung des DCF könnten diese Gegenstand zukünftiger Untersuchungen sein.

8.3.1 Geräteunabhängigkeit, Erweiterung graphischer Benutzeroberflächen

Ein Kernbestandteil des DCF ist die Möglichkeit, abweichende Dialogflüsse für verschiedene Endgeräte zu erstellen. Dies ist besonders dann nötig, wenn die zur Verfügung stehende Darstellungsgröße der einzelnen Geräte stark voneinander abweicht; als Beispiel sei hier das Verwenden einer Anwendung mittels Handy sowie PC genannt.

Insbesondere bei der Umsetzung der Benutzeroberfläche mittels der Wireless Markup Language (WML) kommt es immer noch zu sehr unterschiedlichen Ausgaben auf den verschiedenen Endgerätetypen. Erschwerend kommt hinzu, dass das Javascript-Gegenstück WMLScript nicht die komplette Sprache abdeckt [SGBF01]. Es ist daher möglich, dass nicht alle Endgeräte die neuen Interaktionsmöglichkeiten verwenden können, was insbesondere bei der Verwendung komplexerer Frameworks zum Tragen kommt.

Auf der anderen Seite ist es denkbar, bei Endgeräten, die eine reichhalte graphische Benutzeroberfläche darstellen können, ein clientseitiges Javascript Framework zu integrieren, welches intuitivere und desktopähnlichere Interaktionsmöglichkeiten anbietet. Da das erweiterte DCF eigenständig ermittelt, ob ein Ajax- oder ein normales Event ausgelöst werden soll, muss bei der Verwendung eines clientseitigen Frameworks nur darauf geachtet werden, dass ein asynchroner Request abgesendet wird, um innerhalb der Ajax-Kommunikation zu bleiben.

8.3.2 Integration weiterer Dialogflussmöglichkeiten

Mit der vorgestellten Implementierung ist es möglich, reguläre Dialogflüsse zwischen einzelnen Dialogelementen des DCF durch Ajax-Dialogflüsse zu ersetzen. Das DCF bietet daneben mit Common- sowie mit Compound-Events Möglichkeiten an, mit denen erweiterte Dialogflüsse spezifiziert werden können. Compound-Events können dabei von allen Dialogelementen eines Compounds ausgelöst werden, so dass es nicht notwendig ist, das Event von jedem Element aus zu verlinken. Mittels Common-Events ist es dagegen möglich, Dialogflüsse anwendungsweit zu spezifizieren, so das zum Beispiel Module immer aufgerufen werden können.

Da die Ajax-Komponente so entwickelt wurde, dass Ajax-Events unter Beachtung der im Kapitel 4 dargelegten Regeln äquivalent zu regulären Events eingesetzt werden können, ist es auch möglich, diese Anwendung auf Compound- und Common-Events zu erweitern. Dazu kann die vorhandene Implementierung als Grundlage verwendet werden, um die DFN/DFSL zu erweitern und das Einlesen der Dialogflussspezifikation zu ermöglichen.

Wie in Kapitel 3 vorgestellt, ist auch eine Push-basierte Kommunikation mittels Ajax möglich. Diese ist zum aktuellen Stand noch gar nicht von den Möglichkeiten des DCF abgedeckt; denkbar wäre ein neues Event, was vom Server ausgelöst wird und den Status des Clients ändert. Da beim Eintreten eines Server-Events keine neue Maske angezeigt wird, kann hierzu die clientseitige Ajax-Komponente verwendet werden. Ebenfalls keine eigenständige Repräsentation im DCF haben periodische Events. Es ist denkbar, dass eine Anwendung in einem bestimmten Intervall Daten auf dem Server zwischenspeichert. Die eigentliche Speicherung kann zwar modelliert werden, allerdings nicht, dass diese periodisch erfolgt.

8.3.3 Weitere Verzahnung mit JSF

Zum einfachen Zugriff auf Beans müssen diese sowohl in der `faces-config` des JSF Frameworks als auch in der Spezifikation des Dialogflusses angegeben werden. Durch diese Redundanz kann es mitunter zu Fehlern kommen, da der Name der Bean dem des Datums entsprechen muss. Um dies zu vermeiden, wäre es denkbar, die Integration mit JSF so zu erweitern, dass die Konfiguration der Managed Beans automatisch aus der Spezifikation des DCF erfolgt. Dazu muss dieser neben dem Namen nur noch die Java-Klasse übergeben werden, welche die Bean repräsentiert.

8.3.4 Automatischer Aufbau Ajax-basierter Oberflächen

Mit dem UI-Framework Cepheus können graphische Benutzeroberflächen aus der fachlichen Beschreibung einer Anwendung erstellt werden [BBB+08]. Dieses generiert zur Steuerung des Dialogflusses automatisch DFSL-Dokumente für das DCF. Es ist daher einerseits denkbar, dass Cepheut automatisch Ajax-fähige Interaktionsmöglichkeiten und entsprechende Dialogflüsse aufbaut. Eine Möglichkeit dazu wäre zu prüfen, ob verschiedene Masken zu einer zusammengefasst und deren unterschiedliche Darstellungszustände mittels Ajax getrennt werden können. Andererseits kann die fachliche Beschreibung auch so erweitert werden, dass asynchrone Dialogflüsse manuell markiert werden können.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet.

Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 01.01.2008

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literatur

[BBB+08] BATSUKH, Nomunbilegt; BOOK, Matthias; BRÜCKMANN, Tobias; GEIER, Jens; GRUHN, Volker; KLEBECK, Alex; SCHÄFER, Clemens: Automatic Generation of Ruler-Based User Interfaces of Web Applications. In: *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*. Washington, DC, USA : IEEE Computer Society, 2008, S. 103–108

[BG04] BOOK, Matthias; GRUHN, Volker: Modeling Web-Based Dialog Flows for Automatic Dialog Control. In: *ASE '04: Proceedings of the 19th IEEE international conference on Automated software engineering*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7695–2131–2, S. 100–109

[BHH+04] BYRNE, Steve; HORS, Arnaud L.; HÉGARET, Philippe L.; CHAMPION, Mike; NICOL, Gavin; ROBIE, Jonathan; WOOD, Lauren: Document Object Model (DOM) Level 3 Core Specification / W3C. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 (01.01.2008)

[BK08] BURNS, Ed; KITAIN, Roger: JavaServer Faces Specification Version 2.0 / Sun Microsystems, Inc. 2008. – Forschungsbericht. – 13–1–13–12 S

[BMD07] BOZDAG, Engin; MESBAH, Ali; VAN DEURSEN, Arie: A Comparison of Push and Pull Techniques for Ajax. In: *Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE 07)*, IEEE Computer Society, Oktober 2007. – ISBN 978–1–4244–1450–5, S. 15–22

[Bor86] BORNING, A. H.: Classes versus prototypes in object-oriented languages. In: *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1986. – ISBN 0–8186–4743–4, S. 36–40

[Bos04] BOSCH, Andy: *Java Server Faces. Das Standard-Framework zum Aufbau webbasierter Anwendungen*. München : Addison Wesley, 2004. – ISBN 978–3–8273–2127–5

[BPLSF08] BRAMBILLA, Marco; PRECIADO, Juan C.; LINAJE, Marino; SANCHEZ-FIGUEROA, Fernando: Business Process-Based Conceptual Design of Rich Internet Applications. In: *ICWE '08: Proceedings of the 2008 Eighth International Conference on Web Engineering*. Washington, DC, USA : IEEE Computer Society, 2008. – ISBN 978–0–7695–3261–5, S. 155–161

[CBS08] Kap. 8 Die Befehle Zurück, Aktualisieren und Rückgängig implementieren In: CRANE, Dave; BIBEAULT, Bear; SONNEVELD, Jord: *Ajax in*

practice. Addison Wesley, 2008, S. 301–340. – ISBN 978–3–8273–2596–9

[CFB00] CERI, Stefano; FRATERNALI, Piero; BONGIO, Aldo: Web Modeling Language (WebML): a modeling language for designing Web sites. In: *Comput. Netw.* 33 (2000), Nr. 1-6, S. 137–157. – ISSN 1389–1286

[Con99] CONALLEN, Jim: Modeling Web application architectures with UML. In: *Commun. ACM* 42 (1999), Nr. 10, S. 63–70. – ISSN 0001–0782

[CR03] COOPER, Alan; REIMANN, Robert M.: *About Face 2.0: The Essentials of Interaction Design*. Indianapolis, Indiana, USA : Wiley Oublishing, 2003. – ISBN 978–0–7645–2641–1

[Cro06] CROCKFORD, Douglas. *The application/json Media Type for JavaScript Object Notation (JSON)*. http://www.ietf.org/rfc/rfc4627.txt (01.01.2008). Juli 2006

[DLWZ03] Kap. 1.3. Webbasierte Softwaresysteme In: DUMKE, Reiner; LOTHER, Mathias; WILLE, Cornelius; ZBROG, Fritz: *Web Engineering*. Pearson Studium, 2003, S. 35

[DMC92] DONY, Christophe; MALENFANT, Jacques; COINTE, Pierre: Prototype-based languages: from a new taxonomy to constructive proposals and their validation. In: *SIGPLAN Not.* 27 (1992), Nr. 10, S. 201–217. – ISSN 0362–1340

[Gam06] Kap. 2 JavaScript und DOM In: GAMPERL, Johannes: *AJAX Web 2.0 in der Praxis*. Galileo Computing, 2006, S. 17–64

[Gar05] GARRET, Jesse J. *Ajax: A New Approach to Web Applications*. http://www.adaptivepath.com/ideas/essays/archives/000385.php (01.01.2008). Februar 2005

[Kes08] VAN KESTEREN, Anne: The XMLHttpRequest Object / W3C. 2008. – a WD in Last Call. http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/ (01.01.2008)

[KMK99] KRISHNAMURPHY, Balachander; MOGUL, Jeffrey C.; KRISTOL, David M.: Key differences between HTTP/1.0 and HTTP/1.1. In: *WWW '99: Proceedings of the eighth international conference on World Wide Web*. New York, NY, USA : Elsevier North-Holland, Inc., 1999, S. 1737–1751

[Koc00] KOCH, Nora: *Software Engineering for Adaptive Hypermedia Systems*, Ludwig-Maximilans-Universität München, Diss., 2000

[KR01] Kap. 6 HTTP Protocol Design and Description; 7 HTTP/1.1 In: KRISHNAMURTHY, Balachander; REXFORD, Jennifer: *Web Protocols and Practice*. Indianapolis, Indiana, USA : Addison Wesley, 2001, S. 173 – 300. – ISBN 0201710889

[Kro08] KROISS, Christian: *Modellbasierte Generierung von Web-Anwendungen mit UWE (UML-based Web Engineering)*, Ludwig-Maximilans-Universität München, Diplomarbeit, 2008

[LA94] LUOTONEN, Ari; ALTIS, Kevin: World-Wide Web proxies. In: *Selected papers of the first conference on World-Wide Web*. Amsterdam, The Netherlands, The Netherlands : Elsevier Science Publishers B. V., 1994, S. 147–154

[Lee08] LEEB, Claudia: *Integrierung von Ajax in JSF-Anwendungen*, Fachhochschule Hagenberg, Bachelorarbeit, 2008

[Lie05] LIE, Håkon Wium: *Cascading Style Sheets*, University of Oslo, Diss., 2005

[Loo95] Kap. 7.1 Darstellung unausgewerteter Ausrücke In: LOOGEN, Rita: *Integration funktionaler und logischer Programmiersprachen*. München : Oldenbourg, 1995, S. 204–206. – ISBN 3–486–23040–9

[MCLPSF07] MORALES-CHAPARRO, R.; LINAJE, M.; PRECIADO, J. C.; SÁNCHEZ-FIGUEROA, F.: MVC Web design patterns and Rich Internet Applications. In: *SHCA 2007: Taller en Sistemas Hipermedia Colaborativos y Adaptativos* 1 (2007), S. 39–64

[MD08] MESBAH, Ali; VAN DEURSEN, Arie: A component- and push-based architectural style for ajax applications. In: *Journal of Systems and Software* 81 (2008), Nr. 12, S. 2194–2209. – ISSN 0164–1212

[PS07] PORTENEUVE, Christophe; STEINBERG, Daniel: *Prototype and script.aculo.us*. Raleigh, NC : The Pragmatic Bookshelf, 2007. – ISBN 978–1–934356–01–2

[Ric06] RICHTER, Jan: *Konzeption und Entwicklung von Mechanismen zur Unterstützung des Oberflächen-Datenflusses in Web-Anwendungen mit automatischer Dialogsteuerung*, Universität Leipzig, Diplomarbeit, 2006

[Roo06] OP'T ROODT, Youri: *The effect of Ajax on performance and usability in web environments*, Universiteit van Amsterdam, Masterarbeit, 2006

[SG03] Kap. 3 Understanding Application Layer Protocols In: SYME, Matthew; GOLDIE, Philip: *Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing*. Prentice Hall International, 2003. – ISBN 978–0131014688

[SGBF01] SCHMIDT, Albrecht; GELLERSEN, Hans-W.; BEIGL, Michael; FRICK, Oliver: Entwicklung von WAP-Anwendungen. In: *Kommunikation in Verteilten System (KiVS) - 12. Fachkonferenz der Gesellschaft für Informatik (GI) Fachgruppe*, Springer Verlag, 2001. – ISBN 3–540–41645–5, S. 81–92

[SH04] STENBACK, Johnny; HENINGER, Andy: Document Object Model (DOM) Level 3 Load and Save Specification / W3C. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407 (01.01.2008)

[Whi04] WHITE, Stephen A.: Introduction to BPMN. In: *BPTrends* (2004)

[Wil99] WILDE, Erik: *World Wide Web: Technische Grundlagen*. Berlin : Springer Verlag, August 1999, S. 191. – ISBN 3–540–64700–7

[Zak05] ZAKAS, Nicolas C.: *Professional JavaScript for Web Developers*. Indianapolis, Indiana, USA : Wiley Oublishing, 2005. – ISBN 978–0–7645–7908–0