Thema:

Webtechnologien (AJAX/CSS)

Ausarbeitung im Rahmen der Seminarphase der Projektgruppe
Modulares Enterprise Architecture Management System

Themensteller: Dr. H.-J. Appelrath
Betreuer: Dr. Ulrike Steffens, Jan Stefan Addicks, Matthias Postina

vorgelegt von: Yu Liu
Oldenburg
Email: Yu.Liu@Informatik.Uni-Oldenburg.de

Abgabetermin: 26.10.2007

Inhaltsverzeichnis

1. Einführung

In den letzten Jahren nahm die Nutzung des Internets schnell zu. Immer mehr Internetbenutzer haben den immer stärkeren Wunsch nach einer interaktiveren und umfangreicheren Nutzung von Webseiten. Dieser Wunsch kann durch das klassische Model der herkömmlichen Webapplikation leider nicht erfüllt werden. Ajax ist zurzeit ein Modewort, das zuerst von Jesse James Garrett in seinem Artikel über ein neues Konzept zur Entwicklung von Webapplikationen erwähnt wurde. Seitdem hat Ajax schon innerhalb kürzester Zeit große Verbreitung gefunden. Diese neue Technik folgt den Bedürfnissen von Internetbenutzern und gibt dem Web neue Interaktionsmuster, die häufig bei lokalen Desktop-Applikationen verwendet werden: Nebenläufigkeit, asynchrone Aktualisierung und visuelle Interaktion machen Webapplikationen dynamischer und lassen sie ohne lange Verzögerungen ablaufen. Sie schließen somit die Lücken zwischen Desktop- und Webapplikationen.

In dieser Ausarbeitung werden die in Ajax verwendeten Technologien erläutert und sowie auf mögliche Vorteile und Nachteile eingegangen.

2. Grundlagen zu Ajax

2.1. Was ist Ajax?

Ajax ist an sich keine neue Technologie im Bereich des Webs, sondern vielmehr eine Ansammlung von vielen Basistechnologien. Hat man sich längere Zeit mit der Entwicklung der Webanwendungen beschäftigt, ist man bereits mit den meisten der Basistechnologien vertraut. Die schicke Abkürzung Ajax steht für „Asynchronous JavaScript And XML“ und setzt auf die folgenden Basistechnologien auf. Jede Technologie spielt ihre eigene Rolle in einer Ajax-Applikation. Im Folgenden werden die wichtigsten Basistechnologien zusammengestellt und ihre Rolle kurz erläutert.

Technologie	Rolle
XHTML und Cascading Style Sheet (CSS)	standardgerechte Präsentation
XML	Datenaustausch
Document Object Model (DOM)	dynamische Ausgabe und Interaktion
XMLHttpRequest-Objekt	asynchrone Datenabfragen
JavaScript	bindet alles zusammen

Tabelle 2-1: Basistechnologien und ihre Rolle

Wir werden später auf die einzelnen Technologien und ihre Rolle eingehen.

2.2. Aufbau

Im Folgenden werden zuerst die Interaktionsmodelle der klassischen Web-Applikation und der Ajax-Applikation sowie ihre Unterschiede vorgestellt, dann gehen wir auf die Vor- und Nachteile bei der Verwendung von Ajax in Web-Applikationen ein.

2.2.1. Klassisches Interaktionsmodell

Herkömmliche Webapplikationen werden meistens nicht auf dem Computer des Anwenders installiert, sondern basieren auf der Interaktion zwischen Server und Client, während sich Desktop-Applikationen hingegen auf dem Computer des Anwenders befinden und auch dort funktionieren.

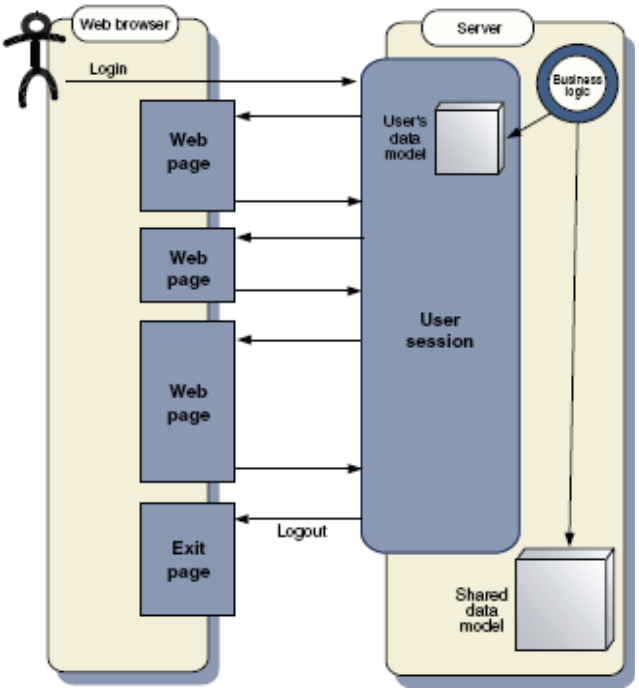

Abbildung 2-1: Klassisches Interaktionsmodell (Quelle[1])

Bisher waren Webapplikationen durch die seitenbasierte Anzeige der Inhalte beschränkt. In den meisten Fällen navigiert der Nutzer, nachdem er sich bei einer Webapplikation anmeldet hat, zwischen verschiedenen HTML-Dokumenten hin und her. Wird eine neue HTML-Seite angezeigt, dann läuft eine traditionelle HTTP-Anfrage wie im Folgenden ab:

1. Der Nutzer interagiert auf einer HTML-Seite, er klickt z.B. einen Link oder einen Sende-Button an.
2. Der Browser schickt eine HTTP-Anfrage an den Server.
3. Der Server verarbeitet die HTTP-Anfrage,
4. kommuniziert mit externen Ressourcen,

[1] Quelle: CRANE, Dave; PASCARELLO, Eric; JAMES, Darren: *Ajax in Action*. Manning Verlag, 2006, S. 18-19

5. bereitet Daten auf und führt Validierungen und Berechnungen aus,
6. füllt eine frische HTML-Seite aus und antwortet dem Client damit.
7. Der Browser empfängt diese neue HTML-Seite und stellt sie dem Nutzer dar.

Wie oben gezeigt, müssen Webapplikationen ständig über das Internet kommunizieren. Sie sind daher von Natur aus langsamer als Desktop-Applikation. Zu beachten ist aber noch, dass die komplette HTML-Seite häufig neu geladen wird, obwohl nur die Aktualisierung kleiner Teile erforderlich ist. Zudem steht das System in der Zeit zwischen Anfrage und Antwort still, also es können also clientseitig keine Berechnungen, Validierungen oder ähnliches durchgeführt werden.

Das klassische Model – seitenbasierte Anzeige von HTML‐Dokumenten – hat seine Hintergründe. Einerseits verfügten die früheren clientseitigen Rechner nicht über hohe Rechenfähigkeit und große Arbeitsspeicher, anderseits war damals die Bandbreite des Internets auch niedrig. In diesem Fall, sollte der Server außer HTML‐Dokumenten nicht vielen Daten und Skriptcodes an Clients mitschicken und somit das Netzwerk und die Clients belasten. Nun ist es aber anders. In den letzten 3 oder 4 Jahren wurde die Bandbreite des Internets erhöht und ebenso die Rechenfähigkeit und Größe des Arbeitsspeichers der clientseitigen Rechner. Deshalb sind sie schon in der Lage, von sich aus den Internetzbenutzern interaktive und umfangreiche Nutzung anzubieten.

2.2.2. Das Interaktionsmodell einer Ajax-Applikation

Was Ajax uns bringt, ist nicht nur das Modewort, das ein Sammelbegriff einer Reihe von Technologien ist, sondern auch eine neue Architektur, die uns aufweist, wie Websites aufgebaut werden sollen. Mit dieser Architektur werden die Lücken zwischen Desktop- und Webapplikation geschlossen und dem Programmierer wird es erleichtert, in Web-Anwendungen das Verhalten einer Desktop-Applikation nachzuahmen. Diese Architektur hat die folgenden wichtigsten Eigenschaften:

1. Integration neuer Informationen: Schicken Anwendungen Anfragen an den Server, bekommen ein paar Informationen zurück und integrieren sie mit der gerade betrachteten Seite, indem sie deren Dom ändern. Eine Komplette neue Ladung der Seite ist hier nicht mehr erforderlich.
2. Asynchrone Interaktion: Anwendungen interagieren die meiste Zeit asynchron mit dem Server. Das heißt, dass der Browser durch die an den Server geschickten Anfragen nicht mehr blockiert wird und das System in der Zeit zwischen Anfragen und Antworten also nicht still stehen muss. Das hat den Vorteil, dass der Nutzer in der Zwischenzeit andere Teile der Anwendung betätigen kann.
3. Umfangreiche Ereignisse: Fast alle Aktionen des Nutzers können Anfragen an den Server auslösen. Moderne Browser wie Internet Explorer und Firefox bieten fast alle Ereignisse, die eine reguläre Desktop-Applikation bietet. Ajax geht den Schritt bis hin zum reichhaltigen Client, indem alle vom Nutzer ausgelösten Ereignisse in asynchrone Anfragen an den Server umwandelt werden können.

Im Vergleich zu einer traditionellen Web-Applikation sieht das Interaktionsmodell einer Ajax-Applikation wie im Folgenden aus:

1. Zu Begin wird nach der Anmeldung eine HTML-Seite mit einem hohen Anteil an JavaScript-Code geladen.
2. Der Browser stellt die Seite gemäß ihrem DOM-Baum dar.
3. Der Nutzer interagiert auf einer HTML-Seite und jede Aktion von ihm könnte eine asynchrone Anfrage an den Server auslösen.
4. Der Server verarbeitet die HTTP-Anfrage,
5. kommuniziert mit externen Ressourcen
6. bereitet Daten auf und führt Validierungen und Berechnungen aus,,
7. antwortet dem Client mit neuen Daten.
8. Der Browser gibt die Daten an eine Call-Back-Funktion in der vorhandenen Seite zurück.
9. Die neuen Daten werden analysiert und in die betrachtete Seite integriert, indem der DOM-Baum geändert wird.
10. Der Browser macht die Änderung sichtbar.

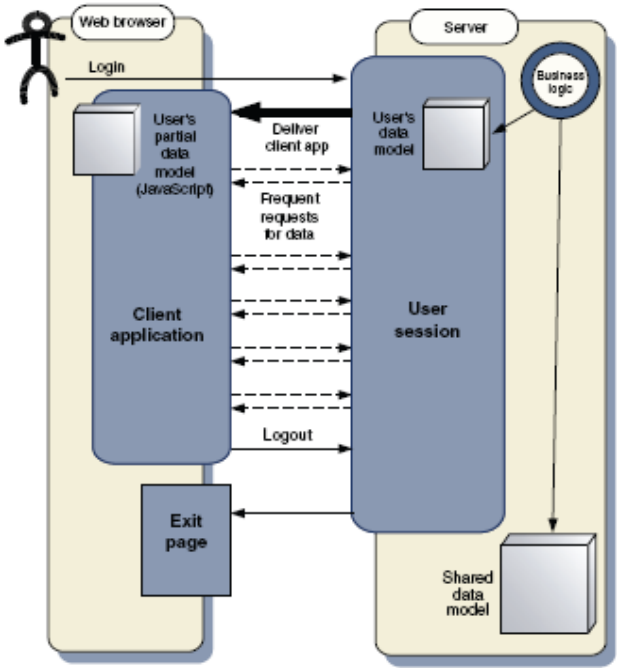

Abbildung 2-2: Interaktionsmodell einer Ajax-Applikation(Quelle[1])

2.2.3. Vergleich der zwei Interaktionsmodelle

Im Folgenden werden die Eigenschaften der zwei Interaktionsmodelle zusammengestellt:

Eigenschaft	Traditionelle Web-Applikation	Ajax-Applikation
zwischen Anfrage und Antwort	still	aktiv
Neuladen der HTML-Seiten	häufig	selten
Sitzungshaltung	serverseitig	clientseitig
Anteil an JavaScript-Code	niedrig	hoch
Datenaufkommen	annähernd gleich	anfangs hoch
Datenredundanz	hoch	niedrig
Interaktivität	niedrig	hoch
Clientseitige Applikationslogik	ganz auf dem Server	teilweise auf dem Client

Tabelle 2-2: Vergleich der zwei Interaktionsmodelle

Wie im letzten Abschnitt genannt, steht eine traditionelle Web-Applikation in der Zeit zwischen Anfrage und Antwort still, während eine Ajax-Applikation inzwischen aktiv bleibt. Betrachtet man eine traditionelle Web-Anwendung weiter, man wird noch feststellen, dass der Browser zuerst nach jeder Anfrage ein annähernd ähnliches Dokument bekommt und anschließend das mittlerweile veraltet gewordene verwirft und schließlich die neue Version darstellt. Dieser Prozessfluss wird durch das zustandlose HTTP-Protokoll bestimmt, weil das damit unmittelbar verbundene Anfrage-Antwort-Paradigma dazu führt, dass bei jeder Benutzeraktion eine zugehörige Anfrage an den Server gerichtet wird, obwohl manche Benutzeraktionen davon keine große Änderung der betrachten HTML-Seite benötigen. Auf dem Server wird der gesamte Status während der Gespräche zwischen Benutzer und Server in einer Sitzung (Session) gehalten. Darin werden auch die Informationen gespeichert, die der Benutzer während der Interaktion mit der Applikation hinterlässt. Das heißt auch, dass umfassende Kommunikation ohne den Weg zurück auf den Server unmöglich ist.

Mit Ajax sind wir dem Ziel des reichhaltigen Clients näher gekommen. In traditionellen Web-Applikationen befindet sich ihre Applikationslogik ganz auf dem Webserver. In einer Ajax-Applikation wird sie hingegen teilweise in Richtung Client verlagert. Diese Verlagerung wird üblichweise beim ersten Login der Ajax-Applikation erfolgen. Deshalb ist das Datenaufkommen in einer Ajax-Applikation am Anfang viel höher als in den traditionellen Web-Applikationen. Die beim ersten Login geladene Seite ist ein komplexes Dokument mit einem hohen Anteil an JavaScript-Code, der genug Funktionalitäten enthält, um später von sich aus den Lebenszyklus zu verwalten. In weiterer Folge interagiert der Benutzer mit der lokalen Applikation, das wird den Status der Applikation verändern und neue Daten herstellen. Da sich die Applikationslogik jetzt teilweise auch auf der Seite des Clients befinden kann, wird es auch ermöglicht, dass der Status auch clientseitig in einer Session behalten wird.

Betrachten wir den ganzen Lebenszyklus einer Ajax-Applikation weiter, werden wir bemerken, dass sich das Datenaufkommen anders als eine traditionelle Web-Applikation verhält. Während sich bei einer traditionellen Web-Applikation die Menge übertragener Information dauernd annährend gleichmäßig verhält, weist eine Ajax-Applikation anfangs ein hohes Datenaufkommen auf und nimmt dieses aber in weiterer Folge ab. Deshalb nimmt damit auch der Anspruch an Bandbreite im Laufe der Zeit ab. Mit längerer Sitzungsdauer schafft eine

Ajax-Applikation einen klaren Vorteil gegenüber einem traditionellen Web-Applikation.

2.2.4. Vorteile und Nachteile von Ajax

In diesem Abschnitt werden wir auf die Vor- bzw. Nachteile bei der Verwendung von Ajax in Web-Applikationen eingehen.

Vorteile:

1. Plattformneutral: wird JavaScript aktiviert, dann können Ajax-Applikation unter allen aktuellen Webbrowser laufen. Dieser Vorteil macht Ajax-Applikationen auf allen Plattformen sehr leicht konfigurierbar.
2. Keine Plug-ins: Da Ajax plattformunabhängig entworfen worden ist, ist die separate Einbindung von Plug-ins nicht notwendig. Ajax schafft damit einen klaren Vorteil gegenüber anderen vergleichbaren Techniken, wie etwa Macromedias Shockwave oder Flash, wobei ein Plug-in benötigt wird, das nicht immer für alle Plattformen verfügbar ist.
3. Interaktivität und umfangreiche Nutzung: Im vergleich zu den traditionellen Web-Applikationen bietet eine Ajax-Applikation fast alle Interaktionsmöglichkeiten, die uns an herkömmliche Desktop-Applikationen erinnern.
4. Einsparung der Bandbreite: In einer Ajax-Applikation wird eine Webseite häufig nur teilweise aktualisiert, deshalb muss die Neuladung der ganzen Webseite nicht bei jeder Anfrage geschehen. Damit wird die Übertragung redundanter Information verringert. Die Bandbreite kann also auf diese Weise eingespart werden.

Nachteile:

1. Latenzzeit: In Ajax-Applikationen soll das Zeitintervall zwischen einer Anfrage und der zugehörigen Antwort berücksichtigt werden. Eine Ajax-Applikation verschickt häufig in kürzeren zeitlichen Intervallen mehrere Anfragen als eine herkömmliche Web-Applikation, das wird den Server mehr belasten.
2. Zurück-Knopf: Da Browsern für gewöhnlich nur statische HTML-Seiten in der Historie speichern können, ist es sehr schwer, den früheren Status durch den Zurück-Knopf wiederzuerstellen. Der Refresh-Knopf und die Aktivierung in den Favoriten haben das gleiche Problem.
3. Deaktivierung von JavaScript: Die JavaScript-Unterstützung kann aus Sicherheitsgründen in Browsern deaktiviert werden. Um Ajax-Applikationen zu benutzen, muss diese Unterstützung jedoch aktiviert werden.
4. Test Aufwand: Die gewissen Unterschiede zwischen einzelnen Browsern erhöhen den Testaufwand einer Ajax-Applikation. Um das Problem zu umgehen, können wir bei der Entwicklung einer Ajax-Applikation auf umfangreiche JavaScrip-Frameworks zurückgreifen, die die Unterschiede vor uns verbergen.

3. Basistechnologien in Ajax

In diesem Kaptitel werden wir die einzelnen in Ajax verwendeten Basistechnologien kurz erklären und durch ein durchgängiges Beispiel ihr Zusammenspiel erläutern.

3.1. HTML und XHTML

Eine Webseite besteht aus hierarchischen verschachtelten (X)HTML-Elementen. Über Ajax können alle Elemente und ihre Layoutinformationen verändert werden. Im Vergleich zu den herkömmlichen Web-Applikationen sollen die Webseiten einer Ajax-Applikation an sich möglichst wenigere Layoutinformationen enthalten. Im Folgenden ist die HTML-Seite des durchgängigen Beispiels abgebildet. Klicken wir den Button *anfordern* an, wird die Funktion *startRequest* aufgerufen. Darin werden in weiterer Folge die Daten vom Server abgerufen und in den -Tag integriert. Die Definition der *startRequest* wird später gegeben.

```
1.  <html>
2.  <head></head>
3.  <body>
4.      <form action="#">
5.          <input type="button" value="anfordern" onclick="startRequest();"/>
6.      </form>
7.      <div id="results"></div>
8.  </body>
9.  </html>
```

3.2. JavaScript

JavaScript ist eine objektbasierte Skriptsprache, die dynamisch und schwach typisiert ist und zur Laufzeit interpretiert, also nicht zuvor kompiliert wird. Jetzt ist JavaScript in fast allen Browsern verfügbar. Deshalb benutzt Ajax JavaScript als die zentrale Programmiersprache, die alle Bausteine von Ajax zusammenbindet. Es gibt drei Methoden, um JavaScript in einer HTML-Seite verwenden.

Methode	Beschreibung	Beispiel
Direkte Notation	verbindet einen JavaScript-Bereich mit einer Webseite.	<script type="text/javascript">. . .</script>
Inline-Referenz	ermöglicht das Einbetten einzelner Befehle in normale Tags	<input type="button" value="anfordern" onclick="startRequest();"/>

Externe JavaScript-Datei	Werden gleiche JavaScript-Funktionen in mehrere HTML-Seiten verwendet oder ist das Skript groß, sollen die Funktionen in separate Dateien ausgelagert werden, um sie besser warten zu können und um uns Übersichtlichkeit zu schaffen.	`<script type="text/javascript" src="[URL einer JavaScript-Datei]"></script>`

Tabelle 3-1: Einbetten von JavaScript

3.3. Das XMLHttpRequest-Objekt

Das XMLHttpRequest-Objekt ist das Herzstück aller Ajax-Applikationen und bietet die Möglichkeit, Anfragen in einer synchronen oder asynchronen Weise zu verschicken. Da das XMLHttpRequest-Objekt nicht standardisiert ist, kann sein Verhalten zwischen einzelnen Browsern variieren, aber trotzdem ist seine Funktionalität bei allen Browsern fast identisch. Ist die direkte Bedienung der Variierung nicht gewünscht, gibt es zahlreiche JavaScript-Bibliotheken, die eine Kapselung vornehmen, um den Aufwand zu verringern.

3.3.1. Erzeugen eines XMLHttpRequest-Objekts

Mit der folgenden Funktion *createXMLHttpRequest* können wir in allen gebräuchlichen Browsern XMLHttpRequest-Objekte erzeugen.

```
1.  function createXMLHttpRequest() {
2.          if (window.ActiveXObject) {
3.              xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
4.          }
5.          else if (window.XMLHttpRequest) {
6.              xmlHttp = new XMLHttpRequest();
7.          }
8.  }
```

Im Internet Explorer 5 und 6 wird das XMLHttpRequest-Objekt als ActiveX-Komponente angeboten. Das Objekt kann in diesen Browsern mit Hilfe des Codeabschnitts zwischen den Zeilen 2 und 4 erzeugt werden. Zeile 2 prüft, ob ActiveX verfügbar ist. Damit wird noch geprüft, ob das XMLHttpRequest-Objekt auf diese Weise unterstützt wird. Zeile 3 erzeugt ein neues ActivX-Objekt für das XMLHttpRequest-Objekt.

In Browsern wie dem Mozilla/Firefox ist XMLHttpRequest-Objekt standardmäßig vorhanden. In diesen Browsern kann das Objekt mit Hilfe des Codeabschnitts zwischen den Zeilen 5 und 7 erzeugt werden. Weil das XMLHttpRequest-Objekt in diesen Browsern dem Window-Objekt untergeordnet ist, können wir wie in Zeile 5 testen, ob das XMLHttpRequest-Objekt auf diese Weise unterstützt wird.

3.3.2. Schicken einer Anfrage an den Server

Um eine Anfrage an den Server zu schicken, muss man zuerst das XMLHttpRequest-Objekt vorbereiten. Wir tun das mit Hilfe des folgenden Methodenaufrufs:

```
xmlHttp.open("Http-Request-Methode",
             "URL", asynchrony?, "Benutzername", "Password");
```

Hier wird keine Verbindung zu einem Server geöffnet, sondern nur vorab konfiguriert.

1. Der erste Parameter gibt die Art der HTTP-Anfrage an. GET ist die gebräuchlichste Methode, um Inhalte vom Server anzufordern. Bei Übertragungen mit GET werden Daten als Bestandteile im Link übergeben. Mit POST werden Daten hingegen nicht im Link übergeben, sondern als Text im Nutzdatenblock mitgeschickt.
2. Das Ziel der Anfrage wird durch den zweiten Parameter angegeben. Dies kann beispielsweise ein JSP-Skript oder eine XML-Datei sein. Aus Sicherheitsgründen muss der URL relativ auf die gleiche Domain verweisen wie die HTML-Datei, die das XMLHttpRequest-Objekt erzeugte.
3. Der dritte Parameter legt fest, ob die Übertragung der Anfrage synchron oder asynchron erfolgen soll. Der boolesche Wert *true* steht für eine asynchrone Übertragung. Sollte an dieser Stelle ein *false* stehen, wird die Übertragung synchron ausgeführt. In der Zeit zwischen Anfrage und Antwort steht das System still. Der Standardwert dieses Parameters ist synchron (*false*).
4. Mit dem vierten und fünften Parameter werden der Benutzername und das Passwort festgelegt, wenn der Zugriff auf einen geschützten Bereich erfolgen soll.
5. Die Angaben *asynchron*, *Benutzername* und *Passwort* sind optional.

Der Sendevorgang erfolgt mit dem Methodenaufruf *send*(). Damit wird eine Anfrage mit den bei dem Methodenaufruf *open* angegebenen Einstellungen ausgelöst.

```
xmlHttp.send("Inhalt");
```

Bei der Übertragung per GET ist der Parameter null. Bei POST besteht der Parameter aus mit & verknüpften Wertepaaren an Daten, die an den Server übertragen werden sollen.

3.3.3. Reaktion auf die Antwort des Servers

Wir haben schon gesehen, wie wir ein XMLHttpRequest-Objekt erzeugen und auf die Übertragung einer Anfrage vorbereiten und diese Anfrage senden. Eine Ajax-Applikation schickt meistens asynchrone Anfragen an den Server und setzt anschließend seine Verarbeitung fort, ohne auf eine Antwort zu warten. Es wird uns interessieren, wie der Ajax-Applikation mitgeteilt wird, wenn die Antwort vom Server vorliegt?

Das XMLHttpRequest-Objekt definiert die Eigenschaft *readyState* und den Event-Handler *onreadstatechange*. Der Event-Handler wird vom Browser aufgerufen, wenn die Eigenschaft *readyState* geändert wird, die den Stand der Datenübertragung angibt. Die folgende Tabelle zeigt die möglichen Werte der Eigenschaft *readState*. Der gebräuchlichste Wert ist 4.

Wert	Bezeichnung	Bedeutung
0	UNINITIALIZED	Es erfolgte noch kein Aufruf der Methode *open()*.
1	LOADING	Die *open()* wurde aufgerufen, aber keine Anfrage abgesetzt.
2	LOADED	Die Anfrage wurde mittels der Methode send() abgesetzt.
3	INTERACTIVE	Teile der Antwort sind bereits verfügbar.
4	COMPLETED	Die Antwort vom Server liegt vollständig vor.

Tabelle 3-2: mögliche Werte der Eigenschaft readState

Die folgende Funktion *startRequest* zeigt, wie wir festlegen, was nach der Antwort des Servers weiter geschehen soll, indem wir in Zeile 3 unsere Funktion *handleStateChange* zu der Eigenschaft *onreadystatechange* zuweisen. Die Funktion wird vom Browser aufgerufen, sobald sich die Eigenschaft *readState* ändert. Mit Zeilen 4-5 wird die Anfrage an den Server vorbereitet und verschickt, um eine Datei *innerHTML.txt* vom Server anzufordern.

```
1.  function startRequest() {
2.      createXMLHttpRequest();
3.      xmlHttp.onreadystatechange = handleStateChange;
4.      xmlHttp.open("GET", "innerHTML.txt", true);
5.      xmlHttp.send(null);
6.  }
7.  function handleStateChange() {
8.      if (xmlHttp.readyState == 4) {
9.          //. . .
10.      }
11. }
```

In Zeile 8 wird mit der Eigenschaft *readyState* ein bestimmter Status der Datenübertragung überprüft. Der Wert 4 steht für die Vollständigkeit der Datenübertragung. Wenn also die Daten vollständig übertragen wurden, setzen wir ab Zeile 9 andere Verarbeitung fort.

3.3.4. Statuscodes in HTTP

Wird die Vollständigkeit der Übertragung bestätigt, erfolgt in der oben deklarierten Funktion die weitere Behandlung. Aus vielerlei Gründen sollte darin zuerst eine Status Abfrage durchgeführt werden. Im HTTP-Protokoll werden verschiedene Statuscodes bzw. Fehlercodes definiert. Einige davon sind in der Folgenden Tabelle dargestellt.

Wert	Bezeichnung	Bedeutung
200	OK	Die Anfrage wurde erfolgreich bearbeitet.
401	Unauthorized	Ohne gültige Authentifizierung geht die Anfrage nicht.
403	Forbidden	Mangels Berechtigung wurde die Anfrage verweigert.
404	Not Found	Die angeforderte Datei wurde nicht gefunden
500	Internal Server Error	unerwartete Serverfehler

Tabelle 3-3: Statuscodes in http

Die Eigenschaft *status* des XMLHttpRequest-Objekts gibt den Statuscode wieder. Damit kann eine Applikation auf die unterschiedlichen Möglichkeiten reagieren. Wir ergänzen das letzte if-Statement in der *handleStateChange*. Das fragt ab, ob die Übertragung erfolgreich war:

```
1.    if (xmlHttp.readyState == 4) {
2.      if (xmlHttp.status == 200) {
3.        //var antwortXMLDoc = xmlHttp.responseXML;
4.        docment.getElementById("results").innerHTML = xmlHttp.responseText;
5.      }
6.    }
```

Lag die Antwort vom Server vollständig vor und wurde die Anfrage auch erfolgreich bearbeitet, kann der Inhalt dieser Antwort mit der Eigenschaft *reponseText* oder *reponseXML* des XMLHttpRequest-Objekts ausgelesen werden. Die Eigenschaft *reponseText* gibt wie in Zeile 4 das Ergebnis als String zurück, während die Eigenschaft *reponseXML* wie in Zeile 3 das Ergebnis als DOM-Baum in Form eines Objekts der Klasse *XMLDokument* repräsentiert. Mit Zeile 4 wird also der Inhalt der Datei *innerHTML.txt* ausgelesen.

3.4. DOM – Document Object Model

Das Document Object Model ist ein Standard des W3C-Konsortiums. Er ist plattform- und programmiersprachenunabhängig definiert und stellt eine einheitliche Schnittstelle für den dynamischen Zugriff auf die strukturierten Daten in (X)HTML- und XML-Dokumenten dar. Sein Konzept beschreibt ein Verfahren, bei dem ein Dokument nicht als statisch sondern als dynamisch betrachtet und aufgebaut wird. Mit Hilfe von DOM lässt sich ein Dokument in einer Baumstruktur mit hierarchischer Verschachtelung der Elemente auffassen. Dieser Ansatz ermöglicht die individuelle Behandlung – auslesen, modifizieren, erstellen und löschen -- der einzelnen Bestandteile – Inhalte, Strukturen und Layoutinformationen – von Webseiten.

Im DOM wird ein Objekt *document* definiert. Dies Objekt ist der Einstiegspunkt, von dem heraus wir auf alle anderen Elemente zugreifen können. In unserem Beispiel bekommen wir die Daten, nachdem die Antwort vom Server empfangen wurde, über das Attribut *responseText* und integrieren sie in den -Tag, deren ID-Wert *result* ist. Mit dem Methodenaufruf

getElementById in Zeile 4 im letzten Codenabschnitt bekommen wir das Objekt, das den -Tag repräsentiert, damit alle Attribute vom -Tag auch zugänglich sind.

3.5. Cascading Style Sheets (CSS)

Ein CSS besteht aus reinem Klartext und beschreibt den Regeln zur Formatierung von Bestandteilen eines (X)HTML-Dokuments. CSS trennt Struktur von Gestaltung, schafft dem HTML-Code bessere Übersichtlichkeit, bietet mehr gestalterische Möglichkeiten und verringert die Wartenkosten eines Webprojekts. Im Folgenden werden Einbindung, Box-Modell, Selektoren, und Positionierung in CSS vorgestellt.

3.5.1. Einbindung von CSS

Es gibt drei folgende Methoden, um das CSS in einer Webseite zu verwenden.

Methode	**Beschreibung**	**Beispiel**
Interne Style Sheets	kann mit Hilfe vom <style>-Tag in eine Webseite eingebettet werden. Das type-Attribut zeigt die verwendete Stylesheet-Sprache.	<style type="text/css"> <! -- irgendwelche Anweisungen --> </style>
Externe Style Sheets	kann mit Hilfe des <link>-Tags hergestellt werden. Das ref-Attribut teilt dem Browser mit, dass eine CSS-Datei eingebunden wird. Das href-Attribut verweist auf den Spreicherort.	<link type="text/css" ref="stylesheet" href="[URL einer CSS-Datei]">
Inline-Definition	Will man nur einem Element individuelle Layoutinformationen geben. Das können wir tun, indem wir ein zusätzliches style-Attribut innerhalb eines HTML-Tags verwenden. Diese Layoutinformation wirkt ausschließlich auf Elmente innerhalb des Tags.	<h1 style="font-family:arial; font-size:30pt;color:blue"> PG:EMA</h1> Hier werden die Schriftfamilie, -größe und -farbe individuell gesetzt.

Tabelle 3-4: Einbindung von CSS

Grundsätzlich ist eine externe Lösung bevorzugt und hat viele offenbare Vorteile – kleinere Dateien, saubere Trennung von Layout und Inhalt, Mehrfachverwendung und bessere Wartbarkeit. Solche Vorteile reduzieren den Aufwand bei der Erstellung und Wartung eines großen Webprojekts erheblich. Eine interne Lösung wird nur dann empfohlen, wenn die Layoutinformationen über das CSS sehr wenig sind.

3.5.2. Selektoren

Mit Selektoren legen wir die Regeln für den Mustervergleich fest, auf welche Bestandteile eines (X)HTML-Dokuments ein Style angewendet werden soll.

Muster	Bezeichnung		Beschreibung	Beispiel
#meineID	ID-Selektor		enthält ein *„#“* gefolgt von einem ID-Wert und stimmt mit jedem Element überein, dessen ID gleich *meineID* ist.	#results { color : red; } zeigt das Element mit dem ID-Wert *results* in rot an.
*	Universeller Selektor		wird als *„*“* angegeben und gilt für alle Elemente eines Dokuments.	* { font-family : arial; } setzt die Schriftart für alle Elemente auf *arial*.
E	Typselektoren	gilt für alle Elemente von einem Typ E.	p { font-family : arial; } setzt die Schriftart für alle *p-Elemente* auf *arial*	.rot { color : red; } rot färbt alle Elemente, die das Attribut *class= „rot“* aufweisen.
.Klassenname		allgemein	gilt für alle Elemente, die das Attribut *class= „Klassename“* aufweisen	h2.rot { color : red;} <h2 class="rot"> rot färbt alle * -Tags*, die das Attribut *class= „rot“* aufweisen
E.Klassenname			elementspezifisch	gilt für bestimmte Elemente, die das Attribut *class= „Klassename“* aufweisen
a:Element	Pseudoklassen-Selektor		Da die a*:Elemente* in verschiedenen Browsern unterschiedlich unterstützt werden, sollten zum Einsatz unfangreiche Tests durchgeführt werden.	*a:link* für den nomalen Hyperlink. *a:visited* für bereits angeklickte Hyperlinks. *a:hover* für Hyperlinks, worüber sich gerade der Mauszeiger befindet.
E N	Selektor für Nachfahren		gilt für alle Elemente vom Typ N, die ein Nachfahre eines E-Elements ist.	div p { color : red; } färbt alle Absätze innerhalb von -Tags in rot.
E>K	Kindselektoren		gilt für alle K-Elemente, die Kind eines E-Elements sind.	div > table { background : red; } färbt table-Elemente in rot, die sich direkt innerhalb eines div-Elements befinden.

E + N	Nachbar-Selektor	gilt für jedes N-Element, dem unmittelbar ein Element E vorausgeht.	h1 + p { color : red; } färbt einen Absatz in rot, der direkt auf eine Überschrift h1 folgt.

Tabelle 3-5: Selektoren in CSS

3.5.3. CSS-Boxen

Jedes Element eines Dokuments besitzt nach der Definition von CSS ein Box-Modell. Das Modell besteht aus folgenden vier Bereichen – dem Inhalt (content), dem Innenabstand (padding), dem Rahmen (border) sowie dem Rand (margin). Das Modell beschreibt, wie der Element-Auftritt in einer HTML-Seite durch das Zusammenwirken der vier Bereiche festgelegt wird. Im Folgenden wird das div-Element in unserem durchgängigen Beispiel auf eine neue Größe gesetzt, deren Bereite 500px (40px + 0px + 10px + 400px + 10px + 0px + 40px) ist.

```
#results { width : 400px; padding : 10px; border : 0; margin : 40px; }
```

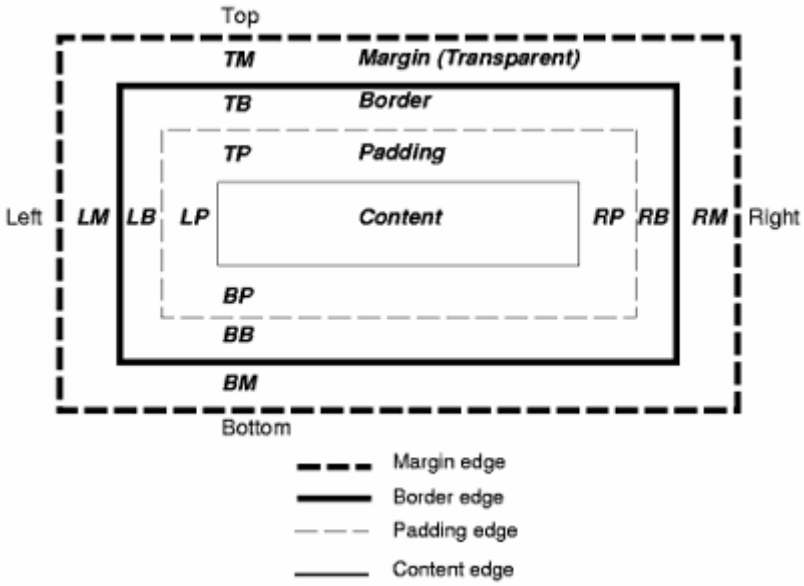

Abbildung 3-1: CSS-Box (Quelle[2])

3.5.4. Positionierung in CSS

In CSS lässt sich Positionen durch die Eigenschaft „position“ definieren, Abstände durch die Eigenschaften „margin“ und „padding“.

[2] Quelle: http://www.w3.org/TR/REC-CSS2/box.html zuletzt besucht am 26.10.2007

Positionierung	Art		Beschreibung	Beispiel
Abstand	gemeinsam	margin	Der Abstand kann mit *margin* gemeinsam für alle Richtungen definiert werden.	p { margin: 1cm 1cm 2cm 2cm; }
	einzeln	margin-top margin-left margin-bottom margin-right	Man kann den Abstand auch für einzelne definieren.	p { margin-top: 1cm; margin-left: 1cm; margin-bottom: 2cm; margin-right: 2cm; }
Position	absolut		gemessen an der linken oberen Ecke des umgebenden Blocks	h1 { position: absolute; left: 25px; top: 25px } lässt alle <h1> 25 Pixel nach unten und rechts von der linken oberen Ecke platzieren.
	fixed		absolut gemessen am "Viewport"	#f1 { position: fixed; left: 25px; top: 25px }
	relativ		gemessen an der Stelle des Elements	h1 { position: relative; left: 25px; top: 25px } verschiebt alle <h1> um 50 Pixel nach unten und rechts.

Tabelle 3-6: Positionierung in CSS

6. Fazit

Ajax ist an sich keine neue Technologie, sondern vielmehr eine neue Architektur. In dieser Architektur spielt jede Basistechnologien eine Rolle – CSS ist für die saubere Trennung zwischen Inhalt, Struktur und Darstellung zuständig, DOM für die dynamische Ausgabe und Interaktion, XMLHttpRequest-Objekt für asynchrone Datenanfragen, XML für den Datenaustausch und JavaScript bindet letztendlich alles zusammen. Diese klare Arbeitsteilung hat traditionelle Web-Applikationen aus dem früheren Chaos gerettet und bringt dem Web mehr Ordnung.

Tabellenverzechnis

Abbildungsverzechnis

Literatur

[JF06] JACOBI, J.; FALLOWS, J. R.: *JSF und Ajax, Rich-Internet-Komponenten entwickeln mit JavaServer Faces und Ajax.* REDLINE GMBH, Heidelberg, 2006

[GGA06] GEHTLAND, J.; GALBRAITH, B.; ALMAER, D.: *Pragmatisch Programmieren Ajax, Eine Pragmatische Einführung in Web 2.0.* Carl Hanser Verlag, München Wien, 2006

[Ste06] STEYER, R.: *Ajax mit Javer-Servlets und JSP.* Addison-Wesley Verlag, 2006

[AS05] ASLESON, R.; SHUTTA, N. T.: *Foudations of Ajax.* Apress Verlag, 2005

[Koc07] KOCH, S.: *JavaScript, Einführung, Programmierung und Referenz.* dpunkt.verlag GmbH, Heidelberg, 2007

[MSM07] MARINSCHEK, M.; SCHNABL, A.; MÜLLAN, G.: *JSF@Work JavaServer Faces und Apache MyFaces erfolgreich einsetzen.* dpunkt.verlag GmbH, Heidelberg, 2007

[Car06] CARL, D.: *Praxiswissen Ajax.* O'Reilly Verlag, 2006

[BMC06] BUDD, A.; MOLL, C.; COLLISON, S.: *CSS Mastery, Webdesign für Fortschrittene.* Addison-Wesley, München, Dez. 2006

[BB06] BERGMANN, O.; BORMANN, C.: *AJAX – Frische Ansätze für das Web – Design.* TEIA Verlag, 2006

[Sel07] SELFHTML: http://de.selfhtml.org/index.htm. zuletzt besucht am 25.10.2007

[Sty07] THE STYLEWORKS: http://www.thestyleworks.de/index.shtml zuletzt besucht am 25.10.2007