Diplomarbeit

Softwarearchitektur von J2EE Web Applikationen mit Web Components am Beispiel von AJAX

Christoph Hohmann
Matr-Nr. 5099117
reboot@gmx.ch

Betreuer
Dr. Wolf-Gideon Bleek
bleek@informatik.uni-hamburg.de

Dr. Daniel Moldt
moldt@informatik.uni-hamburg.de

27. Januar 2007

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Das World Wide Web hat sich in den letzten Jahren immer mehr von einem reinen Informationsangebot zur Dienstleistungsplattform entwickelt [MS04]. Der Nutzer des World Wide Web kann nicht länger nur Informationen nachschlagen und die darin referenzierten Texte durch Anklicken eines Links direkt öffnen, sondern selber den Inhalt des Webs mitgestalten, Dienstleistungen von Unternehmen in Anspruch nehmen und Waren online kaufen.

Um den Nutzer des World Wide Web hierfür eine bessere, schnellere und einfacherer Nutzung interaktiver Webseiten zu ermöglichen, werden die im Web eingesetzten Technologien immer wieder entweder durch neue Technologien ersetzt, durch neue Versionen verbessert oder durch andere Technologien ergänzt. All dies mit dem Ziel dem Nutzer einer Website die gleiche reichhaltige Interaktion zu ermöglichen, wie er sie von Desktop Anwendungen gewohnt ist.

1.2 Probleme

Die Technologien, die derzeit im World Wide Web eingesetzt werden, sind jedoch immer noch nicht ausreichend dafür geeignet, um komplexe interaktive Websites zu realisieren, wie sie Rich-Clients bieten.

Der wesentliche Punkt hierbei ist, dass eine Webseite, wenn sie einmal übertragen wurde, vom Webserver unabhängig ist. Die Interaktionsmöglichkeiten, die sie bietet, müssen vollständig übertragen worden sein, wie bei einem Programm, dass man sich auf seinen Computer herunterlädt. Möchte man also zum Beispiel bei einem Internetdienst etwas suchen, bleibt einem nur die Möglichkeit eine komplett neue Webseite zu laden, da man unmöglich den gesamten Datenbestand des Dienstes mit einer einzigen Webseite herunterladen kann. Dies ist die bisher gängige Praxis zur Realisierung interaktiver Websites.

In den folgenden Abschnitten wird auf die einzelnen Technologien genauer eingegangen und erläutert welche Probleme sich durch diese für die Realisierung interaktiver Websites ergeben.

1.3 Lösungsansatz

Eine mögliche Lösung für das oben genannte Problem bietet Asynchronons JavaScript and XML (Ajax). Ajax selbst ist keine neue Technologie, sondern bezeichnet nur das Zusammenwirken der bisher verwendeten Technologien, um eine Anfrage an einen Webserver, aus einer bereits heruntergeladenen Webseite heraus, durchzuführen und damit den Inhalt der Webseite zu verändern. Dies wird möglich durch eine Erweiterung einer der bisher verwendeten Technologien. Um Ajax nutzen zu können, muss diese Erweiterung jedoch vom Webbrowser unterstützt werden.

Mit Ajax ist somit eine heruntergeladene Webseite nicht länger vom Webserver unabhängig. Sie kann auf Eingaben des Nutzers reagieren, indem sie Daten von einem Webserver abfragt und daraus neue Inhalte generiert, alte Inhalte

ersetzt oder diese löscht. Die alte Webseite bleibt während der Ajax Abfrage erhalten.

1.4 Aufgabenstellung

Ziel der Diplomarbeit ist es für JCommSy, eine Web-Anwendung auf Basis von J2EE, eine Softwarearchitektur zu entwickeln, die es erlaubt, die Interaktion mit der Anwendung durch Ajax zu verbessern. Ajax soll hierfür eine schnellere Interaktion mit der vorhandenen Webseite ermöglichen. Dabei soll die Anwendung nutzbar bleiben, wenn Ajax vom verwendeten Browser nicht unterstützt wird, indem die alte Funktionsweise erhalten bleibt.

Für die Architektur muss zunächst das vorhandene System analysiert werden, um Kriterien festzulegen anhand derer eine Architektur entwickelt werden kann, die eine Unterstützung von Ajax erlaubt. Die Kriterien werden dabei vor allem durch JCommSy vorgegeben, das in seiner Handhabung nicht verändert werden soll, sondern nur an der Oberfläche, an der die Benutzerinteraktion stattfindet. Für die Einbindung von Ajax soll dann ein Werkzeug eingesetzt werden, das sich gut mit den Kriterien verträgt und keine vollkommene Neustrukturierung des JCommSy erfordert.

Die Architektur soll dann im JCommSy umgesetzt werden. Ein Überblick über die Umsetzung gibt es in Abschnitt 5.

2 JCommSy

CommSy ist eine webbasiertes Community System. Das ursprünglich in PHP geschriebene CommSy wird derzeit auf einer J2EE Plattform migriert [Fer04][Sha02]. Dazu werden die einzelnen Bereiche des PHP CommSys nach und nach in Java reimplementiert. Diese Teile werden in das PHP System integriert, bis das komplette System auf Java umgestellt wurde. Das PHP System leitet dazu bei bereits migrierte Bereichen an das Java Servlet weiter und dieses im Gegenzug an das PHP System zurück, für nicht migrierte Bereiche.

Im Folgenden soll der Aufbau des CommSy und die Architektur der Java Reimplementierung vorgestellt werden.

2.1 Übersicht

Das CommSy hat eine hierarchische Struktur (siehe Abbildung 1). Die oberste Ebene bilden die Räume. Um in einem CommSy-Raum arbeiten zu können, müssen die Nutzer des Systems einen Account haben über den sie sich anmelden. Sie können in den Räumen Mitglied werden. Wobei es offene Räume gibt, in denen jeder selber Mitglied werden kann, und geschlossene Räume, bei denen der Zutritt nur durch einen Verwalter des Raumes erlaubt werden kann.

In jedem Raum können je nach Bedarf der Nutzergruppe verschiedene Rubriken aktiviert werden, wie zum Beispiel eine Terminverwaltung oder Ankündigungen. Die Nutzer können in den verschiedenen Rubriken Einträge erzeugen, die für andere Nutzer angezeigt werden. Diese können zu den Einträgen Anmerkungen schreiben oder, sofern der Ersteller des Eintrags es erlaubt, Veränderungen an dem Eintrag vornehmen. Wenn neue Anmerkungen vorhanden sind oder Änderungen vorgenommen wurden, wird dieses den anderen Nutzern direkt in der Auflistung der Einträge angezeigt, so dass sie sich leicht über alle Neuerungen informieren können. Hierfür gibt es beim Betreten des Raumes eine Übersicht, in der die neuesten und die geänderten Einträge aller Rubriken aufgelistet werden.

Außerdem bietet CommSy die Möglichkeit in jedem Raum dazu passende Materialien zu verwalten. Materialien können dabei jede Art von Dateien sein, wie Office-Dokumente, Bilder oder Musikdateien. Diese können mit den Einträgen in anderen Rubriken verknüpft werden.

2.2 Architektur

JCommSy ist nach einer Mehrschichten-Architektur aufgebaut, wie in Abbildung 2 dargestellt.

Die unterste Schicht bildet dabei das Modell, das die Klassen für die Fachwerte und fachliche Objekte enthält, auf denen das CommSy aufgebaut ist.

Diese werden von den Mappern benutzt, die Objekte auf eine relationale Datenbank abbilden. Ihr Aufgabe ist es, Objekte in die Datenbank zu schreiben und aus den Einträgen der Datenbank wieder Objekte zu erstellen. Sie können außerdem ganze Listen von Objekten abgefragen, die gewissen Suchkriterien entsprechen.

Abbildung 1: CommSy Struktur

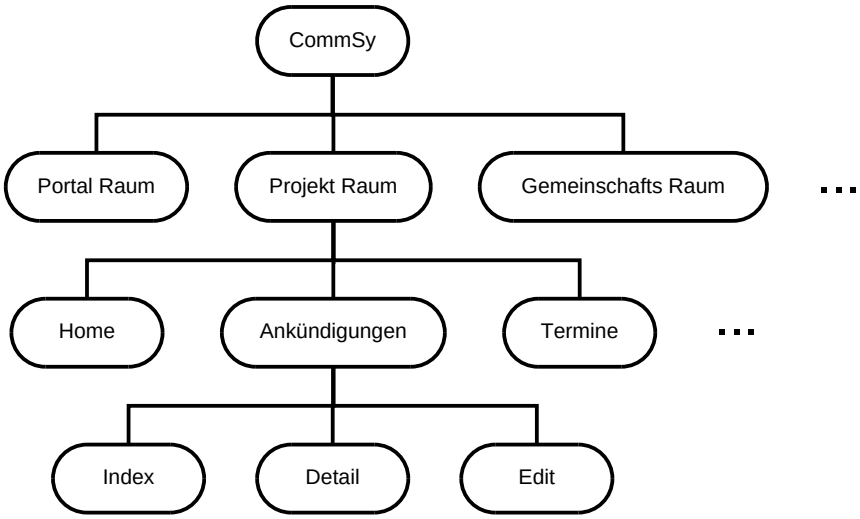


Abbildung 2: JCommSy Schichten Architektur

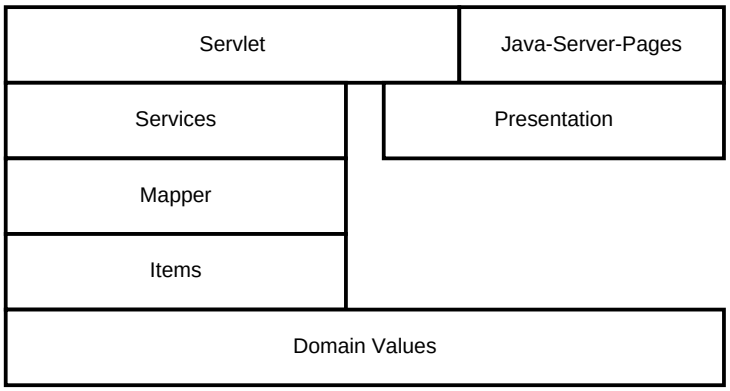
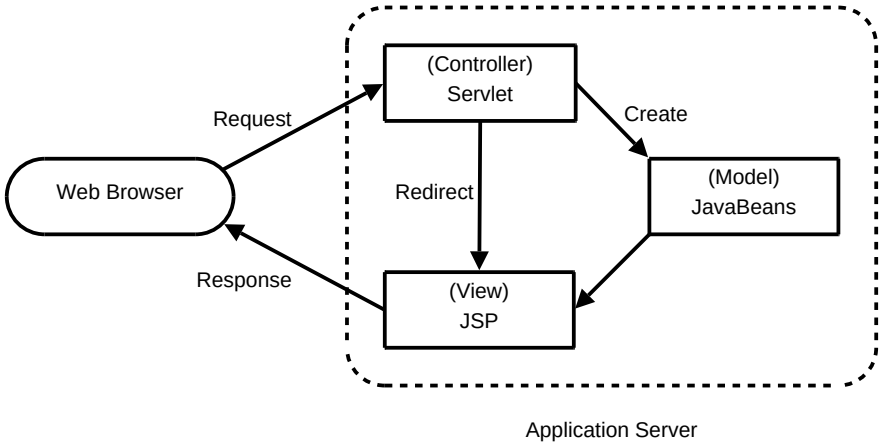


Abbildung 3: JSP Model 2 Architektur



Auf den Mappern setzen die fachlichen Services auf. Diese enthalten die fachliche Funktionalität, die für das CommSy benötigt wird. Dies umfasst die Verwaltung von Räumen, Materialien und der Einträge in den verschiedenen Rubriken.

Auf Basis der Services ist das JCommSy-Servlet aufgebaut. Dieses nutzt die Services, um Daten abzufragen, die für eine spätere Präsentation in einer Webseite benötigt werden. Außerdem werden Eingaben für Änderungen, die der Benutzer der Website tätigt, in Änderungen der Objekte umgesetzt, die über die Services gespeichert werden.

Das JCommSy-Servlet ist nach der JSP-Model-2 (siehe [Pat02] und Abbildung 3) Architektur aufgebaut. Dabei wird jeder Request zuerst an ein Servlet deligiert. Dieses holt alle für die Darstellung der Webseite nötigen Daten von den Services, erstellt Java Beans für die Präsentation der Daten und legt diese im `HttpServletRequest`-Objekt ab. Das Servlet leitet den Request danach an eine Java-Server-Page weiter, die mit den Daten aus den Beans ein HTML Dokument erzeugt, das anschliessend vom Servlet-Container zum Browser übertragen wird.

Im JCommSy wird jeder Request zuerst an ein allgemeines `CommSy`-Servlet deligiert. Das `CommsyServlet` fragt von den Services die Daten des aktuellen Benutzers und des Raumes ab. Daraus werden erstens Beans für den Benutzerbereich erzeugt, über den der Benutzer auf seine persönlichen Seiten zugreifen und zu Räumen wechseln kann, in denen er Mitglied ist. Zweitens werden Beans für den aktuellen Raum erzeugt. Darin enthalten sind zum Beispiel der Raumname und eine Liste der aktiven Rubriken des Raumes.

Danach deligiert das `CommSyServlet` den Request an ein spezielles Servlet für die darzustellende Rubrik. Dieses erzeugt die unterschiedlichen Seiten innerhalb einer Rubrik, wie Index-, Detail- und Editseite. Dafür werden die benötigten Daten über die entsprechenden Services abgefragt und in Beans eingetragen. Außerdem werden Benutzereingaben verarbeitet, wie Suchparameter,

um die Ansicht entsprechend zu verändern, oder Eingaben aus der Editseite, um neue Objekte zu erzeugen oder vorhandene zu verändern.

Die Servlets sind dabei nicht weiter strukturiert. Für jede Funktion einer Rubrik gibt es eine Methode, die Daten abfragt, daraus Java Beans erzeugt, diese in den Request einträgt und am Ende den Request an die CommSy-Java-Server-Page weiterleitet. Dabei wird der Dateiname für die Datei, die die Java-Server-Page für die zugehörige Rubrikfunktion enthält, in den Request eingetragen, so dass diese in die CommSy-Java-Server-Page eingebunden werden kann. Zu jeder Funktion gehört somit eine Java-Server-Page, die alle Daten darstellt. Eine Zerlegung der Funktionalität und der Java-Server-Pages in kleinere Elemente gibt es dabei nicht.

3 Ajax

Die Benutzung von Desktop Anwendungen unterscheidet sich derzeit, von der Benutzung von webbasierten Anwendungen. Die Probleme, die webbasierte Anwendungen haben, sollen zunächst in diesem Abschnitt aufgezeigt werden. Danach wird die Ajax-Technologie vorgestellt, die es ermöglicht, einige dieser Definzite webbasierter Anwendungen zu beseitigen. Abschließend wird ein Toolkit vorgestellt, das die Entwicklung von Ajax Webseiten auf Basis von J2EE Servlets unterstützt.

3.1 Web Technologien

Die im World Wide Web eingesetzten Technologien sind im wesentlichen das Hypertext Transfer Protocol (HTTP), die Hypertext Markup Language (HTML) und JavaScript.

3.1.1 HTTP

Das Hypertext Transport Protocol (HTTP) dient zur Dateiübertragung über Netzwerke [FGM+99]. Es wird für die Übertragung der meisten Dateien im World Wide Web genutzt. HTTP ist auf ein Transportmedium angewiesen, das die Übertragung zwischen Client und Server regelt. In den meisten Fällen wird hierfür das TCP Protokoll verwendet. Bei HTTP muss der Client bei jeder Anfrage vollständig spezifizieren, welche Daten er benötigt. Erst danach werden diese vom Server ausgeliefert. HTTP ist somit ein zustandsloses Protokoll. Mehrere Anfragen in Folge sind von einander unabhängig und können sich nicht auf bereits vorher übertragene Daten beziehen. Ursprünglich wurde HTTP benutzt, um HTML Dateien und Bilder im World Wide Web zu übertragen. Heutzutage wird es zusätzlich für andere Zwecke eingesetzt, zum Beispiel als Transportmedium für höhere Dienste.

3.1.2 HTML

Die HyperText Markup Language (HTML) ist eine Beschreibungssprache für Dokumente im World Wide Web, die auf dem SGML Standard aufbaut [RHJ99]. Mit HTML kann der Text auf Webseiten wie in Textverarbeitungen formatiert werden. Dazu wird über so genannte Tags, die im Text eingebaut werden, angegeben, welche Schriftgröße oder welche Schriftstile man haben möchte, oder welche Bilder man einbinden will. Komplexere Strukturen wie Tabellen lassen sich über ineinander verschachtelte Tags definieren. Die Tags sind dabei selbst ganz normaler Text, so dass zum Erstellen von HTML Dokumenten ein einfacher Texteditor ausreichend ist und keine spezielle Software benötigt wird. Die Tags werden vom Webbrowser interpretiert und in eine entsprechende Darstellung umgesetzt.

Die Interpretation von SGML kann jedoch teilweise schwierig sein, da SGML nicht strikt ist, was die Anordnung von Tags angeht. Tags können in beliebiger Reihenfolge geöffnet und geschlossen werden, teilweise können schließende

Tags sogar ausgelassen werden. Aus diesem Grund gibt es eine HTML Spezifikation, die auf dem XML Standard aufbaut, und als XHTML bezeichnet wird [BPSM+06]. XML ist im Gegensatz zu SGML strikt, was die Einhaltung der Regeln für Tags angeht. Dies macht eine Verarbeitung einfacher, da die Struktur des Dokumentes eindeutig und keine Interpretation des Browsers ist.

3.1.3 JavaScript

Da die Struktur von HTML Dokumenten statisch ist, kann mit reinem HTML keine Benutzerinteraktion erreicht werden. Hierfür wurde JavaScript eingeführt [Fla02]. Bei JavaScript handelt es sich um eine Programmiersprache, die in HTML Dokumenten eingebettet oder referenziert werden kann. Das enthaltene Programm wird vom Webbrowser aufgeführt. Um eine Interaktion mit dem Benutzer zu ermöglichen, kann im HTML Dokument für bestimmte Ereignisse, wie zum Beispiel Mausklicks, ein JavaScript Programm angegeben werden, das ausgeführt wird, wenn das Ereignis eintritt.

Da JavaScript direkt im Browser ausgeführt wird, kann es auf das HTML Dokument, in dem es enthalten ist, zugreifen und dieses verändern. Auf diese Weise kann der Benutzer auf seine Aktionen eine direkte Rückmeldung durch Veränderungen des dargestellten Dokumentes erhalten.

Da das JavaScript Bestandteil der Webseite ist, muss es zusammen mit dieser zumindest einmal übertragen werden, wenn es gecacht werden kann. Dies schränkt die Größe der Programme ein, da sie von den Benutzern der Webseite heruntergeladen werden müssen, was in einer angemessenen Zeit möglich sein muss.

3.2 Interaktion in Web Anwendungen

Interaktion in Web Anwendungen kann auf 2 Weisen realisiert werden. Die erste Möglichkeit ist, dass durch das Anklicken eines Links eine vom Server dynamisch generierte Seite geladen wird. Der Zustand der Anwendung wird dabei vom Server verwaltet, durch jeden neuen Seitenaufruf wird dieser Zustand anhand der Parameter der URL aktualisiert, dann für den neuen Zustand eine Seite generiert und diese an den Browser übertragen.

Da HTML immer eine vollständige Webseite beschreibt, muss bei jeder Interaktion mit der Website, ein vollständiges HTML Dokument übertragen werden. Also müssen die Teile der Seite, die der Browser eigentlich bereits hat, weil sie sich von einer Seite zur nächsten nicht verändert haben, mit übertragen werden. Da dies für den Browser nicht ersichtlich ist, muss er die neue Seite komplett neu aufbauen. Außerdem muss wegen der Zustandlosigkeit von HTTP für jedes dieser Dokumente eine vollständige HTTP Anfrage an den Webserver geschickt werden. Dies nimmt bei jeder Interaktion unnötig Zeit in Anspruch und macht eine direkte Interaktion, wie man sie von Desktop-Anwendungen gewohnt ist, unmöglich.

Die zweite Möglichkeit ist, Interaktion über JavaScript-Programme zu realisieren. Die Möglichkeiten eines JavaScript-Programmes sind jedoch auf das beschränkt, was bei der Übertragung der Webseite an Programmcode und Daten

übertragen wurde. Somit ist es nicht möglich, Teile der Webseite entsprechend der Benutzereingaben zu verändern, wenn dafür Daten benötigt werden, die nur der Webserver kennt und es nicht möglich ist, den gesamten Datenbestand in dem JavaScript Programm mit an den Browser zu übertragen. Solche Änderungen an der Webseite erfordern also immer ein Neuladen der gesamten Seite, da HTML nur die Beschreibung einer ganzen Seite erlaubt, wie bereits oben beschrieben.

3.3 Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (kurz Ajax) bezeichnet das Zusammenwirken verschiedener Technologien, um im World Wide Web neue Web Anwendungen zu realisieren, die eine direktere Interaktion mit dem Benutzer ermöglichen. Der Begriff wurde von Jesse James Garrett in seinem Artikel „Ajax: A New Approach to Web Applications" eingeführt [Gar05].

Ajax nutzt eine Erweiterung von JavaScript, die es erlaubt asynchron HTTP Anfragen auszuführen und das Ergebnis dieser Anfrage durch ein JavaScript verarbeiten zu lassen, sobald diese beendet ist. Währenddessen bleibt das HTML Dokument, in dem das JavaScript enthalten ist, unberührt. Ob und wie es nach Ende der HTTP Anfrage verändert wird, kann von dem erhaltenen Dokument abhängig gemacht werden. Ist das Dokument ein gültiges XML Dokument, wird dieses automatisch in einen Document Object Model Baum abgebildet, so dass die Daten für das JavaScript in einer leicht zu verarbeitenden Struktur vorliegen [HHW+04].

3.3.1 XMLHttpRequest

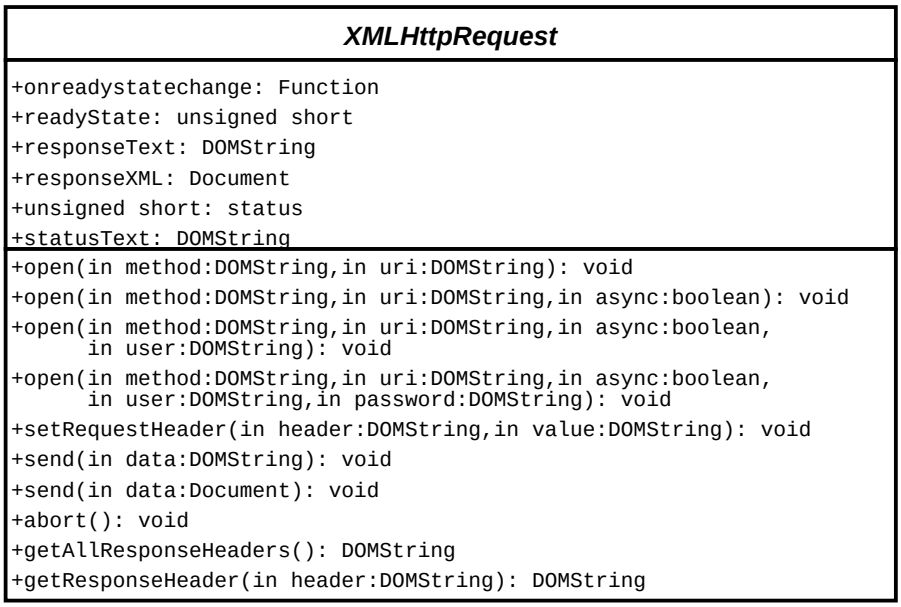
Die Funktionweise des `XMLHttpRequest` wurde ursprünglich von Microsoft definiert und als ActiveX-Objekt im Internet Explorer eingebaut. Inzwischen liegt sie als Standardvorschlag vom W3C für JavaScript vor, der von allen neueren Browsern umgesetzt wird [Kes06]. In folgenden Beispielen wird daher nur die standardisierte Variante benutzt. Das Interface von `XMLHttpRequest` ist als UML Klassendiagramm in Abbildung 4 dargestellt.

Über die Methoden und Attribute können alle Parameter für einen HTTP Request eingestellt werden. Dabei werden alle HTTP Anfragetypen GET, POST, HEAD und PUT unterstützt. Zusätzliche HTTP Header können mit angegeben werden, um spezielle Anfragen, wie sie zum Beispiel für einen Web Service nötig sind, zu ermöglichen. Das Objekt speichert während des Requests die HTTP Response Header und das empfangene Dokument. Diese können nach einem erfolgreichen Request über die Methoden und Attribute des `XMLHttpRequest`-Objektes ausgelesen und verarbeitet werden.

3.3.2 Request Ablauf

Um mit JavaScript einen asynchronen HTTP Request auszuführen, muss zunächst ein `XMLHttpRequest`-Objekt erzeugt werden. In diesem Objekt werden alle Informationen, die für die HTTP Anfrage nötig sind, wie die URL des Dokumentes mit der `open()`-Methode, eingetragen. Um über Änderung am Status

Abbildung 4: XMLHttpRequest Interface



der HTTP Anfrage benachrichtigt zu werden, muss die `onreadystatechange`-Callback Funktion angegeben werden, die in diesem Fall vom Browser aufgerufen wird. Danach kann die Anfrage mit `send()` gestartet werden. Über den Callback wird das JavaScript Programm informiert, wenn sich der Status der Anfrage ändert.

Das Attribut `readyState` gibt den aktuellen Zustand des `XMLHttpRequest`-Objektes an. Die möglichen Werte für `readyState` sind in Tabelle 1 aufgelistet. Wird der Callback aufgerufen und das `readyState`-Attribut enthält den Wert 4, wurde der Request beendet und das `status`-Attribut enthält den HTTP-Status-Code. Das empfangene Dokument kann dann bei einem erfolgreichen HTTP Request (Status-Code 200) entweder als Text über das `responseText`-Attribut oder, sofern es sich um ein gültiges XML Dokument handelt, als XML-Dokument über das `responseXML`-Attribut zugegriffen werden. Die Auswertung der enthaltenen Daten bleibt dabei allein dem JavaScript überlassen. Der Ablauf des Requests ist nochmal als UML Sequenzdiagramm in Abbildung 5 dargestellt.

3.3.3 Beispiel für Ajax JavaScripts

Ein JavaScript, das Ajax-Requests ausführt, wird als Ajax-Engine bezeichnet. Eine Ajax-Engine besteht im wesentlichen aus 2 Teilen. Einer Funktion um Ajax-Requests zu starten, und einer Callback-Funktion, die die Request-Ergebnisse verarbeitet. Die Funktion zum Starten von Ajax-Request kann von Event-Handlern genutzt werden kann, die bei Ereignissen wie zum Beispiel Benutzeraktionen oder Timern aufgerufen werden.

Listing 1 ist ein Beispiel für einen einfachen Event-Handler, der die benö-

Abbildung 5: Asynchroner HTTP Request

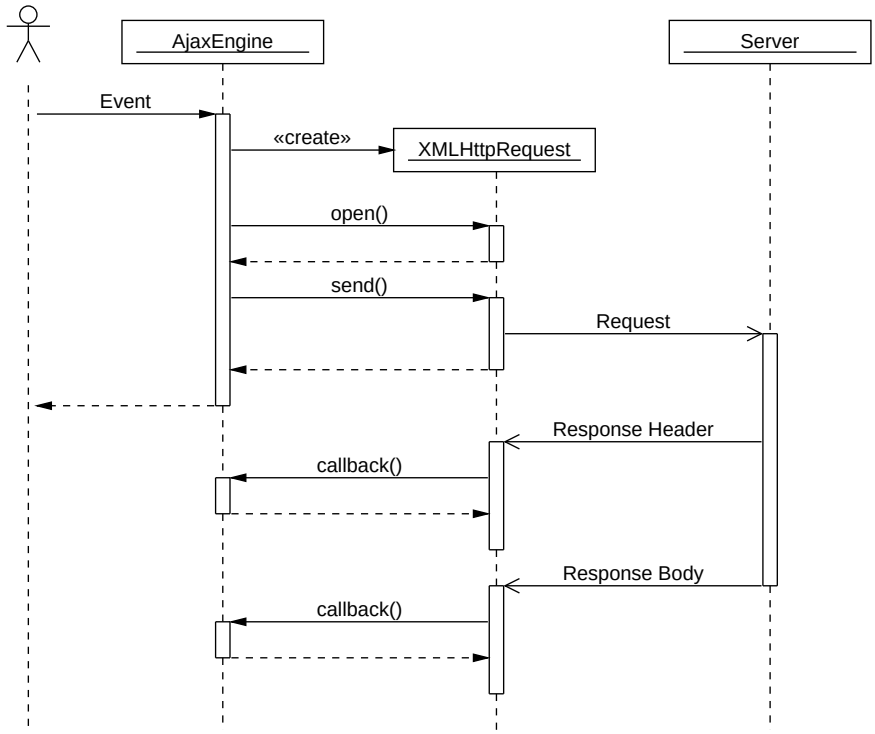


Tabelle 1: XMLHttpRequest Zustände

Wert	Bedeutung
0	**Uninitialized**: Initialer Zustand nachdem das Objekt erzeugt wurde.
1	**Open**: Die `open()`-Methode wurde erfolgreich aufgerufen und das Objekt initialisiert.
2	**Sent**: Der HTTP Request wurde erfolgreich gesendet, es wurden aber noch keine Daten empfangen.
3	**Receiving**: Der HTTP Response Header wurde erfolgreich empfangen. In der Folge wird das Ergebnis Dokument, sofern vorhanden, der HTTP Anfrage empfangen. Das `status`-Attribut enthält den HTTP Status Code.
4	**Loaded**: Der Datentransfer wurde abgeschlossen.

tigten Schritte ausführt, um einen Ajax-Request zu starten, bei dem bei Statusveränderungen die Funktion mit dem Namen `callback` aufgerufen wird.

Listing 1: JavaScript Event-Handler (ajax1.js)

```
1  var request = null;
2
3  function userAction()
4  {
5      request = new XMLHttpRequest();
6
7      request.open("GET", "data.xml", true);
8      request.onreadystatechange = callback;
9      request.send(null);
10 }
```

Der Event-Handler erzeugt zuerst ein `XMLHttpRequest`-Objekt (Zeile 5). Danach wird die `open()`-Methode aufgerufen, um das Ziel des Request anzugeben (Zeile 7). Dann wird der Callback für Statusveränderungen festgelegt (Zeile 8) und der Request mit der `send()`-Methode gestartet (Zeile 9). Das Objekt wird in einer globalen Variable gespeichert (Zeile 1 und 5).

Für eine vollwertige Ajax-Engine wäre noch zusätzlicher Code erforderlich, der verschiedene Browser unterstützt, die nicht das standardisierte `XMLHttpRequest`-Objekt kennen, aber etwas kompatibles haben, wie der Internet Explorer, oder erkennt, wenn der Browser gar kein Ajax unterstützt. Außerdem wäre eine Verwaltung der Requests sinnvoll, wenn die Ajax-Engine die Ausführung mehrerer paralleler Requests unterstützen soll.

Der dazugehörige Callback kann wie in Listing 2 aussehen.

Listing 2: JavaScript Ajax Callback (ajax2.js)

```
1  function callback()
2  {
3      if (request.readyState != 4)
4          return;
5
6      if (request.status != 200) {
7          alert("Error: " + request.statusText);
8          return;
9      }
10
11     var target = document.getElementById("target");
12     var data = request.responseXML.
13         getElementsByTagName("data")[0];
14     target.innerHTML = data.childNodes[0].nodeValue;
15 }
```

Dieser einfache Callback prüft zunächst, ob der Request noch nicht abgeschlossen wurde (`readyState != 4`, Zeile 3-4). Ist dies der Fall, wird der Callback sofort verlassen und somit der Aufruf ignoriert. Ist der Request beendet,

wird der HTTP-Status-Code geprüft. Im Falle eines Fehlers (`status != 200`, Zeile 6-9), wird der Fehler über die JavaScript Funktion `alert()` angezeigt. Bei einem erfolgreichen Request wird ein Teil des HTML-Dokumentes durch Daten aus dem empfangenen Dokument ersetzt.

Wie komplex der Callback für Webseiten aussieht, hängt davon ab, welche Daten man empfängt. Je nach Anwendungen kann dies von einfachen Texten bis hin zu komplexen XML Dokumenten gehen.

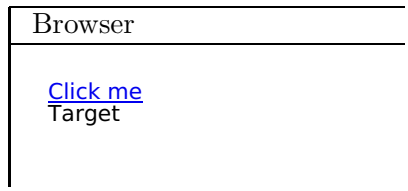
Event-Handler und Ajax-Callback (die üblicherweise in einem einzigen JavaScript stehen) werden zusammen in einem HTML-Dokument eingebunden, das eine Möglichkeit enthält den Event-Handler aufzurufen.

Listing 3: Ajax HTML (ajax.html)

```
1  <html>
2  <head>
3    <script type="text/javascript" src="ajax1.js" />
4    <script type="text/javascript" src="ajax2.js" />
5  </head>
6  <body>
7    <a href="javascript:userAction()">Click me</a>
8    <div id="target">Target</div>
9  </body>
10 </html>
```

In Zeile 3 und 4 werden die beiden JavaScripts aus Listing 1 und 2 eingebunden. In Zeile 7 wird im HTML-Body ein Link definiert, der den Event-Handler aufruft, wenn der Link angeklickt wird. In Zeile 8 wird ein HTML-Element definiert, das später als Ziel dient, in dem der Callback den empfangenen Text einsetzen kann.

Diese einfache HTML Seite sieht im Browser etwa so aus:



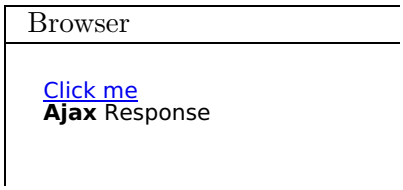
Im Beispiel Event-Handler (Listing 1) wird das Dokument data.xml vom WebServer angefordert und der Inhalt des `<div id="target">...</div>`-Tags durch den Inhalt des `data`-Tags ersetzt.

Listing 4: Beispiel Daten für den Ajax Callback (data.xml)

```
1  <data>
2      <![CDATA[<b>Ajax</b> Response]]>
3  </data>
```

Das XML-Dokument aus Listing 4 enthält den HTML-Text als CDATA, damit die HTML-Tags nicht von XML-Parser verarbeitet werden. So kann der HTML-Text über das JavaScript `innerHTML`-Attribute von HTML-Elementen

ersetzt werden. Die hieraus resultierende Webseite sieht dann im Browser wie folgt aus:



3.3.4 Vorteile von Ajax

Durch Ajax lassen sich auf einfache Weise Webseiten mit Daten, die nur serverseitig vorhandenen sind, dynamisch verändern ohne eine vollständige Webseite zu laden. Die Möglichkeiten der Gestaltung sind dabei lediglich von HTML und JavaScript eingeschränkt. Ein weiterer Vorteil ist, dass das zu übertragende XML-Dokument für Ajax-Requests kleiner ist als eine vollständiges HTML-Dokument, was die Übertragungszeit verkürzt. Da der Browser nur den veränderten Teil der Webseite neu darstellen muss, kann Ajax zu einem beschleunigten Bildaufbau im Browser führen.

3.4 AjaxAnywhere

AjaxAnywhere ist ein Hilfsmittel, das die Entwicklung von Ajax Webseiten auf J2EE Webservern unterstützt [Aja]. AjaxAnywhere stellt keine neue Funktionalität für Webseiten bereit, sondern unterstützt die Aktualisierung einer bereits geladenen Webseite. Voraussetzung für die Nutzung von AjaxAnywhere ist allerdings, dass für die Erzeugung der Webseite Java-Server-Pages benutzt werden. AjaxAnywhere ist dabei minimal invasiv. Die vorhandene Architektur kann dabei beibehalten werden und muss nur um Funktionen für AjaxAnywhere erweitert werden.

Im Folgenden sollen zunächst die Bestandteile von AjaxAnywhere erläutert und gezeigt werden, wie diese in ein vorhandenes Servlet integriert werden können.

3.4.1 Übersicht

AjaxAnywhere enthält 4 Bestandteile, mit denen die Realisierung von Ajax-Anwendungen für J2EE-Anwendungen unterstützt wird. Client seitig wird eine Ajax-Engine bereitgestellt, mit der Ajax-Requests gestartet werden können und die deren Ergebnisse verarbeitet. Server seitig wird eine JSP-TagLib und ein Servlet-Filter zur Verfügung gestellt. Über eine Utilities-Klasse können Parameter des Requests, die Ajax betreffen, abgefragt und verändert werden.

JavaScript AjaxEngine Die Ajax-Engine von AjaxAnywhere muss als JavaScript in die Webseite eingebunden werden. Üblicherweise gelangt man auf Websites entweder über Links oder über Formulare auf eine neue Seite. Für diese beiden Aktionen bietet die Ajax-Engine entsprechende JavaScript-Funktionen,

die entweder eine neue Seite ähnlich einem Link laden oder ein Formular abschicken. Die hierfür nötige Verarbeitung des Request wird von der Ajax-Engine erledigt. Zusätzlich wird eine Information zum Request hinzugefügt, die es server seitig erlaubt, einen Ajax-Request von einem normalen Request zu unterscheiden.

Die Antwort des Servers wird von der Ajax-Engine verarbeitet und Teile der Seite entsprechend verändert. Zusätzlich bietet die AjaxEngine Funktionen wie das Vorladen von Bildern und das Ausführen von in der Antwort enthaltenen JavaScripts an, so dass hierfür kein Programmieraufwand mehr nötig ist.

Servlet-Filter AjaxAnywhere erfordert nur minimale Änderungen an einem Servlet, um Ajax zu unterstützen. Die Ausgabe des Servlets ist daher weiterhin normales HTML. Bei Ajax-Requests wird diese HTML-Ausgabe dann vom AjaxAnywhere Servlet-Filter (implementiert in der `AAFilter`-Klasse) in ein AjaxAnywhere-XML-Dokument transformiert, das die Ajax-Engine im Browser verarbeiten kann. Das Servlet kann mit minimalen Veränderungen weiterhin die gleiche Ausgabe produzieren wie ohne AjaxAnywhere.

JSP-TagLib Mit der JSP-TagLib von AjaxAnywhere werden über Tags in den Java-Server-Pages die Bereiche, Zonen genannt, gekennzeichnet, die durch Ajax-Requests eigenständig aktualisiert werden können. Bei Requests von vollständigen Seiten wird am Anfang und Ende der Zone ein HTML-Tag eingefügt, der später dazu dient, die Zone im HTML-Dokument zu finden. Bei Ajax-Requests werden die Zonen vom Servlet-Filter für die Ajax-Antwort extrahiert, wenn sie client seitig aktualisiert werden sollen.

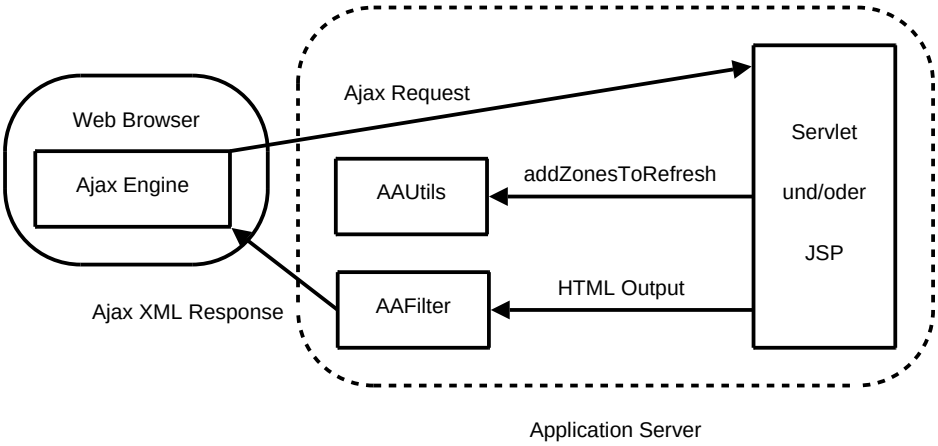
Utilities Die Utilities von AjaxAnywhere (Aufzurufen über statische Methoden der `AAUtils`-Klasse) dienen dazu, im Java-Code des Servlets zu prüfen, ob es sich bei einem Request um einen Ajax-Request handelt. Der Request wird von der Ajax-Engine so durchgeführt, das er von den Utilities als ein Ajax-Request erkannt werden kann. In diesem Fall können über die Utilities die Zonen festgelegt werden, die aktualisiert und vom Servlet-Filter aus der HTML-Seite extrahiert werden sollen.

3.4.2 Integration in Web-Applikationen

Die Integration von AjaxAnywhere in Servlets erfolgt wie in Abbildung 6 dargestellt. Zunächst muss der `AAFilter` in der `web.xml` eingetragen werden. Damit dieser alle Web Requests filtern kann, muss außerdem ein Filter-Mapping eingetragen werden, das den Filter auf die gleichen URLs abbildet wie das Servlet. Wenn noch andere Filter in der `web.xml` eingetragen sind, die das HTML Dokument verändern, müssen diese nach `AAFilter` eingetragen werden, damit dieser als erstes in der Filterkette aufgerufen wird, und die Ausgabe der anderen Filter mit verarbeiten kann.

Im Servlet kann mit den `AAUtils` geprüft werden, ob es sich bei dem Request, um einen Ajax-Request handelt. In diesem Fall kann das Servlet mit dem `AAUtils` in den Request eintragen, welche AjaxAnywhere-Zonen in die

Abbildung 6: AjaxAnywhere Integration



Ajax-Response vom `AAFilter` aufgenommen werden sollen. Es muss im Servlet eine Möglichkeit geschaffen werden, zu erkennen, welche Zonen der Client benötigt, weil sich ihr Inhalt durch den Request verändert hat. Hierfür könnte man zum Beispiel den letzten Zustand der Web-Anwendung im Servlet in der Session speichern und dann mit dem neuen Zustand vergleichen. Eine andere Möglichkeit wäre es, in den Request vom Client bereits mit einzubauen, welche Zonen aktualisiert werden sollen. Welche Möglichkeit in Frage kommt, ist von der Anwendung abhängig.

In den Java-Server-Pages muss zunächst die AjaxEngine von AjaxAnywhere eingebunden werden. Üblicherweise wird hierfür im `<head>`-Tag der Webseite ein `<script>`-Tag eingefügt. Die Bereiche, die man über Ajax verändern möchte, müssen mit dem `<zone>`-Tag aus der AjaxAnywhere-TagLib benannt werden. Zuletzt müssen dann Links und Formulare im HTML der Website so verändert werden, dass sie die Ajax Funktionen aus der AjaxEngine benutzen, zum Beispiel, in dem man bei `<a>`-Tags einen `onclick`-Event-Handler schreibt, der die `ajaxAnywhere.getAJAX()`-Funktion aufruft.

3.4.3 Beispiel

Das folgende Beispiel zeigt den Einsatz von AjaxAnywhere in einer JSP-Model-2 Architektur. Dafür wird zunächst ein Servlet benötigt, dass die Daten für die spätere Darstellung erzeugt. Listing 5 gibt ein einfaches Beispiel für ein solches Servlet mit AjaxAnywhere Unterstützung.

Listing 5: AjaxAnywhere Beispiel Servlet

```
1 public class Servlet extends HttpServlet {
2     protected void doGet(HttpServletRequest req,
3             HttpServletResponse resp) {
4
5         // Set Request Attribute
```

```
6           reg.setAttribute("value", Integer.toString(
7               new Random().nextInt(100)));
8
9           // If Ajax Request, set Zones to Refresh
10          if (AAUtils.isAjaxRequest(req)) {
11              AAUtils.addZonesToRefresh(req, "dynamic");
12          }
13
14          // Forward to JSP
15          req.getRequestDispatcher("/page.jsp")
16              .forward(req, resp);
17      }
18  }
```

In den Zeilen 5 bis 7 werden die Daten für die spätere Darstellung erzeugt. In diesem Beispiel wird dafür einfach eine Zufallszahl erzeugt und diese als Attribut im Request abgelegt, wie es im JSP-Model-2 üblich ist. Die Zeilen 10 bis 12 enthalten den nötigen Code für AjaxAnywhere. Zuerst wird geprüft, ob es sich bei dem Request um einen Ajax-Request handelt. Ist dies der Fall, wird die AjaxAnywhere-Zone festgelegt, die später aus der Java-Server-Page extrahiert und in der Ajax-Response enthalten sein soll. Zum Abschluss wird in Zeile 15 und 16 der Request an die Java-Server-Page weitergeleitet, um die Ausgabe der Seite zu erzeugen.

Die Java-Server-Page aus Listing 6 erzeugt das zugehörige HTML Dokument, das die Daten aus dem Servlet enthält.

Listing 6: AjaxAnywhere Beispiel JSP

```
1  <%@ taglib uri="http://ajaxanywhere.sourceforge.net/"
2      prefix="aa" %>
3
4  <html>
5  <head>
6      <script language="javascript" src="aa.js" />
7  </head>
8  <body>
9      Static Content!<br/>
10     <br/>
11
12     <%-- AjaxAnywhere Zone to Refresh using Ajax --%>
13     <aa:zone name="dynamic">
14     Dynamic Content: <c:out value="${value}"/><br/>
15
16     <a href="javascript:ajaxAnywhere.getAJAX('');">
17     Reload
18     </a>
19     </aa:zone>
20
21 </body>
```

```
22 </html>
```

Zunächst muss in der Java-Server-Page die AjaxAnywhere-TagLib eingebunden werden. Dies geschieht in Zeile 1 und 2. Der Rest der Seite ist zum größten Teil die normale Struktur eines HTML Dokumentes. Für AjaxAnywhere muss, wie in Zeile 6, die Ajax-Engine in die Seite eingebunden werden. Mit den Tags in Zeile 13 und 19 wird der Bereich der Ajax-Anywhere-Zone definiert, die bei Ajax-Requests aktualisiert werden kann. Der Tag bekommt den Namen der Zone als Parameter übergeben. Dieser wurde zuvor bereits im Servlet benutzt, um anzugeben, welche Zone aktualisiert werden soll. In Zeile 14 wird die Zufallszahl ausgegeben, die im Servlet erzeugt wurde. Zeile 16 bis 18 definieren einen Link, der die Funktion der Ajax Engine aufruft, die einen Ajax-Request an das Servlet schickt.

Zusammen bilden Servlet und Java-Server-Page eine funktionierende Ajax-Seite. Der Bereich, der in der Java-Server-Page als AjaxAnywhere-Zone definiert wurde, kann ohne die komplette Seite neu zu laden, aktualisiert werden. In diesem einfachen Beispiel wird nur eine AjaxAnywhere Zone definiert und diese bei jedem Request aktualisiert. In einer komplexeren Webseite können beliebig viele AjaxAnywhere-Zonen definiert werden und das Servlet kann entscheiden, welche davon aktualisiert werden sollen. Es ist möglich, die Zonen ineinander zu verschachteln, so dass durch das Aktualisieren einer Zone neue Zonen definiert werden können. Das Servlet kann bei jedem Request bestimmen, ob die inneren kleinen Zonen oder die umfassenden Zonen aktualisiert werden sollen.

3.4.4 Zusammenfassung

AjaxAnywhere erlaubt eine einfache Realisierung von Ajax Websites, ohne große Veränderungen am Java-Quelltext oder den Java-Server-Pages vornehmen zu müssen. Die Änderungen beschränken sich darauf, die AjaxAnywhere Zonen in den Java-Server-Pages durch Tags zu definieren und dann im Servlet anzugeben, welche dieser Zonen bei Ajax-Requests an den Client übertragen werden sollen.

Wenn AjaxAnywhere auf diese Weise eingesetzt wird, hat dies jedoch den Nachteil, dass bei jedem Request zunächst die ganze Webseite im Server durch die Java-Server-Pages aufgebaut wird. Aus dieser Ausgabe extrahiert der Servlet-Filter dann die Zonen. Dies ist auch der Fall, wenn nur kleine Änderungen an der Webseite durchgeführt werden sollen. Außerdem kann es durchaus schwierig sein zu entscheiden welche Teile der Webseite sich tatsächlich durch die Aktionen des Servlets verändert haben. Das Servlet muss ermitteln in welchen Zonen sich Änderungen befinden, damit diese aus der Webseite extrahiert und übertragen werden können, um den Zustandes der Web-Anwendung beim Client zu aktualisieren.

4 Web Components Architektur für JCommSy

Für die Unterstützung von Ajax ist die bisherige JCommSy-Servlet-Architektur nur wenig geeignet. Zur Realisierung der Ajax Unterstützung musste daher eine neue Architektur für das JCommSy entwickelt werden. Die aus dieser Neustrukturierung des JCommSy resultierende Architektur soll in diesem Abschnitt vorgestellt werden.

4.1 Zielsetzung

Das JSP-Model-2 das JCommSy derzeit verwendet, soll für die neue Architektur weiter verwendet werden.

Beim JSP-Model-2 werden alle Daten, die das Servlet erzeugt, als Attribute in den Request eingetragen und später von den Java-Server-Pages verwendet. Möchte man das Servlet jedoch in viele kleine logische Funktionen zerlegen, dann benötigt man hierfür mit zunehmender Komplexität des Servlets immer mehr Attribute. Diese müssen unter Namen im Request abgelegt werden. Damit diese sich nicht gegenseitig überschreiben, müsste für die Namen der Attribute eine Namenskonvention benutzt werden. Um dies zu vermeiden, soll für diese Architektur hier eine Datenstruktur definiert werden, die den Aufbau der Seite repräsentiert und die für die Darstellung nötigen Daten enthält. In den Servlet-Request muss dann nur noch ein Attribut, die Datenstruktur, eingetragen werden. Die Daten der einzelnen Elemente, aus denen sich die Struktur zusammensetzt, sollen dabei voneinander unabhängig sein.

Die Java-Server-Pages für die Architektur sollen so gestaltet werden, dass sie jeweils ein Element der Datenstruktur als HTML erzeugen. Um aus der Datenstruktur eine vollständige HTML-Seite zu generieren, müssen für die Java-Server-Pages Hilfsmittel zur Verfügung gestellt werden, die es der Java-Server-Page ermöglichen, auf das zugehörige Element der Datenstruktur zuzugreifen und zu ihr zugehörige Elemente einzubinden.

Die Datenstruktur soll im Servlet erzeugt werden. Dafür soll für das Servlet eine Architektur entwickelt werden, die die einzelnen Elemente der Datenstruktur erzeugt und diese zur Gesamtdatenstruktur verknüpft.

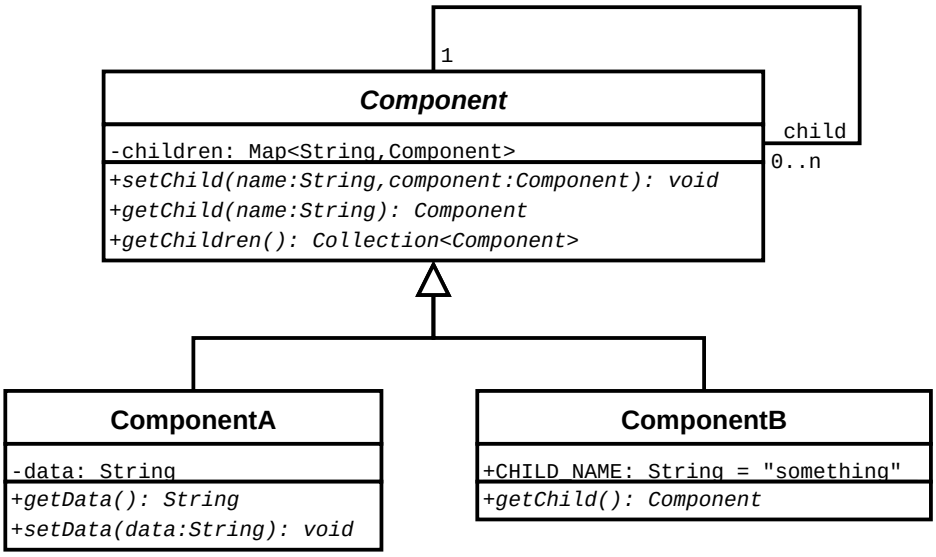
4.2 Architekturentwurf

Im folgenden sollen die 3 Teilbereiche der Architektur im Einzelnen spezifiziert werden. Die Klassen und die Funktionalität mit der später eine konkrete Seite realisiert werden kann, sollen hier definiert werden und erläutert werden, wie hiermit die zuvor definierten Ziele umgesetzt werden.

4.2.1 Datenstruktur

Die Webseite wird in einzelne Komponenten („Web Components") zerlegt. Jede Komponente ist eine Java Bean, in der die Daten für die spätere Darstellung gespeichert werden. Die Komponenten können weitere Komponenten enthalten, so dass diese eine baumartige Datenstruktur entsprechend dem Composite Pattern bilden [GHJV00, Seite 163 ff.]. Da HTML Seiten bereits eine hierarchische

Abbildung 7: Component Klassen



Struktur haben, bietet sich diese als Struktur für die Komponenten an. Die Bezeichnung „Web Component" wurde dabei aus der Bezeichnung „Component" aus dem Composite Pattern und der Tatsache, dass diese zur Erzeugung von Webseiten benutzt werden, zusammengesetzt.

Jede Komponente hat, wie bei Java Beans üblich, für ihre Attribute Getter- und Setter-Methoden zum Setzen und Auslesen der enthaltenen Daten. Die Klassen für jede Komponente werden als Unterklasse der `Component`-Klasse realisiert. Diese Oberklasse enthält die Verwaltung für die hierarchische Struktur.

Wie in Abbildung 7 dargestellt, enthält die `Component` Klasse, als Oberklasse aller Komponenten, Methoden für die Verwaltung der Kindkomponenten. Mit `setChild()` kann eine Kindkomponente unter einem Namen registriert werden. Mit `getChild()` kann diese dann entsprechend unter dem Namen wieder abgefragt werden. Zusätzlich gibt es noch die Methode `getChildren()` die alle Kindkomponenten einer Komponente als `Collection` zurückgibt.

Die Klasse `ComponentA` ist ein Beispiel für eine konkrete Komponente, die Daten enthält. Sie bietet für diese Daten die üblichen Java Bean Getter- und Setter-Methoden an, damit sie später in der zugehörigen Java-Server-Page über die Expression Language abgefragt werden können.

`ComponentB` ist ein Beispiel für eine Komponente, die eine Kindkomponente enthält. Der Name der Kindkomponenten wird hierfür als `public static final String` in die Klasse eingebaut. Diese Konstante kann dann beim Aufruf der `setChild()`-Methode benutzt werden. Um die Kindkomponente später über die Expression Language abfragen zu können, ist zusätzlich eine Getter-Methode ohne Parameter erforderlich. Diese kann dann die `getChild()`-Methode mit der Konstante aufrufen, um die Kindkomponente abzufragen.

4.2.2 Java-Server-Pages

Zu jeder Komponente gehört eine Java-Server-Page, die diese als HTML ausgibt. Der Dateiname, unter dem die Java-Server-Page abgelegt wird, wird dabei per Namenskonvention aus dem Klassennamen der Komponente gebildet. Zur Einbindung einer Komponente in eine Java-Server-Page wird eine TagLib benutzt. Der Tag zum Einbinden einer Komponente bekommt als Parameter das Komponenten-Objekt. Dieser kann über die Namenskonvention den Dateinamen der Java-Server-Page bestimmen und diese in die Webseite einbinden. Das Komponenten-Objekt wird vorher an das Request-Attribut `bean` gebunden. Innerhalb der Java-Server-Page der Komponente kann also mit der Expression Language über den Ausdruck `${bean.attribute}` auf die Daten der Komponente zugegriffen werden. Somit wird keine globale Namenskonvention benötigt, um die Request-Attribute zu benennen. Die Java-Server-Page der Komponente greift nur auf Attribute der Komponenten-Bean zu, die immer unter dem Attribut-Namen `bean` zugreifbar ist. Wenn eine Komponente aus einer anderen Komponente eingebunden wird, würde dies allerdings das bereits vorhandene `bean`-Attribut im Request überschreiben, so dass die Java-Server-Page der äußeren Komponente danach nicht mehr auf die Attribute der zu ihr gehörigen Bean zugreifen könnte. Darum muss der Tag ein bereits vorhandenes `bean`-Attribut vorher speichern und nach der Komponente wieder zurücksetzen.

Die Java-Server-Page der Komponente ist eine ganz normale Java-Server-Page, wie sie über den normalen `<jsp:include>`-Tag eingebunden werden könnte. Diese sollte für sich allein einen gültigen Body einer HTML-Seite ergeben und nicht von HTML-Tags in äußeren Komponenten abhängig sein, wie es zum Beispiel der Fall wäre, wenn die Komponente eine Zeile einer Tabelle enthalten würde. Außerdem sollte die Komponente keine HTML-Tags offen lassen oder HTML-Tags schließen, die in einer anderen Komponente geöffnet wurden. Nur so kann jede Komponente beliebig in andere Komponenten eingebunden und für sich alleine auf Korrektheit geprüft werden. TabLibs und die Expression Language zum Zugriff auf Request-Attribute können ganz normal benutzt werden, was zum Beispiel für das Einbinden von Kindkomponenten nötig ist. Hierfür werden die TagLib für Komponenten und die Bean der Kindkomponente, die über die Expression Language aus der Komponenten-Bean abgefragt werden kann, benötigt.

In den Servlet Request muss nur der Wurzelknoten des Baumes eingetragen werden. Dieser kann über den Tag in eine Java-Server-Page eingebunden werden. Von diesem ausgehend wird dann der gesamte weitere Baum der Datenstruktur abgearbeitet, indem jede Komponente ihre Kindkomponenten wieder über den Tag einbindet.

Für die in Beispielkomponenten aus Abbildung 7 könnten die Java-Server-Pages etwa wie folgt aussehen.

Listing 7: JSP für ComponentA

```
1 <%@ taglib prefix="c"
2     uri="http://java.sun.com/jstl/core"%>
3
```

```
4 Data: <c:out value="${bean.data}"/><br/>
```

Listing 7 enthält die Java-Server-Page für die Komponente A. Hier wird neben dem statischen HTML Text über den `<c:out ... />`-Tag der Java Standard TagLib das Attribut `data` der Java Bean zu dieser Komponente ausgegeben. Der Zugriff erfolgt über die Expression Language. Die TagLib, mit der Komponenten Java-Server-Pages eingebunden werden, hat die Bean zu dieser Komponente automatisch dem Request-Attribut `bean` zugewiesen. Über `${bean.data}` wird dann auf das Attribut der Bean zugegriffen, indem die Expression Language die `getData()`-Methode aufruft.

Listing 8: JSP für ComponentB

```
1 <%@ taglib prefix="component" uri="..."%>
2
3 Child-Component:<br/>
4 <component:include bean="${bean.child}"/>
```

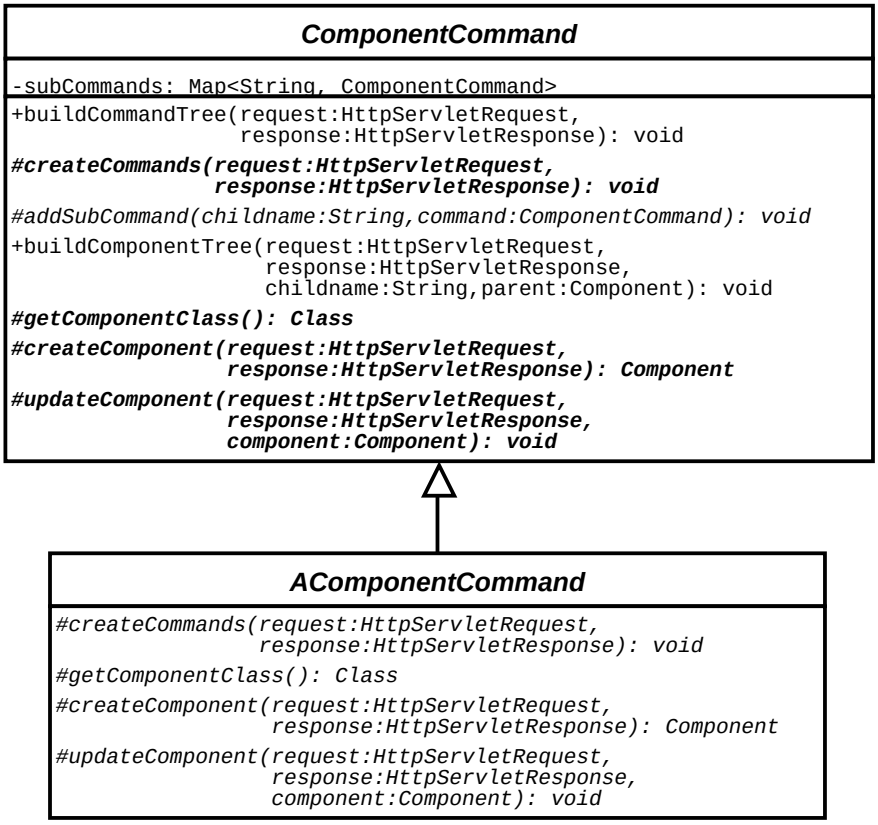
In Listing 8 werden im Gegensatz zu Listing 7 keine Daten der Komponenten-Bean ausgegeben. Hier wird über die eigens für die Architektur definierte TagLib eine Kindkomponente eingebunden. Die Expression Language fragt über die `getChild()`-Methode die zugehörige Bean ab und übergibt sie an den Tag. Dieser kann dann die Bean, wie bereits beschrieben, wieder dem Attribut `bean` zuweisen und die Java-Server-Page zu der übergebenen Bean einbinden.

4.2.3 Servlet Architektur

Zur Erzeugung der Komponenten im Servlet sollen Klassen verwendet werden, die eine gemeinsame Oberklasse haben. Zur Ausführung der Funktion wird nur das Interface der Oberklasse benötigt. Die Klassen werden daher in Anlehnung an das Command Pattern `ComponentCommand`s genannt [GHJV00, Seite 233 ff.]. Abweichend vom Command Pattern enthält die `ComponentCommand`-Klasse bereits Template-Methoden für die Methoden des Interfaces [GHJV00, Seite 325 ff.]. Diese enthalten die Funktionalität, um alle nötigen Commands zum Aufbau einer kompletten Seite rekursiv zu erzeugen und dann die Komponenten in den Komponentenbaum einzuhängen oder zu prüfen, ob bereits vorhandene Komponenten im Baum dem nötigen Typ entsprechen und aktualisiert werden können (siehe Abbildung 8). Außerdem werden die weiteren Commands für die Kindkomponenten aufgerufen.

Ein konkretes Command muss die abstrakten Methoden der `ComponentCommand`-Klasse implementieren. In diesen werden Benutzereingaben verarbeitet, Commands für Kindkomponenten erstellt, Beans für Komponenten erzeugt oder aktualisieren, wenn diese bereits vorhanden sind. Hierbei kann es vom jeweiligen Request abhängig sein, welche Commands erzeugt werden. Dadurch wird festgelegt, welche Komponenten erzeugt werden. Außerdem werden die Attribute der Komponenten anhand der Parameter des Servlet-Requests und Daten, die über die Services abgefragt werden, gesetzt. Die Verarbeitung des Requests durch die Commands erfolgt dabei in 2 Phasen.

Abbildung 8: UML-Klassendiagramm der abstrakten ComponentCommand Klasse

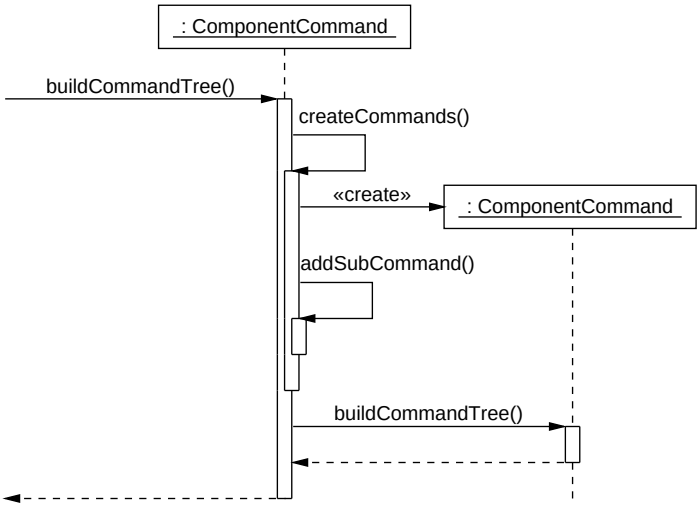


In der ersten Phase wird die Hierarchie der Commands aufgebaut und Benutzereingaben verarbeitet. In der zweiten Phase wird der Komponentenbaum erzeugt oder aktualisiert. Dies stellt sicher, dass in der zweiten Phase in alle Komponenten bereits aktualisierte Daten eingetragen werden, wenn Daten in der ersten Phase verändert wurden. Ansonsten wäre es möglich, dass Daten verändert werden, nachdem diese bereits in eine Komponente eingetragen wurden und bevor sie in eine andere Komponente eingetragen werden. Hierfür ohne 2 Phasen eine korrekte Ausführung zu erreichen, würde Abhängigkeiten in der Ausführungsreihenfolge der Commands erzwingen, die, je größer die Anwendung wird, unüberschaubarer wenn nicht sogar unlösbar werden, wenn die Abhängigkeiten zyklisch werden. Durch die Ausführung in 2 Phasen ist die Reihenfolge der Commands beliebig, wenn sich alle Commands an die Konvention halten.

Um eine komplette Command-Hierarchie zu erzeugen, wird zunächst von einem Servlet ein Wurzel-Command erzeugt und dessen `buildCommandTree()`-Methode aufgerufen (siehe Abbildung 9). Diese ruft zuerst die `createCommands()`-Methode auf. In dieser Methode können Benutzereingaben verarbeitet und wenn nötig persistente oder temporäre Daten verändert werden. Danach werden Commands erzeugt, die später die Kindkomponenten erzeugen sollen. Diese Commands werden mit der `addSubCommand()`-Methode aus der `ComponentCommand`-Klasse den Childnamen zugeordnet, unter denen die von ihnen erzeugten Komponenten später in die Komponente des gerade ausgeführten Commands eingetragen werden. Danach können die `buildCommandTree()`-Methoden der neuen `ComponentCommand`s ausgeführt werden. Dadurch werden hierarchisch alle Commands für die resultierende Webseite des Requests erzeugt und Benutzereingaben verarbeitet. Die Verarbeitung von Benutzereingaben geschieht somit immer in dem Command, das die Komponente erzeugt hat, aus der die Eingaben stammen. Dadurch werden alle Funktionen, die eine Komponente betreffen, in einem Command zusammengehalten und sind für die Entwickler leicht zu überschauen.

Nach der Erzeugung aller Commands wird vom Servlet die `buildComponentTree()`-Methode des Wurzel-Commands aufgerufen (siehe Abbildung 10). Diese bekommt als Parameter eine vom Servlet erzeugte Wurzelkomponente und einen Namen für eine Kindkomponente. Die Methode prüft zuerst, ob in der übergebenen Komponente bereits ein Kind unter dem angegebenen Namen eingetragen wurde. Ist dies der Fall, wird der Typ der Kindkomponente geprüft, da der Komponentenbaum für eine Kindkomponente bereits aus einem vorherigen Request eine Kindkomponente eines anderen Typs enthalten könnte. Wenn der Typ mit dem erwarteten Typ des Commands, den die `buildComponentTree()`-Methode über die `getComponentClass()`-Methode abfragen kann, übereinstimmt, dann wird die `updateComponent()` Methode mit der bereits vorhandenen Kindkomponente aufgerufen. Diese kann dann anhand des Servlet-Requests und der Historie der Servlet Anwendung entscheiden, welche Attribute der Komponente verändert werden müssen. Ist keine Komponente vorhanden oder vom falschen Typ, dann wird die `createComponent()`-Methode aufgerufen, um eine neue vollständig initialisier-

Abbildung 9: UML-Sequenzdiagramm der buildCommandTree() Methode



te Komponente zu erzeugen. Diese wird dann als Kindkomponente eingetragen, wobei sie unter Umständen eine bereits vorhandene Komponente vom falschen Typ ersetzt. Abschliessend wird die `buildComponentTree()`-Methode der in der ersten Phase erzeugten Commands mit der zuvor verwendeten oder erzeugten Komponente und dem zu dem Command gehörigen Childnamen aufgerufen.

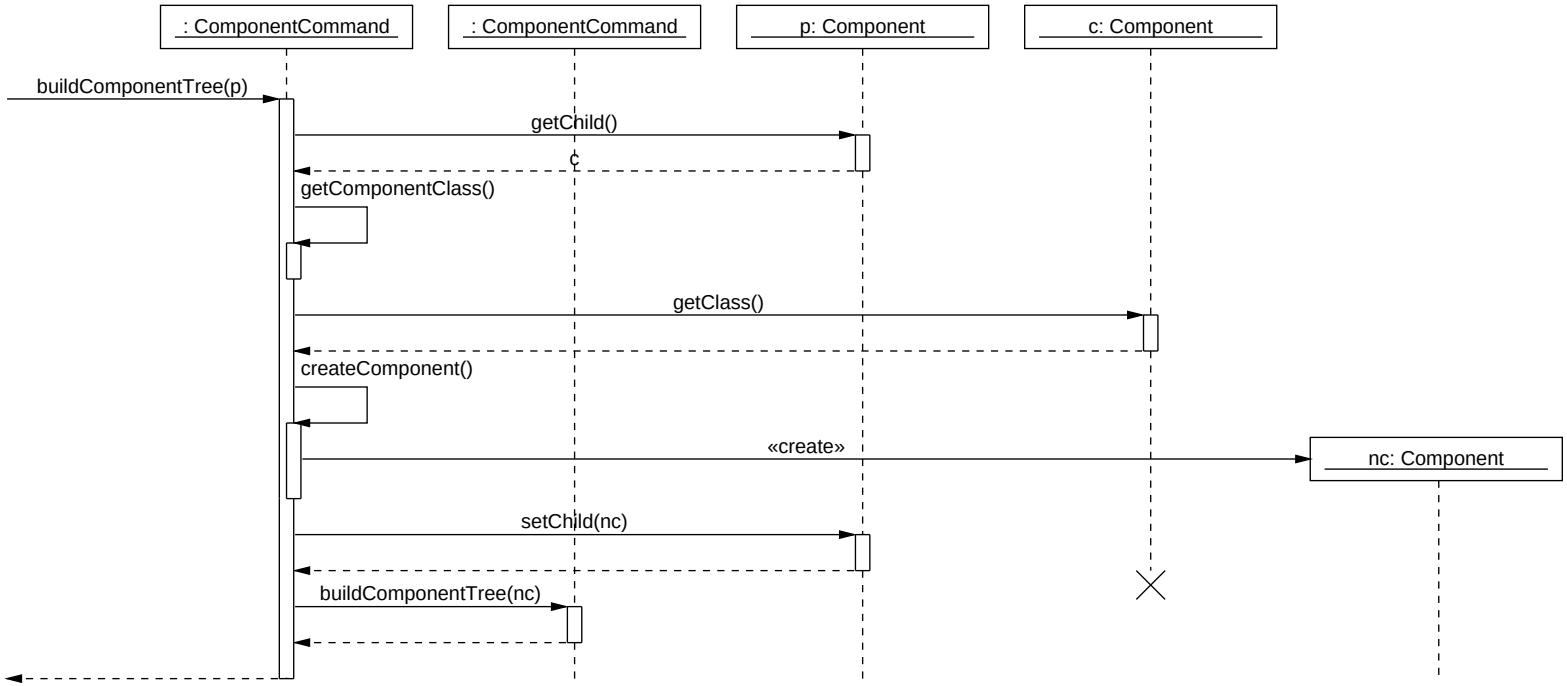
Wird die `buildComponentTree()`-Methode eines Commands aufgerufen, wird von dieser eine Komponente unter dem Childnamen eingetragen und alle weiteren Commands ausgeführt, so dass immer ein vollständiger Komponentenbaum entsteht. Durch die Typprüfung der Komponenten in jedem Command wird sichergestellt, dass im Endeffekt der Baum immer aus den für den Request erwarteten Komponenten besteht, unabhängig davon, in welchem Zustand sich der Komponentenbaum vorher befand.

Die Wurzelkomponente und damit der gesamte Komponentenbaum wird vom Servlet in der Session abgelegt, so dass der Komponentenbaum bei weiteren Requests wiederverwendet werden kann. Das Servlet muss dann bei weiteren Requests nur noch das erste Command erzeugen und dessen `buildCommandTree()`- und `buildComponentTree()`-Methoden aufrufen, um den Komponentenbaum und alle in ihm enthaltenen Komponenten in einen neuen Zustand zu überführen.

4.3 Ajax Unterstützung für Web Components

Die zuvor beschriebene Architektur erlaubt eine einfache Umsetzung von Ajax unterstützten Seiten. Dazu sind nur die im folgenden aufgeführten geringfügigen Erweiterungen notwendig.

Abbildung 10: UML-Sequenzdiagramm der buildComponentTree() Methode für den Fall einer vorhandenen Kindkomponente mit falschem Typ



4.3.1 Erweiterung der Komponenten

Für eine Aktualisierung einzelner Komponenten mit Ajax müssen diese wissen, ob sie im Laufe eines Requests verändert wurden. Dazu wird der `Component`-Klasse ein `boolean`-Flag hinzugefügt, das diesen Status enthält. Die Setter-Methoden der konkreten Komponenten müssen diesen Flag setzen, wenn ein Attribut auf einen veränderten Wert gesetzt wird. Außerdem wird der Status von der `Component`-Klasse selbst gesetzt, wenn die Kindkomponenten der Komponente verändert werden.

Des weiteren wird eine Methode benötigt, um diesen Status der Komponenten abzufragen und wieder auf „unverändert" zurückzusetzen. Das Zurücksetzen einer Komponente soll dessen Kindkomponenten mit zurücksetzen, da diese bei der Ausgabe einer Komponente immer mit ausgegeben werden.

Bei der Ausgabe der Komponenten wird durch die TagLib zu jeder Komponente eine AjaxAnywhere Zone definiert. Zu jeder Komponente wird der AjaxAnywhere Zone dafür ein eindeutiger Namen zugewiesen, damit Komponenten mehrfach verwendet werden können. Dieser kann auf beliebige Art erzeugt werden. Er muss nur bei folgenden Requests, in dem die gleiche Komponente aus der Session wieder vorkommt, gleich sein. Eine Möglichkeit hierfür ist es zum Beispiel, die `System.identityHashCode()`-Methode zu verwenden.

4.3.2 Integration im Servlet

Das Servlet kann wie bei Requests, bei denen eine komplette Seite geladen wird, die Verarbeitung des Requests durch die Commands anstoßen. Bei einem Ajax-Request sollte der letzte Komponentenbaum bereits in der Session gespeichert sein. Der Veränderungsstatus der Komponenten wurde dabei beim letzten Request am Ende zurückgesetzt. Somit haben nach dem Durchlauf der Commands für den Ajax-Request nur die Komponenten, die sich verändert haben, den Veränderungsstatus gesetzt.

Im Servlet können dann alle Äste des Baumes durch eine Tiefensuche über den Baum herausgesucht werden, die durch den Request verändert wurden. Wird eine Komponente mit verändertem Status gefunden, wird diese in eine Liste eingetragen. Die Suche muss dann an diesem Ast nicht weiter in die Tiefe fortgesetzt werden. Am Ende der Suche hat das Servlet dann eine flache Liste aller Äste des Komponentenbaumes, die verändert wurden. Diese Liste kann dann in den Servlet Request eingetragen werden und der Request an eine spezielle Java-Server-Page für Ajax-Requests weitergeleitet werden. Zuvor wird der Veränderungsstatus aller Komponenten der Liste und ihrer Kindkomponenten wieder auf unverändert zurückgesetzt.

Sollte widererwarten bei einem Ajax-Request keine Session gefunden werden, weil diese vielleicht bereits abgelaufen ist, muss der Browser an die URL des Ajax-Request umgeleitet werden, um die Seite komplett neu zu laden. Dies ist nötig, da Komponenten über eindeutige AjaxAnywhere Zonennamen beim Browser identifiziert werden. Wenn eine Komponente im Baum durch eine andere ersetzt wird, muss für diese eine neue Zone definiert werden, denn diese neue Komponente hat einen anderen eindeutigen Namen. Um eine neue Zone

zu definieren, muss die jeweilige Elternkomponente aktualisiert werden. Dies geschieht bereits dadurch, dass bei Veränderungen der Kindkomponenten bei der Elternkomponente der Veränderungsstatus gesetzt wird. Mit der Wurzel des Komponentenbaumes ist dies aber nicht möglich, da sie keine Elternkomponente hat. Geht die Information über ihren Zonenname verloren, muss eine neue Seite erzeugt werden.

Die Information, dass der Browser auf eine neue Seite geschickt werden soll, kann in einem Ajax-Request nur an die Ajax-Engine übermittelt werden. Die Ajax-Engine muss dann den Browser an die URL weiterleiten. Dadurch kann die Seite für den Browser einmal komplett neu erzeugt werden. Das Umleiten des Browsers ist ohne Schwierigkeiten möglich, da sich die URL bei Ajax-Requests nicht von denen bei Requests für komplette Seiten unterscheidet. Die Ajax-Engine kann vom Servlet zur Umleitung also einfach die URL bekommen, an die der Ajax-Request gerichtet war. Der `AAFilter` von AjaxAnywhere erzeugt automatisch eine Ajax-Response für Umleitungen, wenn ein Servlet über die `ServletResponse` eine Umleitung einstellt. Andere Umleitungen die das Servlet erzeugen könnte, werden also von AjaxAnywhere korrekt weitergeleitet und erfordern keine gesonderte Behandlung im Servlet.

4.3.3 Java-Server-Pages zur Aktualisierung der Components

Da die Komponenten und ihre Java-Server-Pages unabhängig von ihren Elternkomponenten sind, können sie problemlos ohne diese als HTML ausgegeben werden. Um eine Ajax Response zu erzeugen, kann also eine Java-Server-Page benutzt werden, die einfach die Komponentenliste aus dem Servlet durchgeht und jede dieser Komponenten mit der TagLib einbindet. Das Ergebnis ist dann eine HTML Seite, die nur die veränderten Komponenten nacheinander enthält. Durch die TagLib werden die Komponenten wieder als Ajax Anywhere Zonen mit dem gleichen Namen wie bei vorherigen Requests markiert.

Die veränderten Komponenten können dadurch von der Ajax-Engine aus der Seite extrahiert und an den entsprechenden Stellen im HTML Baum, die als AjaxAnywhere Zonen markiert sind, der bereits vorhandenen Webseite im Browser eingesetzt werden. Diese Stellen im HTML Dokument entsprechen dabei der Position der Komponenten im Komponentenbaum des Servlets, so dass das Ergebnis dieser Ersetzung eine Webseite ist, die der entspricht, die sich ergeben hätte, wenn der gesamte Komponentenbaum ausgegeben worden wäre.

Ausgehend von dieser neuen Webseite kann also wieder ein Request getätigt werden, der Komponenten aktualisiert und deren HTML-Ausgabe über Ajax im Browser ersetzt.

4.3.4 Integration der Ajax Engine

Zum Starten von Ajax-Requests muss abschließend noch das JavaScript der Ajax-Engine in den Header des HTML-Dokumentes eingebaut werden und Links und Formulare entsprechend mit `onclick`-Event-Handlern versehen werden.

Da sich die Request Parameter für diese Architektur bei Ajax-Requests nicht

von normalen Requests unterscheiden, bietet es sich an, statt den normalen HTML `<a ...>`-Tags eine TagLib zu verwenden, die `<a ...>`-Tags mit einem entsprechenden `onclick`-Event-Handler erzeugt, der die nötigen Funktionen der Ajax-Engine aufruft. Auf diese Weise lässt es sich einfach erreichen, Ajax in der Web Anwendung zu deaktivieren, indem der Tag der TagLib entsprechend konfiguriert werden kann.

4.4 Testbarkeit

Um Verifikation der Korrektheit der einzelnen Komponenten und Commands, die für eine Anwendung geschrieben werden, zu ermöglichen, müssen diese testbar sein. Für Unit Tests müssen die Funktionen einzelnen Klassen isoliert mit fiktiven Eingaben ausführbar und die Ergebnisse überprüfbar sein. Ein Durchlauf des gesamten Servlets ist zum Testen meistens ungeeignet, da dieser zu lange dauert und viel zu viele mögliche Abläufe hat, um alle zu testen.

4.4.1 Komponenten

Da die `Component`-Klassen Java Beans sind, speichern diese nur Daten. Ein Test braucht also nur überprüfen, ob diese Daten, die eingetragen werden, korrekt wiedergeben werden. Bei Beans, die aus den Daten neue Werte berechnen, können die Ergebnisse von Berechnungen überprüft werden. Die Implementierung der Abläufe in der `Component`-Oberklasse lässt sich testen, indem eine Unterklasse zum Testen angelegt wird, über die die `protected`-Methoden der abstrakten `Component`-Klasse aufgerufen werden können.

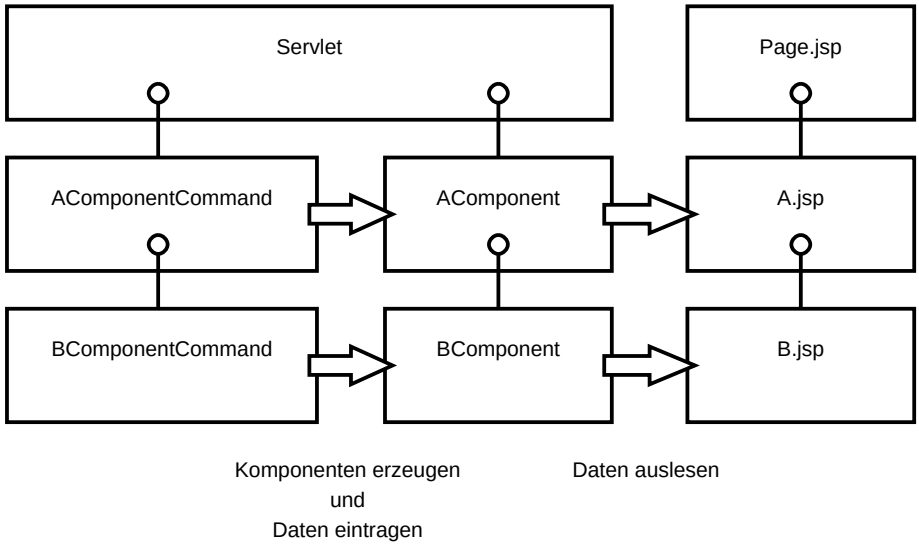
Die zu der Komponente gehörende Java-Server-Page kann getestet werden, indem durch ein Test-Servlet eine Komponenten Java Bean ohne Kindkomponenten erzeugt, diese in den Request einträgt und die zu der Bean gehörige Java-Server-Page eingebunden wird. Dadurch wird nur die Darstellung der Komponente erzeugt und es kann sichergestellt werden, dass diese die Daten aus der Bean enthält und bestimmte Kriterien an die HTML Struktur erfüllt, wie zum Beispiel die Anzahl der Zeilen einer Tabelle, wenn diese sich aus Daten der Bean ergibt. Außerdem wird so geprüft, ob sich die Java-Server-Page vom Servlet Container kompilieren lässt und keine Syntax-Fehler enthält.

4.4.2 ComponentCommands

Als letztes müssen die ComponentCommands getestet werden. Die Funktion der `ComponentCommand`-Klasse selber lässt sich wieder testen, indem eine spezielle Unterklasse zum Testen angelegt wird, die prüft, ob die abstrakten Methoden korrekt aufgerufen werden, wenn die Implementierung in der `ComponentCommand`-Klasse aufgerufen wird.

Die `ComponentCommand`s für die Anwendung selbst implementieren nur die vier abstrakten Methoden. Diese können einzeln aufgerufen werden und ihr Ergebnis getestet werden. Da die `ComponentCommand`s für die Kindkomponenten nicht direkt ausgeführt werden, sondern nur angelegt werden, muss kein kompletter Durchlauf der Anwendung getestet werden. Jedes `ComponentCommand` is für sich isoliert testbar.

Abbildung 11: Übersicht über die Architektur



4.5 Zusammenfassung

Die Architektur (siehe Abbildung 11) zerlegt eine Webseite in kleine Teile: die Komponenten. Diese bestehen nur aus einer Java Bean und einer dazugehörigen Java-Server-Page. Da die Java-Server-Page einzig auf Daten aus der Bean zugreifen muss, sind die Komponenten vom Rest der Seite vollkommen unabhängig. Eine neue Komponente kann so völlig unabhängig von Änderungen am Servlet entwickelt und getestet werden. Zur Einbindung in eine Webseite, die bereits aus Komponenten besteht, müssen dann nur noch ein oder mehrere `ComponentCommand`s, wenn es verschiedene Anwendungszwecke für die Komponente gibt, geschrieben werden, die die Komponente erzeugen, aktualisieren, `ComponentCommand`s für weitere Kindkomponenten anlegen und Eingaben des Benutzers aus Formularen in der Komponente verarbeiten. Diese `ComponentCommand`s können unabhängig vom Rest des Servlets zunächst getestet werden. Abschließend kann es dann in anderen `ComponentCommand`s erzeugt und verwendet werden.

Durch diese Architektur zerfällt ein Servlet und die von ihr erzeugte Webseite in viele kleine, einfache und überschaubare Bausteine, was die Weiterentwicklung einer komplexen Website mit vielen Entwicklern unterstützen sollte.

5 Prototypische Realisierung

In diesem Abschnitt soll die zuvor beschriebene Architektur im JCommSy umgesetzt werden. Um eine weitere Rubrik zum JCommSy zu migrieren, wurde hierfür die Ankündigungsrubrik ausgewählt. Diese wird dann in die bisher verwendete JCommSy Seite eingebunden. Eine Umsetzung der gesamten Seitenstruktur würde erfordern, dass die bisher vorhandene Terminrubrik mit umgesetzt wird, was den Rahmen dieser Arbeit sprengen würde.

5.1 ComponentServlet

Die bisherige JCommSy Architektur sieht vor, dass vom Einstiegsservlet, dem `CommsyServlet` der Request an ein `RubricServlet` weitergeleitet wird. Dafür sollte für jede Rubrik ein Servlet geschrieben werden. Für die Komponenten-Architektur ist dies nicht mehr sinnvoll, da die gesamte Seite über `ComponentCommand`s aufgebaut werden soll. Verzweigungen können dann innerhalb der `ComponentCommand`s realisiert werden. Die Funktion des Servlets besteht nur noch darin, die Wurzelkomponente und ein `ComponentCommand` zu erzeugen und den rekursiven Aufruf der Commands zu starten.

Danach kann das `ComponentServlet` bei Requests für eine vollständige Webseite den Request an die CommSy-Java-Server-Page weiterleiten. Statt wie sonst eine Java-Server-Page für die Rubrik-Seite im Request anzugeben wird stattdessen immer eine spezielle Java-Server-Page angegeben, die die Wurzelkomponente über die TagLib einbindet. Dadurch wird die gesamte Komponenten-Hierarchie dann ausgegeben.

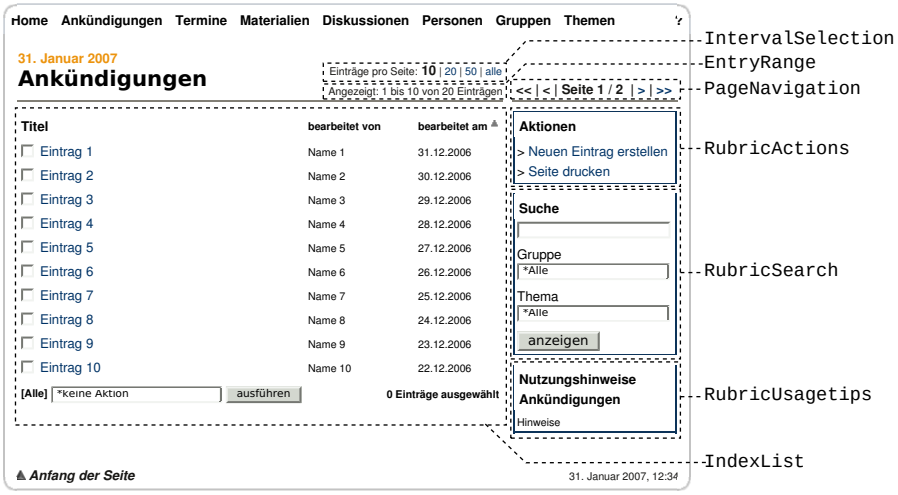
Bei Ajax-Requests wird der Request stattdessen an eine spezielle Ajax-Java-Server-Page weitergeleitet. Wie im Kapitel 4 beschrieben, wird dazu aus dem Komponenten-Baum eine Liste mit veränderten Komponenten erzeugt, die dann in der Java-Server-Page nacheinander eingebunden werden.

Solange die Verwendung der Komponenten-Architektur nicht auf den gesamten Raum ausgeweitet werden kann, wird vom `ComponentServlet` das `AnnouncementCommand` erzeugt, das als erstes aufgerufen wird.

5.2 Komponenten

Für die Klassen der Komponenten wurde im JCommSy das Package `de.unihamburg.informatik.jcommsy.presentation.components` angelegt, das die Implementierung der `Component`-Klasse enthält, die als Oberklasse aller Komponenten dient. Für neue Komponenten muss eine Unterklasse der `Component`-Klasse in diesem Package erstellt werden. Dort können dann Attribute eingetragen werden, die selbst oder von ihnen abgeleitete Werte auf der Seite ausgegeben werden sollen. Hierfür müssen dann die entsprechenden Getter- und Setter-Methoden angelegt werden. Wenn die Komponente Kindkomponenten enthalten soll, muss hierfür eine String-Konstante angelegt werden, unter der die Konstante mit der `setChild()`-Methode aus der `Component`-Klasse eingetragen wird. Außerdem wird noch eine parameterlose Getter-Methode benötigt, die die `getChild()`-Methode mit dieser Konstante aufruft und das Ergeb-

Abbildung 12: Komponenten der RubricIndex Komponente



nis zurückliefert.

Zur Einbindung von Komponenten in die JCommSy-Seite und in andere Komponenten würde ein Include-Tag in der bereits vorhandenen JCommSy-TagLib hinzugefügt. Dieser bekommt als Parameter die Bean der Komponente. Diese wird dann an als Attribute `bean` im Request eingetragen und die zu der Komponenten-Bean gehörige Java-Server-Page aus dem `jsp/xhtml` eingebunden. Diese muss den Namen der Klasse mit der Endung `.jsp` haben.

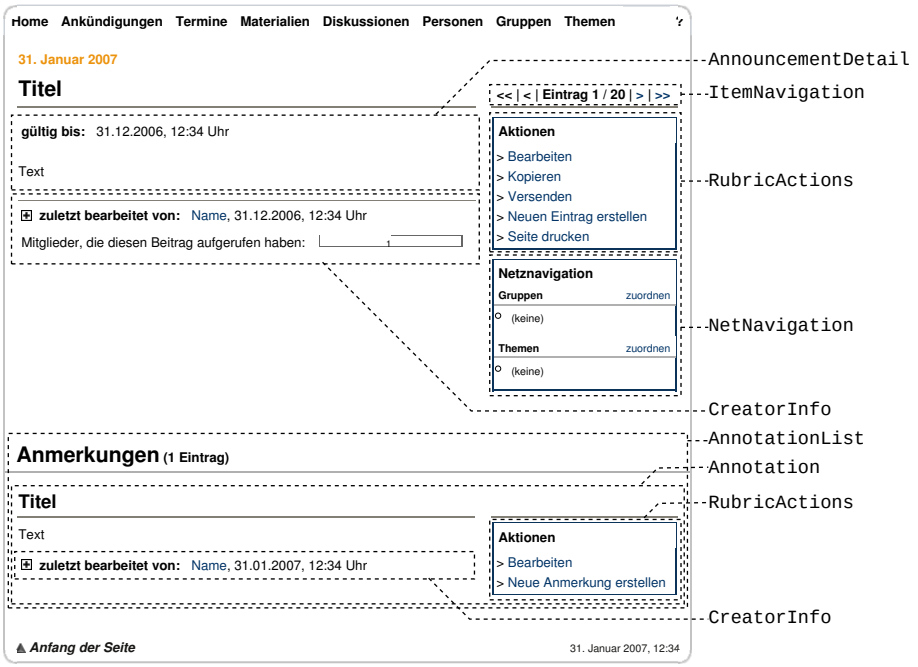
Innerhalb der Java-Server-Pages kann dann über die Expression-Language auf die Bean im Request und ihre Attribute zugegriffen werden. Hier über können die Beans für Kindkomponenten abgefragt und an weitere Include-Tags übergeben werden, um die Kindkomponenten einzubinden.

Die nächsten 3 Abschnitte geben eine kurze Übersicht über die Komponenten die für die Index-, Detail- und Editseite im Rahmen der Arbeit erstellt wurden und wo diese auf den Seiten eingesetzt werden. Alle Komponenten wurden wenn möglich so erstellt, dass sie nicht nur in einem speziellen Kontext eingesetzt werden können. Somit konnte für einige Elemente, wie zum Beispiel die Aktions-Box, die auf mehreren Seiten verwendet wird, die gleichen Komponenten verwendet werden. Dazu muss die Komponente nur durch die Commands mit anderen Daten befüllt werden.

5.2.1 Indexseite

Die Indexseite (siehe Abbildung 12) hat eine recht einfache Struktur, die nicht dynamisch ist. Somit wäre es möglich gewesen, diese in einer einzigen Komponente zu realisieren. Um die Komponenten möglichst übersichtlich zu halten, wurden dennoch einzelne Elemente der Indexseite als eigenständige Komponenten realisiert, obwohl dies dazu führt, dass diese teilweise die selben Daten enthalten. Im Gegenzug erhält man dafür kleine überschaubare HTML Fragmente für die Java-Server-Pages der Komponenten, die keine tiefe Verschachtelung durch etliche Fallunterscheidungen enthalten. Die Elemente der Indexseite, die

Abbildung 13: Komponenten der RubricDetail Komponente



auf anderen Seiten zu finden sind, sollten sowieso als Komponenten realisiert werden.

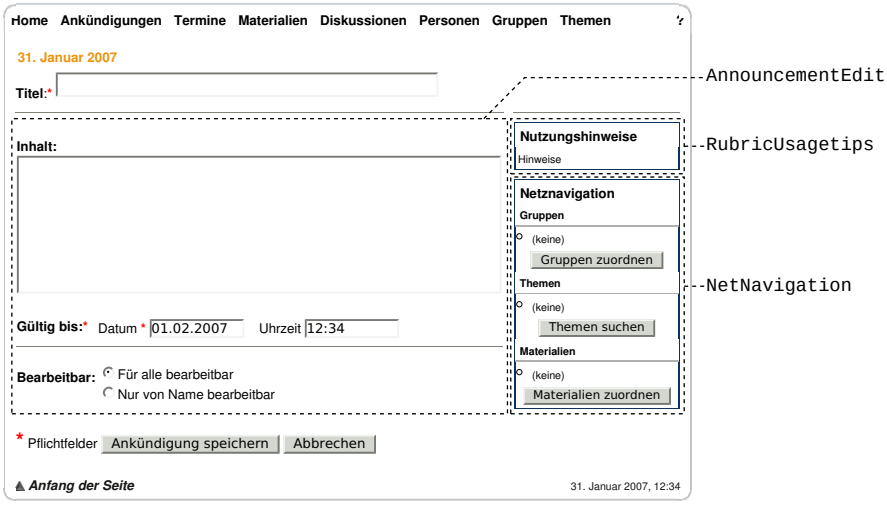
Somit erhält man für die Indexseite eine `RubricIndex`-Komponente, die den grundsätzlichen Aufbau der Indexseite enthält und die Komponenten für die einzelnen Elemente in der Indexseite positioniert. Die einzelnen Bausteine der Indexseite sind die Intervalauswahl (`IntervalSelection`), mit der man die Länge der Liste von Einträgen variieren kann, die Anzeige der aktuellen Position innerhalb der Liste (`EntryRange`), die Navigation zum Blättern innerhalb der Liste (`PageNavigation`), die Suchbox (`RubricSearch`) und die Liste der Einträge selbst (`IndexList`). Zusätzlich enthält die Indexseite noch die allgemeinen Komponenten für die Nutzungshinweise (`RubricUsagetips`) und eine Aktionsbox (`RubricActions`).

Die `IndexList`-Komponente wurden dabei so entworfen, dass sie für die Indexseiten aller Rubriken verwendet werden kann. Die Spalten der Liste sind nicht fest vorgegeben, und können dynamisch festgelegt werden.

5.2.2 Detailseite

Die Detailseite (siehe Abbildung 13) wird durch die `RubricDetail`-Komponente erzeugt. Diese enthält die einzelnen Komponenten für die Elemente der Detailseite. Allerdings hat die Detailseite keine feste Struktur, wie die Indexseite, die bis auf die Spalten der Liste für alle Rubriken gleich ist. Zum einen gibt es einen Bereich, in dem die speziellen Daten der Items für die Rubrik angezeigt werden. Dieser unterscheidet sich daher je nach Itemtyp, der angezeigt werden soll. Darum gibt es hierfür eine spezielle Komponente. Für die Ankündigungsrubrik

Abbildung 14: Komponenten der RubricEdit Komponente



ist dies die `AnnouncementDetail`-Komponente. Der Rest der Seite besteht aus allgemeinen Komponenten. Dazu gehören `RubricActions`, die bereits auf der Indexseite verwendet wurde, `NetNavigation`, `CreatorInfo`, `ItemNavigation` und `AnnotationList`.

`AnnotationList` ist jedoch, im Gegensatz zu den bisherigen Komponenten, eine Komponente, die eine Liste von Komponenten ausgeben kann. Dazu enthält die Komponente eine Liste mit `Annotation`-Kindkomponenten, die innerhalb der `AnnotationList`-Komponente, dann in einer Schleife ausgegeben werden. Die `Annotation`-Komponenten enthalten wieder `CreatorInfo` und `RubricActions`-Komponenten. Außerdem gibt es noch einen Spezialfall, der in der Abbildung nicht dargestellt ist. Wenn die Liste der Anmerkungen leer ist, soll eine Aktionsbox angezeigt werden. Da diese aber normalerweise Bestandteil der `Annotation`-Komponente ist, muss im Fall einer leeren Liste eine extra `RubricActions`-Komponente in der `AnnotationList`-Komponente eingetragen werden. Sind Anmerkungen vorhanden, dann soll diese Komponente jedoch nicht angezeigt werden.

5.2.3 Editseite

Die Editseite wird von der `RubricEdit`-Komponente erzeugt (siehe Abbildung 14). Diese benötigt wie die Detailseite eine spezielle Komponente für die Items (`AnnouncementEdit`). Zusätzlich enthält sie noch die bereits auf der Indexseite verwendete `RubricUsagetips`-Komponente und die `NetNavigation`-Komponente. Letztere wird hier allerdings in einem Editmodus verwendet. Dieser wurde in der gleichen Komponente realisiert, da sich der grundlegende Aufbau der Netznavigation nicht von der Detailseite unterscheidet.

5.3 ComponentCommands

Die `ComponentCommands` für das JCommSy befinden sich in Subpackages des `de.unihamburg.informatik.jcommsy.control`-Packages. Abstrakte und rubrikunabhängige Commands befinden sich im `common`-Subpackage und Commands für spezielle Rubriken in einem extra Package für jede Rubrik. Für die Ankündigungsrubrik ist die das `announcement`-Package.

Um ein neues Command zu erstellen, muss eine Unterklasse der `ComponentCommand`-Klasse im passenden Package erstellt und die 4 abstrakten Methoden implementiert werden. Wenn das Command Items oder Parameter benötigt, die es nicht selbst ermitteln kann oder wenn dies nicht sinnvoll ist, kann zusätzlich ein Konstruktor hinzugefügt werden.

Für die Hauptkomponenten der einzelnen Seiten wurden abstrakte Commands geschrieben. Diese implementieren die grundlegenden Funktionen und erstellen die Commands für die allgemeinen Kindkomponenten. Um eine Seite für eine spezielle Rubrik zu erstellen, muss dann nur noch eine konkrete Unterklasse für die Rubrik erstellt werden, die die abstrakten Methoden implementiert. So enthält zum Beispiel das `RubricIndexCommand` bereits das meiste, was für eine Indexseite benötigt wird. Es werden bereits die Commands für die Navigation, die Aktionen, die Nutzungshinweise etc. erstellt. Ein konkretes Command für eine Rubrik braucht nur noch das Command zu erstellen, das die eigentliche Indexliste erstellt, da diese sich in jeder Rubrik unterscheidet.

Für die meisten allgemeinen Komponenten gibt es ein Command, dass die nötigen Daten für die Komponente entweder selbst aus dem Request ermittelt oder diese im Konstruktor bekommt, wenn es nicht sinnvoll ist, diese mehrmals abzufragen. So können zum Beispiel nur alle Anmerkungen zu einem Item auf einmal vom `AnnotationService` abgefragt werden. Das `AnnotationList`-Command fragt also diese Liste für das Item ab und übergibt die einzelnen `Annotation`-Objekt über den Konstruktur an die `Annotation`-Commands.

Für einige Komponenten gibt es mehrere Commands, da diese in verschiedenen Kontexten benutzt werden, wo es keinen Sinn macht, alle verschiedenen Verwendungszwecke der Komponente durch Parametrisierung eines einzigen Commands zu realisieren. Ein Beispiel hierfür ist die `RubricActions`-Komponente. Für diese gibt es ein `RubricIndexActionsCommand` für die Indexseite, ein `RubricDetailActionsCommand` für die Detailseite und ein `AnnotationActionsCommand` für die Anmerkungen.

6 Ergebnis

6.1 Zusammenfassung

Im Abschnitt 2 wurde dann das JCommSy-Projekt vorgestellt. Als erstes würde hierfür ein Überblick über die JCommSy Architektur gegeben, die aus einer Persistenzschicht, fachlichen Services und Servlets besteht. Die Servlets wurde danach im Detail vorgestellt, da sie für die spätere prototypische Einbindung von Ajax relevant sind.

In Abschnitt 3 wurden zuerst die Technologien auf denen das World Wide Web basiert erläutert und welche Probleme sich durch diese für das Gestalten interaktiver Websites ergeben. Gleichzeitig bilden diese jedoch die Grundlage von Ajax. Wie Ajax diese Technologien verknüpft, um die vorhandenen Probleme zu lösen, wurde danach erläutert. Hierfür wurde das `XMLHttpRequest` Objekt von JavaScript eingeführt und anhand eines Beispiels gezeigt, wie es benutzt werden kann, um asynchron Daten aus einem JavaScript abzufragen.

Dann wurde im Abschnitt 4 eine Web Component Architektur entwickelt, die den Anforderungen genügt, die für einen Einsatz im JCommSy benötigt werden, und eine einfache Integration von Ajax erlaubt. Die hierfür benötigten Klassen, Java-Server-Pages und TagLibs wurden vorgestellt und im einzelnen erläutert, warum die Architektur in dieser Weise gestaltet wurde.

Eine Komponente besteht dabei jeweils aus einer Java Bean, die benötigte Daten enthält, und einer Java-Server-Page, die die Daten in HTML ausgibt. Über eine TagLib werden die Komponenten eingebunden, ohne das die Komponente über den Kontext in dem sie verwendet wird, etwas wissen muss. So können Komponenten andere Komponenten enthalten oder in anderen Komponenten eingebunden werden. Die Architektur erlaubt es außerdem einzelne Komponenten mit Ajax zu aktualisieren. Die Architektur verändert somit die Benutzung der Webseite nicht, sondern beschleunigt diese nur. Neue Funktionen die nur mit Ajax möglich sind, wie das automatische Vervollständigen von Eingaben, werden nicht unterstützt.

6.2 Ausblick

HTTP Anfragen an einem Webserver aus einem JavaScript asynchron durchzuführen bietet viele neue Möglichkeiten. Da die Empfangenen Daten vom JavaScript selbst verarbeitet werden müssen, können sowohl Daten in einer strukturierten Form, HTML-Fragmente oder JavaScript bei bedarf heruntergeladen werden. Es kann somit also jede Art von Web-Anwendung realisiert werden, die mit den Gestaltungsmöglichkeiten von HTML und Cascading Style Sheets und den Programmiermöglichkeiten von JavaScript realisiert werden kann. Hierbei kann ein Großteil der Arbeit sowohl von Client erledigt werden, der den Webserver nur noch wie eine Art Web Service kontaktiert um Daten abzufragen, oder vom Server, der Änderungen an den Client vorgibt. Ob man die eine oder andere Möglichkeit nutzt oder beide kombiniert, bleibt dabei dem Ersteller der Website überlassen.

Für herkömmliche Websites bringt Ajax nicht viel neues, außer einer verbesserten Interaktion, die einem eher das Gefühl vermittelt, man würde mit

einer Anwendung arbeiten, anstatt sich von einer Webseite zur nächsten weiter zu klicken. Solange der Datenbestand der Website wenig dynamisch ist, wird dies so bleiben. Wenn der Inhalt einer Website jedoch von den Surfern kontinuierlich verändert wird, dann wird sicher Ajax mehr Nutzen bringen. In Zukunft wird Ajax sicherlich durch mächtige Frameworks unterstützt werden, die die Entwicklung von Websites vereinfachen werden und für den Entwickler die eigentliche Arbeitsweise von Ajax abstrahieren und höhere Funktionen zur verfügung stellen.

Ajax befindet sich noch in der Erprobungsphase. Es ist eine Basistechnologie, die für verschiedene Zwecke eingesetzt werden kann. Aus diesem Grund beschränkt sich die Vorstellung von Ajax in dieser Arbeit erstmal auf die Grundlagen und die Integration in eine vorhandene Web Anwendung. Diese begrenzt den Einsatz von Ajax derzeit noch recht stark. Eine Anwendung von Grund auf, auf Ajax aufzubauen würde sicherlich wesentlich größere Möglichkeiten bieten. Auszuloten welche Möglichkeiten Ajax in Zukunft noch bieten wird, würde den Rahmen der Arbeit jedoch sprengen.

7 Literatur

[Aja] *AjaxAnywhere*. `http://ajaxanywhere.sourceforge.net/`

[BPSM+06] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0*. `http://www.w3.org/TR/2006/REC-xml-20060816`. Version: 2006

[Fer04] Ferner, Jens: *PHP 5 Referenz*. Data Becker, 2004. – ISBN 3815823692

[FGM+99] Fielding, Roy T. ; Gettys, Jim ; Mogul, Jeffrey C. ; Frystyk Nielsen, Henrik ; Masinter, Larry ; Leach, Paul J. ; Berners-Lee, Tim: *Hypertext Transfer Protocol — HTTP/1.1*. Internet proposed standard RFC 2616, Juni 1999

[Fla02] Flanagan, David: *JavaScript - das umfassende Referenzwerk. 2. Auflage*. O'Reilly, 2002. – ISBN 3897213303

[Gar05] Garrett, Jesse J.: *Ajax: A New Approach to Web Applications*. `http://www.adaptivepath.com/publications/essays/archives/000385.php`. Version: Februar 2005

[GHJV00] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns Elements of Reusable Object-Oriented Software*. Massachusetts : Addison-Wesley, 2000. – ISBN 0201633612

[HHW+04] Hors, Arnaud L. ; Hégaret, Philippe L. ; Wood, Lauren ; Nicol, Gavin ; Robie, Jonathan ; Champion, Mike ; Byrne, Steve: *Document Object Model (DOM) Level 3 Core Specification*. `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`. Version: 2004

[Kes06] Kesteren, Anne van: *The XMLHttpRequest Object*. `http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/`. Version: 2006

[MS04] Meinel, Christoph ; Sack, Harald: *WWW - Kommunikation, Internetworking, Web-Technologien*. Springer-Verlag, 2004. – ISBN 3540442766

[Pat02] Patzer, Andreas: *JSP Examples and Best Practices*. Computer Bookshops, 2002. – ISBN 1590590201

[RHJ99] Raggett, Dave (Hrsg.) ; Hors, Arnaud L. (Hrsg.) ; Jacobs, Ian (Hrsg.): *HTML 4.01 Specification*. World Wide Web Consortium, Dezember 1999

[RJB98] Rumbaugh, Jim ; Jacobson, Ivar ; Booch, Grady: *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. – ISBN 020130998X

[Sha02] SHANNON, Bill: *Java 2 Platform, Enterprise Edition (J2EE), 1.4 Specification.* `http://java.sun.com/j2ee/j2ee-1_4-pfd3-spec.pdf`. Version: 2002