HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG

Fakultät Informatik

Xajax – Ein Ajax-Framework für PHP

Ausarbeitung im Fach Verteilte Systeme 2
Prof. Dr. Jürgen Wäsch
Wintersemester 2007/2008

von

Max Nagl
nagl@fh-konstanz.de

Andreas Hofmann
ahofmann@htwg-konstanz.de

Inhaltsverzeichnis

1. Einleitung

Diese Ausarbeitung entstand im Rahmen der von Professor Dr. Jürgen Wäsch gehaltenen Vorlesung „Verteilte Systeme 2“ im Wintersemester 2007/2008 an der HTWG Konstanz. Das Themengebiet dieses Semesters war Ajax, insbesondere die Untersuchung von Ajax-Frameworks.

In diesem Dokument wird das Ajax-Framework xajax vorgestellt, wobei in den Kapiteln 2, 3 und 5 drei zentrale Fragen beantwortet werden:

- Was ist Xajax?
- Wie verwendet man Xajax?
- Warum sollte man Xajax verwenden?

Zusätzlich wird in Kaptitel 4 der Funktionsumfang von Xajax noch einmal detaillierter beschrieben und ein Blick in die Zukunft von Xajax geworfen.

Im Rahmen dieser Arbeit ist auch eine Beispielanwendung in Form eines Dateibrowsers entstanden. Der Quellcode für diesen befindet sich im Anhang.

2. Was ist Xajax?

In diesem Abschnitt wird das Ajax-Framework Xajax vorgestellt. Dabei wird erklärt, worum es sich bei Xajax handelt und ein Blick auf die zugrunde liegenden Konzepte geworfen. Xajax wird kurz mit anderen Ajax-Frameworks verglichen und eine Einordnung wird vorgenommen.

2.1. Überblick

Xajax ist eine quelloffene PHP-Klassenbibliothek zur Erstellung von AJAX-Applikationen. Der Schwerpunkt von Xajax liegt darin, serverseitig in PHP implementierte Funktionen so verfügbar zu machen, dass sie clientseitig durch Javascript aufgerufen werden können. Die komplette - durch Ajax asynchrone - Serverkommunikation wird dabei maskiert, der Programmierer muss sich nicht darum kümmern.

2.2. Ablauf der Entwicklung

Der allgemeine Ablauf der Entwicklung mit Xajax ist folgender:

- Der Programmierer implementiert reine PHP-Funktionen und meldet diese bei Xajax an.
- Der Rückgabewert wir durch ein xajaxResponse-Objekt[1] erzeugt.
- Xajax erstellt dynamisch die entsprechenden Javascript-Funktionen.
- Diese Funktionen können jetzt clientseitig aufgerufen werden, z.B. in Event-Handlern von HTML-Tags.

Der Programmierer kommt also nicht nur ohne Kenntnisse über asynchrone Kommunikation mit Ajax aus, sondern fast gänzlich ohne Javascript-Kenntnisse.

2.3. Datenfluss

Der Datenfluss zwischen Client und Server verläuft je nach Richtung unterschiedlich.

[1] Für mehr Informationen zu xajaxResponse siehe Abschnitt 4.2.

Anfragen an den Server werden als klassischer CGI-Request verschickt. Standardmäßig nutzt Xajax hier POST, es ist jedoch auch GET möglich. Es werden dabei jeweils Schlüssel/Wert-Paare mit dem Namen der aufzurufenden Funktion und eventuell den übergebenen Parametern gesendet.

Die Antwort des Servers kommt in Form von XML, wobei Xajax hier ein eigenes XML-Format mit einem etwas eigentümlichem Benennungsschema definiert hat. Elementnamen enthalten keine Vokale, so heißt zum Beispiel das Rootelement „xjx“, Attribute wiederum bestehen nur aus einem Buchstaben, zum Beispiel „n“ für „Name“. Xajax unterstützt kein JSON, dies ist jedoch für zukünftige Versionen geplant (siehe Abschnitt 4.4).

Abbildung 2.1 zeigt den Datenfluss beim Aufruf einer hypothetischen PHP-Funktion foo(bar).

1. Als erstes wird durch die Funktion Xajax_foo(bar) der JavaScript-Teil von Xajax aufgerufen. Die Namen der generierten Javascript-Funktionen sind immer identisch mit dem Namen der PHP-Funktion zuzüglich des Präfixes „Xajax_“.
2. Der JavaScript-Teil von Xajax startet eine POST-Request an den Webserver und ruft somit den PHP-Teil von Xajax auf.
3. Der PHP-Teil von Xajax ruft die selbst geschriebene PHP-Funktion foo(bar) auf.
4. In der PHP-Funktion foo(bar) wurde eine Instanz der Klasse XajaxResponse erstellt und an den PHP-Teil von Xajax zurückgegeben.
5. Der PHP-Teil von Xajax formt das XajaxResponse-Objekt in XML um und übergibt dieses an des JavaScript-Teil, welcher das Objekt auswertet und die enthaltenen Anweisungen an dem lokalen HTML-Dokument durchführt.

2.4. Lizenz

Im Moment wird Xajax unter der LGPL-Lizenz vertrieben, welche die Nutzung für kommerzielle und proprietäre Projekte erlaubt. Änderungen an Xajax selbst müssen jedoch im Quellcode veröffentlicht werden. Zukünftige Versionen von Xajax werden unter der BSD-Lizenz, welche diese Einschränkung nicht mehr hat, veröffentlicht werden. Somit kann man Xajax für seine Zwecke anpassen, ohne diese Änderungen öffentlich zu machen.

Obwohl Xajax quelloffen ist, unterliegt die offizielle Entwicklung der vollständigen Kontrolle eines kleinen Kernteams, bestehend aus Jared White und J. Max Wilson, den Erfindern von Xajax, sowie Eion Robb. Mitglieder der Xajax-Community können sich zwar an der Entwicklung beteiligen, alle Fremdbeiträge müssen jedoch erst durch das Kernteam begutachtet und abgesegnet werden, bevor sie in die offiziellen Builds einfließen.

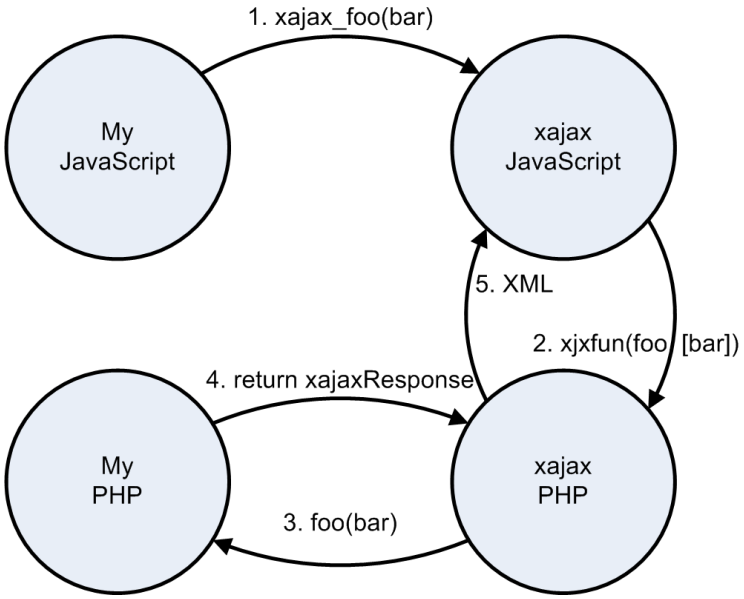

Abbildung 2.1.: Datenfluss beim Aufruf einer von Xajax bereitgestellten Funktion

2.5. Einordnung des Frameworks

Um Xajax gegenüber anderen Ajax-Frameworks abzugrenzen, werden wir nun eine Einordnung nach bestimmten Kategorien (Mintert u. Leisegang, 2006, 113) vornehmen.

2.5.1. Client/Server

Bei Xajax handelt es sich um ein serverseitiges Framework, nicht um ein clientseitiges. Somit wird primär nicht mit Javascript innerhalb der vom Server an den Client gelieferten Webseite programmiert, sondern vor Auslieferung der Webseite auf dem Server deren Erzeugung beeinflusst.

2.5.2. Abstraktion

Xajax bietet einen hohen Level an Abstraktion, der Programmierer kommt nicht mit den Abläufen der Ajax-Kommunikation in Berührung und muss auch keine tiefergehenden Kenntnisse von Javascript besitzen.

2.5.3. Funktionalität

Die Kernfunktionalität von Xajax ist das Verfügbarmachen von serverseitig implementierten Funktionen im Browser via Ajax. Es bietet keine darüber hinausgehenden Funk-

tionen wie etwa zur Erstellung vollständiger komplexer Anwendungen inklusive Benutzeroberfläche (z.B. GWT, DWR ...), zur Darstellung visueller Effekte (z.B. Scriptaculous, MooFx ...) oder zur Erweiterung des Sprachumfangs von JavaScript (z.B. Prototype, JQuery ...)

2.5.4. Laufzeitbibliothek/Generator

Das Xajax-Framework läuft zur Laufzeit der Webapplikation, d.h. es wird bei jedem Seitenaufruf ausgeführt und alle dynamischen Funktionen werden zu diesem Zeitpunkt generiert. Die Alternative wäre ein Generator, welcher den nötigen JavaScript-Code einmalig erzeugt und nur bei Implementierungsänderungen neu erstellt.

2.5.5. Unterstützte Sprachen

Xajax unterstützt ausschließlich PHP als Serversprache. Es gibt verschiedene Ajax-Frameworks für verschiedene Sprachen, etwa GWT und DWR für Java oder Atlas für .NET. Außerdem gibt es etwa auch Frameworks, welche versuchen, serverseitig mehrere Sprachen zu unterstützen, zum Beispiel Sajax.

2.5.6. Laufzeitumgebung

Xajax benötigt lediglich PHP, egal ob als Modul oder CGI, also läuft es auf jedem Webserver, der PHP unterstützt.

3. Wie verwendet man xajax?

In diesem Abschnitt wird Schritt für Schritt erklärt, wie man eine Xajax-Anwendung erstellt, welche auf Knopfdruck das Produkt aus zwei Eingabefeldern berechnet und das Ergebnis in ein weiteres Feld schreibt.

3.1. Installation

Im Folgenden wird die Installation von Xajax beschrieben (White u. Wilson, 2007d). Um Xajax betreiben zu können, müssen folgende Voraussetzungen erfüllt sein:

- Web Server: Apache oder IIS
- PHP 4.3.x oder PHP 5.x
- Unterstützte Browser: Internet Explorer 5.5, Firefox 1.0, Safari 1.3, Opera 8.5.

Als erstes sollte man die neueste Version von Xajax herunterladen. Anschließend müssen folgende Ordner in ein Verzeichnis auf dem Webserver kopiert werden:

- xajax_controls
- xajax_core
- xajax_js
- xajax_plugins

Damit ist die Installation bereits abgeschlossen.

3.2. Initialisierung

Zu Beginn muss man Xajax in PHP einbinden. Die geschieht durch den Befehl require oder require_once. Um ein mehrmaliges Einbinden zu vermeiden, ist die Funktion require_once vorzuziehen. Sie verhindert das mehrmalige Einlesen der Xajax-Quelldateien.

Deshalb sieht die erste Code-Zeile einer Xajax-Anwendung meist so aus:

```
require_once("xajax_core/xajax.inc.php");
```

Um Xajax-Funktionen verwenden zu können, muss nun eine Instanz der Klasse Xajax angelegt werden:

```
$xajax = new xajax();
```

3.3. Implementierung der serverseitigen Funktionen

Um PHP-Funktionen aus JavaScript aufrufen zu können, müssen diese in der Instanz der Xajax-Klasse registriert werden. Die geschieht mittels der Funktion registerFunction. Um beispielsweise die Funktion „berechnen“ zu registrieren kann man folgende Zeile verwenden:

```
$xajax->registerFunction("berechnen");
```

Implementiert man in PHP mehrere Funktionen namens „berechnen“ mit unterschiedlich vielen Parametern, so werden mit dem obigen Aufruf alle diese Funktionen registriert. Welche aufgerufen wird, wird durch die Anzahl der im JavaScript übergebenen Parameter bestimmt.

Die registrierte Funktion muss natürlich auch implementiert werden. Als Beispiel wird hier die Funktion „berechnen“ mit zwei Parametern implementiert:

```
function berechnen($a, $b)
{
    $objResponse = new xajaxResponse();
    $objResponse->assign("Ergebnis",
                "value", $a * $b);
    return $objResponse;
}
```

Diese Funktion berechnet das Produkt aus den zwei Parametern und weist das Ergebnis dem HTML-Element mit der ID „Ergebnis“ zu. Dazu wird als erstes eine Instanz der Klasse xajaxResponse angelegt. Diese Klasse wird benötigt um Änderungen an der Webseite durchzuführen. Sie wird in Kapitel 4 noch genauer erklärt.

Um die Funktionen abzuarbeiten, die durch JavaScript aufgerufen wurden, verwendet man die folgende Funktion:

```
$xajax->processRequest();
```

Hierbei ruft Xajax gegebenenfalls die aufgerufene Funktion in PHP auf und beendet anschließend PHP. Wurde keine Funktion mittels Xajax aufgerufen, wird PHP einfach fortgesetzt. Es ist wichtig, diese Funktion aufzurufen, bevor in PHP irgendetwas ausgegeben wurde.

Um das benötigte JavaScript zu generieren wird die Funktion printJavaScript aufgerufen. Dies sollte am Besten im Head der HTML-Ausgabe geschehen:

```
<html>
  <head>
    ...
    <?php $xajax->printJavaScript(); ?>
  </head>
  ...
</html>
```

3.4. Aufruf der Serverfunktionen durch den Client

Um nun die PHP-Funktionen aus JavaScript aufzurufen wird vor den Funktionsnamen der Präfix „xajax_“ gehängt. Da der Aufruf von PHP-Funktionen in Xajax meist direkt in ein HTML-Element geschrieben wird, werden kaum JavaScript-Kenntnisse benötigt. Das folgende Code-Segment erstellt einen Button, der auf Knopfdruck die PHP-Funktion „berechnen“ aufruft:

```
<button onclick="xajax_berechnen(2,3);">berechne</button>
```

Hier werden als Parameter die Konstanten 2 und 3 übergeben. Um die Werte aus einem Inputfeld zu erhalten muss ein wenig mehr JavaScript verwendet werden:

```
<button onclick="xajax_berechnen(document.getElementById('a').value,
   document.getElementById('b').value);">berechne</button>
```

Dies ruft die Funktion mit den Werten aus den Eingabefeldern „a“ und „b“ auf.

3.5. Quellcode

Der gesamte Quellcode des Beispiels ist hier zu sehen:

```
<?
require_once("xajax_core/xajax.inc.php");

$xajax = new xajax();
$xajax->registerFunction("berechnen");

function berechnen($a, $b) {
  $objResponse = new xajaxResponse();
  $objResponse->assign("ergebnis", "value", $a * $b);
  return $objResponse;
```

```
}

$xajax->processRequest();

?>
<html>
  <head>
    <?php $xajax->printJavaScript(); ?>
  </head>
  <input type="text" id="a" value="2" size="2">*
  <input type="text" id="b" value="3" size="2">=
  <input type="text" id="ergebnis" size="2" value="">
<button onclick="xajax_berechnen(document.getElementById('a').value,
   document.getElementById('b').value);">berechne</button>
</html>
```

3.6. Datenfluss

Nun wird betrachtet was passiert wenn der Button gedrückt wird. Der Datenfluss wird in Abbildung 3.1 dargestellt.

1. Als erstes wird durch die Funktion xajax_berechnen(a,b) der JavaScript-Teil von Xajax aufgerufen.
2. Der JavaScript-Teil von Xajax startet eine Post-Anfrage an den Webserver und ruft somit den PHP-Teil auf.
3. Der PHP-Teil von Xajax ruft die selbst geschriebene PHP-Funktion berechnen(a,b) auf.
4. In der PHP-Funktion berechnen(a,b) wurde eine Instanz der Klasse xajaxResponse erstellt und an den PHP-Teil von Xajax zurückgegeben.
5. Der PHP-Teil von Xajax formt das xajaxResponse Objekt in XML um und übergibt dieses an den JavaScript-Teil, welcher das Objekt auswertet.

Die Kommunikation zwischen JavaScript und PHP läuft in Xajax auf zwei verschiedene Arten:

- Von JavaScript nach PHP:
  Hier wird ein POST-Request verwendet. Was in dem Beispiel genau übertragen wird, ist hier zu sehen:

  ```
  POST /multi.php HTTP/1.1
  content-type: application/x-www-form-urlencoded
  User-Agent: Opera/9.23 (Windows NT 5.1; U; en)
  Host: siebn.de
  ```

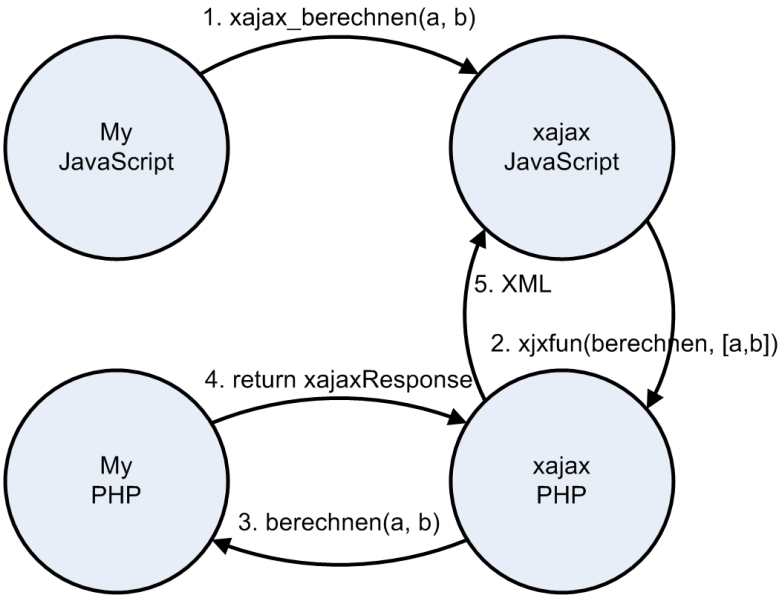

Abbildung 3.1.: xajData Flow

```
Referer: http://siebn.de/multi.php
style_cookie=null
Connection: Keep-Alive, TE
Content-Length: 59
Content-Transfer-Encoding: binary

....e.xjxfun=berechnen&xjxr=1195069649031&xjxargs[]=2&xjxargs[]=3
....e.HTTP/1.1 200 OK
```

- Von PHP nach JavaScript:
  In dieser Richtung wird XML verschickt. Der gesamte Verkehr ist hier abgebildet:

```
Date: Wed, 14 Nov 2007 19:47:27 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) PHP/4.3.10-22
X-Powered-By: PHP/4.3.10-22
Content-Length: 92
Keep-Alive: timeout=15, max=1000
Connection: Keep-Alive
Content-Type: text/xml ; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<xjx><cmd n="as" t="ergebnis" p="value">6</cmd></xjx>
```

4. Funktionsumfang

Dieses Kapitel betrachtet den Funktionsumfang von Xajax etwas detaillierter. In Xajax gibt es drei zentrale Konstrukte: Auf der Serverseite die beiden PHP-Klassen „xajax“ und „xajaxResponse“, auf der Clientseite das Javascript-Objekt „xajax“. Im folgenden werden die wichtigsten Funktionen dieser Konstrukte vorgestellt. Anschließend werfen wir noch einen Blick in die Zukunft von Xajax.

4.1. Die PHP-Klasse „xajax“

Die PHP-Klasse „xajax“ bündelt die Kernfunktionalität von Xajax. Wie in Kapitel 3 gezeigt, bietet Sie Methoden zur Registrierung von PHP-Funktionen, zur Erstellung des entsprechenden Javascript-Codes und zur Verarbeitung von CGI-Requests. Darüber hinaus bietet sie noch einige Methoden zur Vornahme diverser Einstellungen. (White u. Wilson, 2007a)

setRequestURI (string $sRequestURI) Legt die URI fest, an die Requests gesendet werden.

setWrapperPrefix ($sPrefix) Hiermit lässt sich ein anderer Präfix als ”xajax_”für die generierten Javascript-Funktionen einstellen.

setLogFile (string $sFilename) Legt fest, in welche Datei Xajax Logs schreiben soll. Standardmäßig wird kein Log angelegt.

setCharEncoding (string $sEncoding) Legt die Zeichencodierung der von Xajax erzeugten HTML-Ausgabe fest (zum Beispiel (ÏSO-8859-1öder ütf-8”).

registerFunction (mixed $mFunction, string $sRequestType) Registriert eine PHP-Funktion, die durch Javascript aufrufbar sein soll. Übergeben wird entweder ein String mit dem Funktionsnamen oder ein String-Array mit folgendem Aufbau: `array("myFunctionName", "myClass", "myMethod")`

Damit lassen sich statische Methoden von PHP-Klassen registrieren. Ersetzt man den zweiten String durch eine Objektvariable, kann man auch Methoden von instanzierten Objekten registrieren.

registerExternalFunction (mixed $mFunction, string $sFile, string $sRequestType) Von der Funktion her identisch mit `registerFunction`, nur mit zusätzlichen Parameter für den Dateinamen der PHP-Datei, in der die Funktion zu finden ist.

registerCatchAllFunction (mixed $mFunction) Legt eine PHP-Funktion fest, welche

aufgerufen werden soll, wenn die gewünschte PHP-Funktion nicht gefunden wird. Dies kann eigentlich nur auftreten, wenn man in Javascript direkt `xajax.call()` aufruft.

registerPreFunction (mixed $mFunction) Hier legt man eine PHP-Funktion fest, die vor jedem Aufruf einer registrierten PHP-Funktion ausgeführt werden soll. Die Rückgabewerte von beiden Funktionen werden kombiniert.

processRequests () Diese Methode muss aufgerufen werden, um die Ajax-Requests verarbeiten zu können. Sie sollte nur ein einziges Mal und erst nach allen Funktionsregistrierungen aufgerufen werden.

printJavascript (string $sJsURI=, string $sJsFile=NULL) Gibt den von Xajax generierten Javascript-Code aus. Sollte innerhalb des HTML-Tags head aufgerufen werden. Die Parameter dienen dazu, Pfad und Namen der Javascript-Datei von Xajax anzugeben, falls diese nicht im selben Verzeichnis wie die PHP-Datei liegen.

getJavascript (string $sJsURI=, string $sJsFile=NULL) Von der Funktion her identisch mit `printJavascript`, gibt den Javascript-Code jedoch als String zurück.

4.2. Die PHP-Klasse „xajaxResponse“

Aus Sicht des Programmierers bietet die PHP-Klasse „xajaxResponse“ den eigentlichen Nutzen von Xajax. In jeder serverseitig implementierten PHP-Funktion, die über Xajax clientseitig aufrufbar sein soll, wird als Rückgabewert ein Objekt vom Typ „xajaxResponse“ erwartet. Mit den Methoden von „xajaxResponse“ kann man Befehle definieren, welche clientseitig ausgeführt werden sollen, hauptsächlich Änderungen am DOM. (White u. Wilson, 2007b)

addConfirmCommands (integer $iCmdNumber, string $sMessage) Fügt einen Bestätigungsdialog hinzu, mit dem der Benutzer in die Ausführung der DOM-Modifikationen eingreifen kann. `$iCmdNumber` ist die Anzahl der Befehle, die Übersprungen werden sollen, wenn der Benutzer auf „Abbrechen“ klickt, `$sMessage` ist der Text, der im Bestätigungsdialog angezeigt werden soll.

addAssign (string $sTarget, string $sAttribute, string $sData) Diese Methode setzt das Attribut `$sAttribute` des Elements mit der ID `$sTarget` auf den Wert `$sData`. Um den Inhalt des Elements selbst zu ersetzen, kann als Attribut „innerHTML“ angegeben werden.

addAppend (string $sTarget, string $sAttribute, string $sData) Diese Methode erweitert das Attribut `$sAttribute` des Elements mit der ID `$sTarget` auf den Wert `$sData`. Der neue Inhalt wird hinter den bisherigen eingefügt.

addPrepend (string $sTarget, string $sAttribute, string $sData) Erweitert das Attribut `$sAttribute` des Elements mit der ID `$sTarget` auf den Wert `$sData`. Der neue Inhalt wird vor dem bisherigen eingefügt.

addReplace (str $sTarget, str $sAttribute, str $sSearch, str $sData) Diese Methode ersetzt beim Element mit der ID `$sTarget` alle Vorkommnisse des Strings `$sSearch` im Wert des Attributs `$sAttribute` mit dem String `$sData`.

addClear (string $sTarget, string $sAttribute) Löscht den Wert des übergebenen Attributs.

addAlert (string $sMsg) Zeigt ein Dialogfenster mit dem übergebenen String an.

addRedirect (string $sURL, integer $iDelay) Leitet den Browser auf die angegebene Adresse um. Es kann eine Verzögerung in Sekunden angegeben werden.

addScript (string $sJS) Führt den übergebenen Javascript-Code direkt aus.

addScriptCall ($sFunc, $arg0, $arg1, ...) Diese Methode ruft die übergebene JavaScript-Funktion mit den übergebenen Parametern aus.

addRemove (string $sTarget) Entfernt das Element mit der ID `$sTarget`.

addCreate (str $sParent, str $sTag, str $sId) Fügt einen neuen HTML-Tag vom Typ `$sTag` mit der ID `$sId` als Kindelement des Elements mit der ID `$sParent` ein.

addInsert (string $sBefore, string $sTag, string $sId) Fügt einen neuen HTML-Tag vom Typ `$sTag` mit der ID `$sId` vor dem Element mit der ID `$sBefore` ein.

addInsertAfter (string $sAfter, string $sTag, string $sId) Fügt einen neuen HTML-Tag vom Typ `$sTag` mit der ID `$sId` nach dem Element mit der ID `$sAfter` ein.

addEvent (string $sTarget, string $sEvent, string $sScript) Fügt dem Element mit der ID `$sTarget` einen neuen Event-Handler hinzu. Bei Eintritt des Ereignisses wird der übergebene Javascript-Code ausgeführt.

addHandler (string $sTarget, string $sEvent, string $sHandler) Fügt dem Element mit der ID `$sTarget` einen neuen Event-Handler hinzu. Bei Eintritt des Ereignisses wird die übergebene Javascript-Funktion ausgeführt.

addRemoveHandler (str $sTarget, str $sEvent, str $sHandler) Diese Methode entfernt einen Event-Handler.

4.3. Das Javascript-Objekt „xajax“

Das Javascript-Objekt „xajax“ ist das clientseitige Gegenstück zur PHP-Klasse „xajaxResponse“ und bietet größtenteils Methoden zur Verarbeitung der von dieser erzeugten Anweisungen an. Es ist jedoch zu beachten, dass ein Programmierer, der mit Xajax arbeitet, in der Regel nicht direkt auf dieses Objekt zugreift. Es bündelt vielmehr die Kernfunktionalität für die von Xajax automatisch generierten Javascript-Funktionen und wird entsprechend meist nur aus diesen heraus aufgerufen. (White u. Wilson, 2007c)

loadingFunction () Diese Funktion kann überladen werden, um eine eigene Funktion aufzurufen, wenn Xajax einen Request abschickt. Dies kann benutzt werden, um zum Beispiel einen Ladebalken anzuzeigen.

doneLoadingFunction () Diese Funktion kann überladen werden, um eine eigene Funktion aufzurufen, wenn Xajax eine Response erhält. Dies kann benutzt werden, um zum Beispiel einen Ladebalken wieder auszublenden.

call (sFunction, aArgs, sRequestType) Schickt einen XMLHttpRequest ab, um die angegebene PHP-Funktion aufzurufen. Wird in der Regel nicht direkt aufgerufen, sondern von den durch Xajax generierten Javascript-Funktionen.

viewSource () Zeigt den HTML-Quellcode der Seite *nach* Durchführung der Änderungen durch Xajax an.

4.4. Zukünftige Versionen

Wie bei vielen freien Softwareprojekten gibt es von Xajax eine stabile Version, die für die produktive Nutzung gedacht ist, und eine unstabile Version für Entwickler, zum Austesten neuer Funktionalitäten. Während der Funktionsumfang der aktuellen stabilen Version prinzipiell auf die Kapselung von Ajax-Aufrufen beschränkt ist, enthält die Entwicklerversion einige interessante Ansätze für erweiterte Funktionalität.

4.4.1. Objektorientiertes Erzeugen von HTML

Xajax soll künftig eine Abstraktionsschicht für HTML enthalten, mit der komplette Webseiten oder Teile objektorientiert erzeugt werden können (serverseitig mit PHP). In den aktuellen Entwicklerversionen ist dies schon ansatzweise vorhanden, aber noch sehr unvollständig und funktionsarm. Auch die Usability lässt noch Wünsche offen - so muss man zum Beispiel für jeden HTML-Tag, den man verwenden möchte, von Hand eine Datei einbinden (html.inc.php, body.inc.php, div.inc.php ...), statt auf dynamische Einbindung, wie etwa beim Zend Framework, zu setzen. Sollte dieses Feature jedoch einmal ausgereift sein, würde es Xajax um einiges attraktiver für komplette Neuentwicklungen von Webapplikationen machen.

4.4.2. Plugins

Es soll zukünftig für Entwickler möglich sein, oft gebrauchte Funktionen in Xajax-Plugins zu kapseln, welche dann einfach in Xajax-Projekte eingebunden werden können. Auf der Projekthomepage findet sich bereits eine Rubrik für Plugins mit nach Kategorien geordneten Downloads. Eine nähere Inspektion zeigt jedoch, dass viele Kategorien leer sind und es bisher insgesamt nur drei Plugins gibt, welche alle die Nutzung der unstabilen Version vorraussetzen.

4.4.3. JSON

Da das Xajax-eigene XML-Format den für XML typischen Overhead aufweist, ohne dies mit einem für Menschen gut lesbaren Quellcode auszugleichen, wurde von Xajax-Anwendern oft der Wunsch nach JSON-Unterstützung geäußert, was eine effektivere Datenübertragung ermöglichen würde. Die Xajax-Entwickler haben angekündigt, diese Funktion in zukünftigen Versionen umzusetzen, bisher existiert sie jedoch nicht einmal in den unstabilen Versionen.

5. Warum sollte man xajax einsetzen?

Dieser Abschnitt beleuchtet die Vor- und Nachteile von Xajax.

5.1. Vorteile

5.1.1. Quelloffenheit

Die Quelloffenheit von Xajax bringt die üblichen damit verbundenen Vorteile. Es gibt eine aktive Benutzergemeinschaft, welche in Foren und durch die Erstellung von Tutorials Hilfestellung geben kann. Durch die Wahl der LGPL- bzw. zukünftig BSD-Lizenz besteht auch kein großes Risiko der Kommerzialisierung, etwa in Form von Nutzungsgebühren. Dank der liberalen Lizenzen (besonders BSD) unterliegt die Verwendung von Xajax praktisch keinen Einschränkungen.

5.1.2. Lernaufwand

Für einen einzelnen PHP-Programmierer oder ein Team aus PHP-Programmierern und JavaScript-Programmierern ist der Einstieg in Xajax mit minimalem Lernaufwand verbunden.

- Ein PHP-Programmierer muss kein Javascript lernen, um die Serverseite zu programmieren.
- Ein Javascript-Programmierer muss kein PHP lernen, um die Clientseite zu programmieren.
- Keiner von beiden muss irgendeine Ahnung von Ajax haben.

5.1.3. Cross-Browser-Kompatibilität

Wie bereits erwähnt kapselt Xajax die Ajax-Funktionalität vollständig, so dass der Programmierer nicht damit in Berührung kommt. Dies führt auch dazu, dass dieser sich nicht um Probleme mit der Kompatiblität verschiedener Browser mit bestimmten Ajax-Vorgehensweisen kümmern muss. Dies übernimmt Xajax im Hintergrund. Die stabile Version unterstützt die meisten großen JavaScript-Engines, unter anderem die des Internet Explorers, Firefox und anderer Mozilla-Browser sowie Opera. Safari und Konqueror

werden zur Zeit nicht einwandfrei unterstützt, dies soll sich laut Entwicklern aber bis zur nächsten finalen Version geändert haben.

5.1.4. Integration

Da Xajax rein PHP-basiert ist, lässt es sich einfach mit anderen PHP-Bibliotheken und -Frameworks kombinieren. So wäre es durchaus denkbar, eine Webanwendung mit Hilfe eines Rapid Development Frameworks, zum Beispiel CakePHP oder Zend Framework, aufzubauen und Xajax zur Bereitstellung von Ajax-Funktionalität zu integrieren. Eine andere Möglichkeit wäre die Verwendung eines in PHP geschriebenen CMS, etwa Typo3 oder Joomla.

Auch clientseitig bietet sich eine Verwendung von Xajax in Kombination mit anderen Werkzeugen an. Xajax bietet selbst keine „schicken“ Effekte und Widgets, kann aber ohne Probleme zusammen mit entsprechenden clientseitigen Javascript-Frameworks oder -Bibliotheken verwendet werden.

5.2. Nachteile

5.2.1. Dokumentation

Die Dokumentation von Xajax ist leider weder vollständig noch ausführlich. Sie besteht größtenteils aus einer Aufzählung der verfügbaren Funktionen und einer kurzen Erklärung derer Bedeutung und Parameter. Für Hilfestellung ist man deshalb oft auf Tutorials oder das Forum angewiesen, wo sich der wahre Wert der oben erwähnten aktiven Benutzergemeinschaft zeigt.

Auch außerhalb des Internets findet sich relativ wenig zu Xajax. Hier wird deutlich mehr zu populäreren Ajax-Frameworks wie Prototype oder GWT publiziert. Insgesamt fanden wir nur zwei Bücher, in denen Xajax überhaupt erwähnt wird. Am umfangreichsten behandeln Mintert u. Leisegang (2006) das Thema, hier wird Xajax auf circa neun Seiten gewürdigt (sechs davon Funktionsreferenz).

5.2.2. Weiterentwicklung

Ein weiterer Nachteil von Xajax ist die unsichere und teilweise schleppend verlaufende Weiterentwicklung. An der stabilen Version gab es innerhalb der letzten zwei Jahre nur ein einziges Update. Features wie JSON werden auch schon seit langem versprochen, finden sich aber selbst in der instabilen Version bislang nicht. Da die Entwicklung stark auf das dreiköpfige Kernteam beschränkt ist, ist unklar, was passieren würde, wenn diese das Projekt aufgeben.

6. Fazit

Zusammenfassend lässt sich sagen, dass Xajax Entwicklern, welche bereits mit PHP vertraut sind, eine schnelle und effiziente Art bietet, mit minimalem Lernaufwand Ajax-Anwendungen zu entwickeln.

Aufgrund seiner beschränkten Funktionalität eignet sich Xajax weniger zum Aufbau vollständiger komplexer Webanwendungen, anders als zum Beispiel GWT oder Atlas. Dies muss jedoch kein Nachteil sein. Xajax konzentriert sich bewusst auf die Kernkompetenz „Ajax-Aufrufe kapseln“ und versucht überhaupt nicht, eine „eierlegende Wollmilchsau“ zu sein. Es ist ein spezialiertes Werkzeug mit einem klar definierten Zweck, dessen Stärke vor allem in der Zusammenarbeit mit anderen Werkzeugen liegt.

Somit eignet sich Xajax insbesondere für die nachträgliche Erweiterung bereits vorhandener Projekte, welche in PHP implementiert sind. Auch für neue Projekte, für die schon Rahmenbedingungen, wie etwa ein bestimmtes CMS, festgelegt worden sind, ist Xajax eine gute Wahl.

Ob sich an der Position von Xajax etwas ändern wird, wenn die geplanten neuen Funktionen hinzugefügt werden, muss die Zukunft zeigen. Es scheint inzwischen die Intention der Entwickler zu sein, Xajax weg vom kleinen Spezialtool zu führen und nach und nach zu einem umfassenden Anwendungsframework zu machen. Doch bis dahin ist es noch ein weiter Weg.

A. Beispiel „Dateibrowser“

```
<?

$root = ".";

require_once ("xajax_core/xajax.inc.php");
$xajax = new xajax();

$xajax->registerFunction("opendirectory");
$xajax->registerFunction("closedirectory");
$xajax->registerFunction("loadfilecontext");
$xajax->registerFunction("savefilecontext");

function closedirectory ($dirname) {
    $objResponse = new xajaxResponse();
    $objResponse->assign("right_$dirname", "innerHTML",
    "<img src=\"icons/closedfolder.gif\" />" . $dirname);
    $objResponse->assign("left_$dirname", "innerHTML",
    "<a onclick=\"xajax_opendirectory('$dirname');\">"
    . "<img src=\"icons/plusbottom.gif\" /></a>");
    return $objResponse;
};

function opendirectory ($dirname) {
    $objResponse = new xajaxResponse();
    $objResponse->assign("right_$dirname", "innerHTML",
    listdir($dirname));
    $objResponse->assign("left_$dirname", "innerHTML",
    "<a onclick=\"xajax_closedirectory('$dirname');\">"
    . "<img src=\"icons/minusbottom.gif\" /></a>");
    return $objResponse;
};

function loadfilecontext ($filename) {
    global $root;
    $objResponse = new xajaxResponse();
    $lines = implode('', file("$root/" . $filename));
```

```
    $objResponse->assign("filename", "value", $filename);
    $objResponse->assign("filecontent", "value", $lines);
    $objResponse->assign("status", "innerHTML", "File opened.");
    return $objResponse;
};

function savefilecontext ($filename, $text) {
    global $root;
    $handle = fopen("$root/" . $filename, "w");
    fwrite($handle, $text);
    fclose($handle);
    $objResponse = new xajaxResponse();
    $objResponse->assign("status", "innerHTML", "File saved.");
    return $objResponse;
};

$xajax->processRequest();

function listdir ($dirname) {
    global $root;
    $str = "<div class=\"filetable\">"
           . "<img src=\"icons/openfolder.gif\" />"
           . $dirname . "<table border=\"0\">";
    $d = dir("$root/$dirname");
    $next = $d->read();
    while (false !== ($entry = $next)) {
        $next = $d->read();
        if ($next == false)
            $str .= "<tr><td id=\"left_$dirname/$entry\""
                 . " class=\"filetable_lastleft\">";
        else
            $str .= "<tr><td id=\"left_$dirname/$entry\""
                 . " class=\"filetable_left\">";
        if (is_dir($entry)) {
            if (strcmp($entry, ".")
            && strcmp($entry, "..")) {
                $str .= "<a"
                     ."onclick=\""
                     ."xajax_opendirectory('$dirname/$entry');"
                     ."\">";
                if ($next == false)
                    $str .= "<img src=\"icons/plusbottom.gif\" />";
                else
                    $str .= "<img src=\"icons/plus.gif\" />";
```

```
                $str .= "</a></td><td id=\"right_$dirname/$entry\">"
                     . "<img src=\"icons/closedfolder.gif\" />" .
                 $entry .
                 "</td></tr>\n";
            }
        } else {
            if ($next == false)
                $str .= "<img src=\"icons/joinbottom.gif\" />";
            else
                $str .= "<img src=\"icons/join.gif\" />";
            $str .= "</td><td id=\"right_$dirname/$entry\">"
                 . "<a onclick=\"".
                 . "xajax_loadfilecontext('$dirname/$entry');\""
                 . ">"
                 . "<img src=\"icons/last.gif\" />"
                 . $entry . "</a></td></tr>\n";
        }
    }
    $d->close();
    return $str . "</table>";
};

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<?php
$xajax->printJavascript();
?>

<style type="text/css" media="screen">
.filetable {
font-family: monospace;
font-size: 12px;
}

.filetable table {
margin: 0px;
padding: 0px;
border-collapse: collapse;
}
```

```
.filetable tr {
margin: 0px;
padding: 0px;
}

.filetable td {
margin: 0px;
padding: 0px;
}

.filetable_left {
vertical-align: top;
background-image: url(icons/normal.gif);
background-repeat: repeat-y;
background-position: top left;
}

.filetable_lastleft {
vertical-align: top;
}
</style>

</head>

<body>

<table border="0">
<tr>
<td style="vertical-align: top;">
<?
echo listdir(".");
?>
</td>
<td><a id="status"></a><br />
Filename: <input id="filename" type="text" value=""\><br />
<textarea id="filecontent" name="filecontent"
cols="100" rows="30"></textarea><br />
<a
onclick="xajax_savefilecontext(
document.getElementById('filename').value,
document.getElementById('filecontent').value
);">Save</a>
</td>
</tr>
```

```
</table>
</form>
</body>
</html>
```

Literaturverzeichnis

[Mintert u. Leisegang 2006] Mintert, Stefan ; Leisegang, Christoph: *Ajax. Grundlagen, Frameworks und Praxislösungen*. December 2006

[White u. Wilson 2007a] White, Jared ; Wilson, J. M.: *Documentation: xajax.inc.php*. Version: Dezember 2007. `http://www.xajaxproject.org/wiki/Documentation:xajax.inc.php`, Abruf: 8. Januar 2008

[White u. Wilson 2007b] White, Jared ; Wilson, J. M.: *Documentation: xajaxResponse.inc.php*. Version: September 2007. `http://www.xajaxproject.org/wiki/Documentation:xajaxResponse.inc.php`, Abruf: 8. Januar 2008

[White u. Wilson 2007c] White, Jared ; Wilson, J. M.: *Documentation: xajax_uncompressed.js*. Version: September 2007. `http://www.xajaxproject.org/wiki/Documentation:xajax_uncompressed.js`, Abruf: 8. Januar 2008

[White u. Wilson 2007d] White, Jared ; Wilson, J. M.: *How to install xajax*. Version: September 2007. `http://www.xajaxprojekt.org/wiki/How_to_install_xajax`, Abruf: 7. Januar 2008

[White u. Wilson 2007e] White, Jared ; Wilson, J. M.: *Tutorial: Learn xajax in 10 Minutes*. Version: April 2007. `http://www.xajaxproject.org/docs/xajax-in-10-minutes.php`, Abruf: 8. Januar 2008

[White u. Wilson 2007f] White, Jared ; Wilson, J. M.: *xajax Multiplier*. Version: Juli 2007. `http://www.xajaxproject.org/examples/multiply/multiply.php`, Abruf: 8. Januar 2008