Bachelorarbeit

im Studiengang „Wirtschaftsinformatik“

Implementierung und Integration eines Ajax-Frameworks für die Erstellung eigenständiger Widgets in E-Commerce-Applikationen

Julian Harrer

Hochschule München

Fakultät für Informatik und Mathematik

März 2011

_____ _____

(Familienname, Vorname) (Ort, Datum)

_____ _____ / _____

(Geburtsdatum) (Studiengruppe / WS/SS)

ERKLÄRUNG

Gemäß §14 Abs. 5 Satz 2 APO i.V. m. § 35 Abs. 7 RaPO

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Abstract

Diese Arbeit beschreibt die Konzeption, Implementierung und Integration eines serverseitigen Ajax-Frameworks. Das Ajax-Framework stellt die Implementierung eines plattformunabhängigen Kommunikationsprotokolls dar, welches innerhalb dieser Bachelorarbeit konzipiert wurde. Zudem wurden zusätzliche Mechanismen wie ein Template-Mechanismus und ein Mechanismus für das Rendering von Komponenten in das Framework integriert. Die Client-Logik des Frameworks basiert auf dem JavaScript-Framework JQuery. Die Server-Logik wurde mit Hilfe der Java-Enterprise-Edition (JEE) implementiert. Die Integration des Frameworks wird beispielshaft anhand einer E-Commerce-Anwendung dargestellt.

Inhaltsverzeichnis

Abstract 5

1. Einleitung 9

1.1. Motivation 9

1.2. Aufgabenstellung 12

1.3. Aufbau 13

2. Grundlagen 14

2.1. Client-Server-Architektur 14

2.1.1. Das Hypertext Transport Protokoll 15

2.1.2. Dynamische Webanwendungen 18

2.1.3. Rich Client-Anwendungen 20

2.2. Ajax 21

2.2.1. HTML und CSS 21

2.2.2. JavaScript und DOM 23

2.2.3. XML 24

2.3. Web-Frameworks 25

2.3.1. Komponenten-basierte und aktionsbasierte Web-Frameworks 26

2.3.2. Client-zentrische und Server-zentrische Web-Frameworks 26

2.3.3. Ajax-Frameworks 27

3. Anforderungsdefinition und Einordnung 30

4. Spezifikation des Frameworks 33

4.1. Das Kommunikationsprotokoll AKP 33

4.1.1. Der AKP-Request 33

4.1.2. Der AKP-Response 34

4.1.3. Der Lebenszyklus einer AKP-Anfrage 37

4.1.4. Der Anfragekontext ... 39
4.1.5. AKP-Anweisungen ... 40
4.2. Besonderheiten des Frameworks ... 47
4.2.1. Der Template-Mechanismus ... 47
4.2.2. Das Rendering von Komponenten ... 50
4.2.3. Umgang mit nicht JavaScript-fähigen Clients ... 51
5. Details der Implementierung ... 52
5.1. Beschreibung der Entwicklungsumgebung ... 52
5.2. Grundlagen der Client-Logik ... 55
5.2.1. Das JavaScript-Objektmodell ... 55
5.2.2. Die Konfiguration des akf-Objektes ... 57
5.2.3. Die Client/Server-Kommunikation ... 57
5.3. Grundlagen der Server-Logik ... 59
5.3.1. Das Ajax-Response-Objektmodell ... 60
5.3.2. Die Ajax-Response-Tags ... 60
5.4. Implementierung des Template-Mechanismus ... 62
5.4.1. Die Server-Logik ... 62
5.4.2. Die Client-Logik ... 64
5.5. Implementierung des Komponenten-Renderings ... 64
5.6. Generierung von alternativen Inhalten für nicht JavaScript-fähige Clients ... 65
6. Integration des Frameworks ... 66
6.1. Beschreibung des Widgets ... 66
6.2. Umsetzung des Widgets ... 67
7. Ergebnisse und Fazit ... 69
Literaturverzeichnis ... 70

1. Einleitung

1.1. Motivation

Webtechnologien gewinnen zunehmend an Bedeutung bei der Entwicklung komplexer Geschäftsanwendungen. Dabei stellt das Internet für Unternehmen eine weltweit verfügbare Plattform für die Vermarktung verschiedener Leistungen über webbasierte Anwendungen dar. Die Leistungen bzw. Dienste eines Unternehmens werden dabei auf einem Webserver betrieben und sind über einen Browser zugänglich.

Die Vorteile von Webanwendungen gegenüber Desktopanwendungen sind eine meist kürzere Entwicklungszeit und eine schnellere Verfügbarkeit der angebotenen Dienste. Die Dienste einer Webanwendung können ohne die Installation zusätzlicher Software und unabhängig vom Betriebssystem eines Benutzers über einen Browser in Anspruch genommen werden. Dies ermöglicht es auch kleineren Unternehmen, Leistungen schnell und kostengünstig auf den Markt zu bringen.

Eine Aufgabe moderner Webtechnologien ist die Steigerung der Benutzerfreundlichkeit (Usability) einer Webanwendung. Seit der Entstehung des Internets bis heute veränderten sich die Anforderungen an die Benutzbarkeit von Webapplikationen. Ursprünglich wurden Webseiten dafür konzipiert Benutzern wissenschaftliche Inhalte über das Internet zur Verfügung zu stellen. Im Zuge der Weiterentwicklung grundlegender Web-Technologien wie dem Hypertext Transport Protocol (HTTP) und der Hypertext Markup Language (HTML) verbreiteten sich im Internet neue Webanwendungen außerhalb des wissenschaftlichen Kontextes. Im Zusammenhang mit der wachsenden Zahl an Internetnutzern und der zunehmenden Komplexität der Webanwendungen, stiegen auch die Ansprüche an die Bedienbarkeit einer Webanwendung.

Moderne Webanwendungen bieten ihren Benutzern eine komfortable und desktopähnliche Benutzeroberfläche mit interaktiven Funktionen. Sie müssen in der Lage sein, komplexe Dienste über eine benutzerfreundliche Oberfläche anzubieten. Um diesen Anforderungen gerecht zu werden, entstanden unterschiedliche Ansätze und Technologien, welche die Benutzerfreundlichkeit steigern und die Möglichkeiten bei der Entwicklung einer Webanwendung erweitern sollen.

Ajax ist eine verbreitete Technik für die Umsetzung benutzerfreundlicher Webanwendungen. Die Client/Server-Kommunikation innerhalb klassischer Webanwendung läuft synchron ab. Synchron bedeutet, dass die Webanwendung während einer Serveranfrage bis zur Antwort des Servers nicht in der Lage ist, auf weitere Benutzereingaben zu reagierten, oder sonstige Programmlogik auszuführen. Bei jeder synchronen Serveranfrage wird der gesamte Inhalt einer Webseite neu geladen, wobei ebenfalls Inhalte geladen werden, die sich nicht verändert haben. Dies erhöht zum einen die Reaktionszeit und zum anderen die Netzwerkbelastung einer Webanwendung.

Ajax-basierte Anwendungen umgehen die Nachteile klassischer Webanwendung mit Hilfe von asynchronen Serveranfragen. Über asynchrone Serveranfragen können Daten von einem Webserver geladen und von der Client-Logik der Anwendung verarbeitet werden, ohne die Benutzbarkeit einer Webanwendung einzuschränken. Somit können gezielt die Daten angefordert werden, die für eine bestimmte Aktion benötigt werden. Dies schont einerseits die Netzwerkressourcen und erhöht die Reaktionszeit einer Webanwendung. Des Weiteren muss ein Webserver nicht bei jeder Client-Anfrage den gesamten Inhalt einer Webseite generieren, sondern kann sich auf die Generierung einzelner Teilbereiche der Webseite beschränken. Abbildung 1 verdeutlicht den Unterschied asynchroner Serveranfragen gegenüber synchroner Serveranfragen. Die gepunkteten Abschnitte symbolisieren dabei die Wartezeit eines Clients während einer Serveranfrage. Bei einer synchronen Serveranfrage wird bei jeder Serveranfrage die Benutzbarkeit der gesamten Webanwendung unterbrochen. Während einer asynchronen Serveranfrage kann ein Client die Webanwendung weiterhin bedienen und ggf. neue Anfragen an den Server stellen. Ajax-basierte Webanwendungen können somit schneller als klassische Webseiten auf Benutzereingaben reagieren, da mit Hilfe von Ajax-Techniken nebenläufige Serveranfragen ausgeführt werden können.

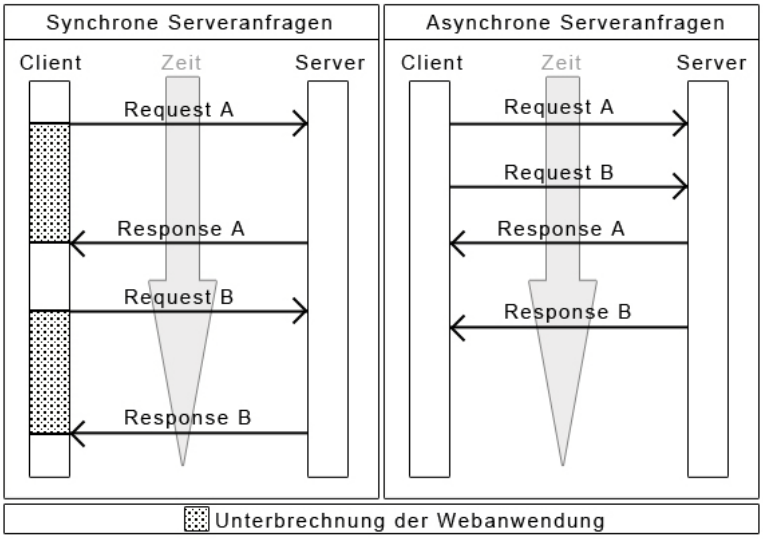

Abbildung 1 Vergleich - Asynchrones und Synchrone Serveranfrage

Da für die Umsetzung einer Ajax-basierten Webanwendung nur standardisierte Web-Technologien benötigt werden, muss ein Webentwickler für die Implementierung einer Ajax-Anwendung keine neuen Client-Technologien wie beispielsweise Adobe-Flash oder Microsoft Silverlight erlernen. Des Weiteren ermöglicht Ajax die Entwicklung von eigenständigen Seitenkomponenten die auch als Widgets bezeichnet werden. Widgets sind in sich abgeschlossene Teile einer Webseite, die unabhängig voneinander aktualisiert und bedient werden können.

Als einfaches Beispiel eines Ajax-Widget dient folgendes Tabpanel des Web-Shops www.alternate.de. Ein Tabpanel ist eine Seitenkomponente mit mehreren, sich überlappenden Inhalten (Tabs). Abbildung 2 zeigt das Tabpannel mit den Tabs „Ausführliche Details“, „Passendes Zubehör“, „Bewertung“ und „Preisentwicklung“. Es werden nur die Inhalte des aktiven Tabs angezeigt. In dieser Darstellung ist das der Tab Preisentwicklung.

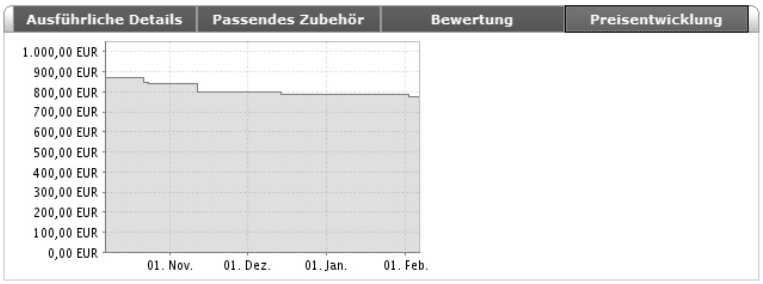

Abbildung 2 Alternate Tabpane - Quelle(www.alternate.de)

Bei einem Wechsel des Tabs wird der darzustellende Inhalt über einen asynchronen Ajax-Aufruf geladen und innerhalb des Widgets angezeigt. Während des Ladevorgangs kann der Rest der Webseite weiterverwendet werden.

Durch die Verwendung von Ajax-Techniken innerhalb einer Webanwendung wächst die Komplexität und somit der Entwicklungsaufwand einer Webanwendung im Gegensatz zu klassischen Webseiten. In der Praxis werden Ajax-Frameworks dazu eingesetzt die Entwicklung von Ajax-Anwendungen über eine einheitliche Client/Server-Kommunikation zu vereinfachen. Da es keine Vorgaben für die asynchrone Client/Server-Kommunikation einer Ajax-Anwendung gibt, entwickelten sich unterschiedliche Ansätze und Pattern für die Umsetzung von Ajax-Funktionalitäten. Ajax-Frameworks bieten Entwicklern von Webanwendungen vorgefertigte Funktionen oder Komponenten für die Implementierung von Ajax-Anwendungen. Dabei unterscheidet man zwischen rein clientseitigen JavaScript Frameworks und serverseitigen Frameworks die sowohl auf dem Client als auch auf dem Server betrieben werden.

1.2. Aufgabenstellung

Konzeption eines Kommunikationsprotokolls

Die Client/Server-Kommunikation einer Ajax-Anwendung ist nicht standardisiert. Jedes Framework verfolgt einen eigenen Ansatz bei der Vereinheitlichung der Kommunikation zwischen dem Server und den Webclients.

Die Spezifizierung eines Kommunikationsprotokolls für das Ajax-Framework soll die Implementierung des Frameworks auf verschiedenen Plattformen ermöglichen Die Aufgabe innerhalb der Konzeptionsphase des Kommunikationsprotokolls ist unter anderem die Definition eines einheitlichen Nachrichtenformats.

Das Protokoll soll u.a. folgende Fragestellungen behandeln:

- Wie läuft der Datenaustausch einer asynchronen Serveranfrage ab?
- Wie werden Änderungen von Seitenkomponenten durchgeführt?
- Wie können Fehler behandelt werden?
- Wie können erweiterte Funktionen in den spezifizierten Ablauf integriert werden?

Implementierung eines Ajax-Frameworks

Das plattformunabhängige Protokoll soll anschließend auf der Programmierplattform Java Enterprise Edition (JEE) implementiert werden. Das serverseitige Framework soll dabei keine vorgefertigten Komponenten bereitstellen, sondern einem Webentwickler als Hilfestellung bei der Entwicklung eigener Widgets dienen.

Die Aufgabe des Frameworks ist die Vereinheitlichung der Client/Server-Kommunikation über das spezifizierte Kommunikationsprotokoll. Dabei ist die Serverseite dafür verantwortlich Daten wie Inhalte oder Fehlermeldungen über ein vorgeschriebenes Format wie XML an den Client zu senden. Der Client hat die Aufgabe empfangene Serverantworten zu interpretieren und ggf. Änderungen in der Darstellung durchzuführen.

Folgende Anforderungen werden an das Framework gestellt:

- Einfache Erweiterbarkeit
- Einfache Integration in bestehende Projekte
- Berücksichtigung von nicht JavaScript-fähigen Clients
- Keine weiteren Abhängigkeiten zu serverseitigen Frameworks (nur JEE)

Integration des Ajax-Frameworks

Um die Möglichkeiten und Funktionen des Frameworks anhand eines konkreten Beispiels zu prüfen, wird das Framework in ein vorhandenes JEE-Projekt integriert. Dazu muss das Framework zunächst in die bestehende Architektur einer E-Commerce-Anwendung integriert werden. Im darauf folgenden Schritt wird ein Widget mit Hilfe des integrierten Frameworks umgesetzt. Dabei sollen möglichst viele Funktionen des Frameworks verwendet werden.

1.3. Aufbau

Das folgende Kapitel 2 vermittelt die grundlegenden Technologien und Ansätze von Webanwendungen. Kapitel 3 „Anforderungsdefinition und Einordnung“ beschreibt die Anforderungen des zu Implementierenden Frameworks. Nach der Definition der Anforderungen folgt die Beschreibung der Konzepte und Mechanismen des Frameworks in Kapitel 4 „Spezifikation des Frameworks“. Das darauf folgende Kapitel „Details der Implementierung“ gibt einen Einblick in die Implementierung des Ajax-Frameworks Nach der Darstellung der Implementierung folgt eine Beschreibung der Integration des Ajax-Frameworks. Hierbei werden die Schritte erläutert, die für die Integration des Frameworks und die Umsetzung des Widgets benötigt werden. Abschließend folgt eine Zusammenfassung der Ergebnisse dieser Bachelorarbeit in Kapitel 7 „Ergebnisse und Fazit“.

2. Grundlagen

Dieses Kapitel befasst sich mit den grundlegenden Konzepten und Technologien die bei der Umsetzung moderner Webanwendungen zum Einsatz kommen. Hierbei stehen vor allem Konzepte im Vordergrund, die innerhalb von Ajax-basierten Anwendungen relevant sind.

2.1. Client-Server-Architektur

Die Dienste einer Webseite werden auf einem Webserver betrieben und sind für verschiedene Clients über das Internet verfügbar. Für gewöhnlich werden Webseiten über das Netzwerkprotokoll HTTP aufgerufen. HTTP steht für Hypertext Transport Protokoll und dient u.a. der Abfrage von Serverressourcen. Normalerweise werden Browser als Web-Client eingesetzt um die Ressourcen eines Webservers mit Hilfe einer Serveranfrage (Request) zu ermitteln und die Serverantwort (Response) zu verarbeiten. Die Ressourcen eines Webservers sind über so genannte Uniform-Resource-Locator (URL) zugänglich. Eine URL beinhaltet unter anderem den eindeutigen Pfad zu einer Ressource (Basham, Sierra, & Bates, 2008). Beispiele für Ressourcen sind HTML-Dokumente, Bilder oder PDF-Dokumente. Abbildung 3 zeigt eine vereinfachte Darstellung einer Serveranfrage über das HTTP.

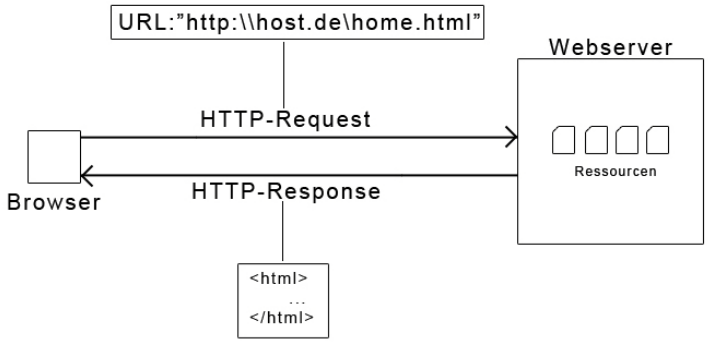

Abbildung 3 Vereinfachte Serveranfrage

In der Abbildung wird ein HTML-Dokument mit Hilfe des HTTP von einem Webserver angefragt. Der Webserver ist für das Auffinden der Ressourcen verantwortlich, er entscheidet anhand der URL, welche Ressource innerhalb der Serverantwort abgelegt werden soll und schickt eine HTTP-Antwort mit dem HTML-Dokument als Inhalt an den Browser zurück. Der Browser ist letztendlich für die Interpretation bzw. der Darstellung des HTML-Dokuments verantwortlich. Der Prozess der Darstellung innerhalb des Browsers wird auch als Rendering bezeichnet.

2.1.1. Das Hypertext Transport Protokoll

Die Kommunikation zwischen einem Browser und dem Webserver geschieht für gewöhnlich über das Hypertext Transport Protocol (HTTP). HTTP ist ein Netzwerkprotokoll und setzt auf den beiden darunterliegenden Protokollen TCP (Transmission Control Protocol) und dem IP (Internet Protocol) auf. Das HTTP regelt innerhalb einer Webanwendung die Kommunikation zwischen dem Webclient und dem Webserver über ein Request/Response-Modell.

Ein HTTP-Client initiiert eine Client/Server-Kommunikation über einen sogenannten HTTP-Request. Der Request beinhaltet u.a. die URL der angeforderten Ressource. Der Server empfängt den Request über einen bestimmten Port der Servermaschine. Nach der Verarbeitung der Anfrage sendet der Webserver bei Bedarf eine Antwort in Form einer HTTP-Response-Nachricht an den Client zurück.

Das HTTP ist ein zustandsloses Protokoll. Jeder Request eines Clients wird von dem Webserver unabhängig voneinander betrachtet. Das HTTP-Protokoll sieht bei der Kommunikation zwischen dem Client und dem Server keine Speicherung von Client-Informationen wie die Adresse eines Clients auf Seiten des Servers vor. Der Server ist somit nicht in der Lage einen sogenannten Conversation-State aufzubauen. Ein Conversation-State wird innerhalb von Webanwendungen benötigt um eine längerfristige Benutzersitzung (Session) aufrecht zu erhalten. Eine Benutzersitzung kann über das Setzen von Cookies auf dem Client aufgebaut werden. Mit Hilfe von Cookies können Informationen wie z.B. eine Session-ID auf dem Client hinterlegt werden, welche bei jedem HTTP-Request an den Server übertragen werden. Über diese Session-ID kann ein Client identifiziert und einer Benutzersitzung zugeordnet werden (Basham, Sierra, & Bates, 2008, S. 10).

HTTP-Nachrichten

Eine HTTP-Nachricht setzt sich aus einem Header und einem Body zusammen. Der Header beinhaltet Meta-Informationen, welche für die Kommunikation zwischen dem Client und dem Server von Bedeutung sind. Der Body-Teil einer Nachricht ist optional und enthält die eigentlichen Inhalte wie beispielsweise HTML-Dokumente, die an den jeweiligen Kommunikationspartner übertragen werden sollen.

Über einen *HTTP-Request* wird eine Client/Server-Kommunikation angestoßen. Die Art der Anfrage wird über die Angabe einer HTTP-Methode festgelegt. Man unterscheidet zwischen den HTTP-Methoden *OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE* und *CONNECT*. Innerhalb von Ajax-basierten Anwendungen kommen in den meisten Fällen nur die beiden HTTP-Methoden *GET* und *POST* zum Einsatz (w3.org - HTTP).

HTTP-GET-Request

Ein *GET*-Request dient der Abfrage von Serverressourcen. Der *GET*-Request besteht dabei lediglich aus dem Head-Teil einer HTTP-Nachricht. Bei dem Absenden einer *GET*-Anfrage können zusätzliche Anfrageinformationen in Form von Request-Parametern über die Erweiterung der URL angegeben werden. Das folgende Beispiel zeigt eine URL mit zusätzlichen URL-Parametern. Der URL-Teil, der die Parameter enthält, wird auch als Query-String bezeichnet und wird mit einem Fragezeichen eingeleitet. Die einzelnen Parameter werden über ein kaufmännisches Und-Zeichen voneinander getrennt.

```
http://www.example.de/index.jsp?parameter=value&secondParameter=newValue
```

Folgende Tabelle erläutert einige der wichtigsten *HTTP-Header* anhand einer *GET-Anfrage*:

	GET /resource/home.html HTTP/1.1 *Host: www.example.edu* *Connection: Keep-Alive* *Accept: text/html,text/xml* *Accept-Language: en-us* *Accept-Encoding: gzip, deflate* *Accept-Charset: utf-8-* *Agent: Mozilla/4.0 (compatible; MSIE5.0; Windows NT)*
Host	Dieser Header enthält den Hostnamen des Servers, an den die Anfrage gerichtet ist. In diesem Fall ist der Hostname www.example.edu und die gesamte URL der Anfrage http://www.example.edu/resource/home.html
Accept	Dieser *Header* enthält eine Auflistung der Typen, die von dem Client interpretiert werden können. Die Typen müssen in Form von *MIME-Typen* angegeben werden. Ein *MIME-Typ* besteht aus einem Medientyp und einem Subtyp. Der *MIME-Type „text/html“* steht beispielsweise für den Typen eines HTML-Dokuments, wobei „text“ für den Medientyp und „html“ für den Subtyp steht.
Accept-Language	Der *Accept-Language* Header vermittelt dem Server welche Sprache der Client bevorzugt, dabei steht *„en-us“* beispielsweise für amerikanisches Englisch.
Accept-Charset	Der *Accept-Charset*-Header gibt an welche Zeichencodes der Client interpretieren kann.

Tabelle 1 HTTP-Request-Header

HTTP-POST-Request

Mit Hilfe eines *POST*-Request können Daten an einen Server übertragen werden. Neben dem Head-Teil der Nachricht besitzt ein *POST*-Request zusätzlich einen Body-Teil mit den zu übertragenden Daten. Ein *POST*-Request wird beispielsweise beim Absenden eines Formulars übertragen. Der Vorteil bei der Datenübertragung über ein POST-Request ist, dass die Daten nicht wie bei einem GET-Request an die URL angehängt sondern sich im Body-Teil der Nachricht befinden. Somit können mehr Daten übertragen werden, da die Länge eines URL in der Regel begrenzt ist.

HTTP-Response

Eine *HTTP-Response-Nachricht* besteht ebenfalls aus einem Head- und einem Body-Teil. Der Body-Teil enthält dabei die Ergebnisse der Verarbeitung einer Serveranfrage. Einer der wichtigsten HTTP-Response-Header im Zusammenhang mit Ajax ist der *Content-Type-Header*. Dieser Header spezifiziert welche Art von Daten der Body einer HTTP-Response-Nachricht enthält. Dies wird wie bei dem Request-Header *Accept* über die Angabe eines MIME-Types geregelt. Im Fall einer Ajax-Anwendung könnte der Body einer HTTP-Response-Nachricht beispielsweise Daten in Form eines XML-Dokuments enthalten, wobei als Content-Type der Serverantwort der MIME-Type „*application/xml*“ oder „*text/xml*“ angegeben werden müsste.

Ein weiterer wichtiger Bestandteil der Header-Informationen einer HTTP-Antwort ist der Statuskode. Der Statuskode vermittelt dem Client ob die Anfrage erfolgreich verarbeitet werden konnte. Das HTTP spezifiziert neben einem Statuskode für die erfolgreiche Verarbeitung einer Anfrage unterschiedliche Kodes für verschiedene Verarbeitungsfehler. Die folgende Tabelle erläutert einige der wichtigsten Statuskodes (Gamperl, 2006).

Kode	Info	Beschreibung
200	*OK*	Dieser Statuskode vermittelt dem Client, dass Anfrage des Servers erfolgreich verarbeitet werden konnte
401	*Unauthorized*	Der Client der Serveranfrage ist für die angeforderten Serverressourcen nicht berechtigt.
404	*Not Found*	Die Angeforderte Ressource konnte über den Webserver nicht ermittelt werden.
408	*Request Time-out*	Die Serveranfrage konnte nicht innerhalb einer bestimmten Zeit verarbeitet werden.
500	*Internal Server Error*	Bei der Verarbeitung einer Serveranfrage ist ein interner Fehler aufgetreten.
503	*Service Unavailable*	Der Angeforderte Dienst ist vorübergehend nicht erreichbar.

Tabelle 2 HTTP-Statuskode

2.1.2. Dynamische Webanwendungen

Die Inhalte einer Webseite wie Bilder und Texte wurden ursprünglich statisch in die HTML-Dokumente einer Webseite eingebunden. Die Antwort einer Serveranfrage für eine bestimmte Ressource enthält in einem solchen Fall jedes Mal den gleichen Inhalt. Änderungen des Inhalts können dabei nur über die Anpassung der einzelnen Dokumente geschehen. Da solche Dokumente ausschließlich statische Inhalte enthalten spricht man auch vom „statischem Web“.

Über statische Dokumente ist es nur bedingt möglich komplexe Webapplikationen zu entwickeln, da für jeden möglichen Zustand einer Webanwendung ein eigenes Dokument erstellt werden müsste. Die Ausführung von Logik auf Seiten des Webservers ist bei diesem Ansatz nicht vorgesehen. Statische Webseiten werden den Anforderungen moderner Webanwendungen nicht gerecht, da sie ihren Benutzern weder maßgeschneiderte Inhalte noch interaktive Funktionen wie das Hinzufügen neuer Inhalte bieten können.

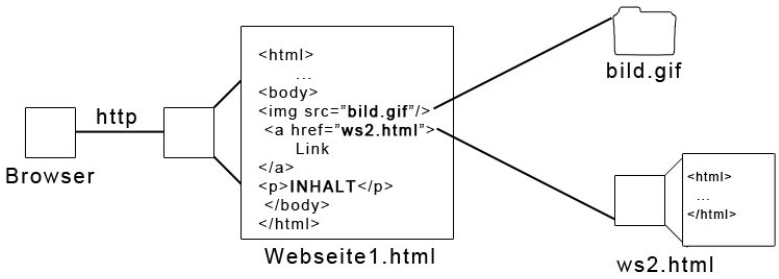

Abbildung 4 Statische Webanwendung

Der Unterschied von statischen zu dynamischen Webseiten ist, dass die Inhalte dynamischer Webseiten zur Laufzeit auf einem Webserver generiert werden. Der Server setzt bei modernen Webanwendungen, je nach Anfrage eines Clients, die einzelnen HTML-Bausteine der Webseite zusammen und sendet das Ergebnis als vollständiges HTML-Dokument an den Client zurück. Dadurch ist es beispielsweise möglich vor der Generierung der Seite, Daten wie den Namen des Benutzers oder Artikeldaten eines Internetshops, aus einer Datenbank abzufragen und in die Darstellung zu integrieren.

Die Generierung dynamischer Webseiten geschieht häufig über eine serverseitige Programmierplattform wie beispielsweise Java oder .Net. Die HTML-Dokumente werden also nicht wie bei statischen Webanwendungen als vorgefertigte Ressourcen auf dem Server gehalten sondern dynamisch mit Hilfe einer Programmiersprache generiert (Basham, Sierra, & Bates, 2008, S. 24).

Abbildung 5 verdeutlicht den Ablauf einer Serveranfrage, welche dynamische generierte Inhalte liefert.

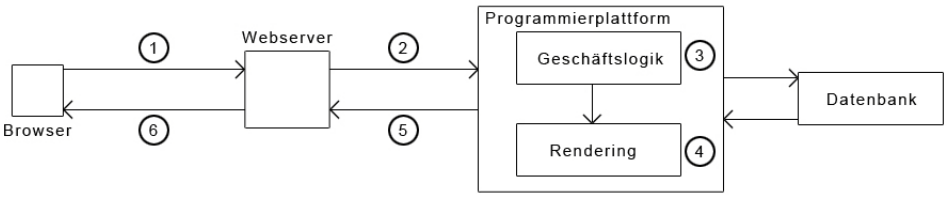

Abbildung 5 Ablauf der Serveranfrage einer dynamischen Webanwendung

Im ersten Schritt wird wie bei einer statischen Anfrage ein HTTP-Request an den Webserver gesendet. Dies geschieht beispielsweise über die Betätigung eines Hyperlinks oder das absenden eines Formulars.

Die Anfrage wird nun von dem Webserver an die Programmierplattform weitergeleitet (Schritt 2). Hierbei kann der Webserver entweder auf der gleichen Servermaschine wie die Programmierplattform betrieben werden oder über das Netzwerk mit der Plattform kommunizieren. Häufig wird die Programmierplattform über einen Applikationsserver bereitgestellt, der die Geschäftslogik der Webanwendung enthält.

Im nächsten Schritt (Schritt 3) wird die Geschäftslogik bzw. Anwendungslogik ausgeführt. Während der Ausführung der Geschäftslogik können Daten von externen Quellen wie beispielsweise einer Datenbank abgefragt werden. Die Geschäftslogik ist unter anderem für die Bereitstellung der für die Darstellung benötigten Daten verantwortlich.

Das serverseitige Rendering (Schritt 4) dient dazu, die gesammelten Daten in ein für den Browser verständliches Format wie HTML oder PDF zu bringen. Nach dem Render-Prozess wird das gesamte Ergebnis zunächst an den Webserver (Schritt 5) und dann mit Hilfe einer HTTP-Response-Nachricht an den Browser gesendet(Schritt 6). Die Aufgabe des Browsers ist nun die Darstellung der grafischen Oberfläche. Dem Browser ist dabei weder bekannt, wer die Serveranfrage verarbeitet hat, noch das der Inhalt dynamisch generiert wurde.

Der Ansatz dynamischer Webanwendungen bietet den Benutzern einer Webseite mehr Interaktionsmöglichkeiten und maßgeschneiderte Inhalte. So ist es beispielsweise möglich Benutzerdaten an eine Webanwendung zu senden, welche auf Seiten des Servers persistiert werden und über einen weiteren Request abgefragt werden können.

Der dynamische Charakter moderner Webanwendungen ermöglicht die Umsetzung von Ajax-basierten Anwendungen. Innerhalb von Ajax-Anwendungen wird das Ergebnis einer asynchronen Serveranfrage meist in Form von dynamisch generierten Inhalten an den Client gesendet. Diese Inhalte können beispielsweise Fragmente einer HTML-Seite oder Daten in Form von XML-Nachrichten darstellen.

2.1.3. Rich Client-Anwendungen

Als Rich-Client-Anwendung (RCA) werden Webanwendungen bezeichnet, die über eine ähnlich hochwertige Benutzeroberfläche wie Desktop-Anwendungen verfügen. Rich Internet Anwendungen stellen eine Mittellösung zwischen Thin- und Fat-Clients innerhalb der Client-Server Architekturen dar (Wähner, 2011).

Thin-Client-Architekturen lagern die gesamte Anwendungslogik in den Server aus. Der Client dient dabei als leichtgewichtiger Terminal und ist für die Weiterleitung von Benutzereingaben an den Server und der Darstellung der Ergebnisse verantwortlich. Der Server verarbeitet die Eingaben der Benutzer und sendet die Ergebnisse an den Client weiter. Über diesen Architekturansatz benötigt der Client weniger Ressourcen wie Arbeitsspeicher oder CPU.

Fat-Client-Architekturen halten die gesamte oder zumindest einen großen Teil der Anwendungslogik auf dem Client. Fat-Client-Anwendungen wie Desktop-Anwendungen können somit meist ohne Netzwerkanbindung verwendet werden. Zudem stellen Fat-Client-Lösungen ihren Benutzern meist eine komfortable Benutzeroberfläche zur Verfügung.

Rich-Client-Anwendungen halten die Anwendungslogik einer Webapplikation sowohl im Client als auch auf dem Server. Über diesen Ansatz kann die benötigte Rechenleistung einer Webanwendung zwischen dem Client und dem Server aufgeteilt werden. Ein weiterer Aspekt von RCAs ist der Benutzerkomfort. Rich-Client-Anwendungen bieten ihren Benutzern eine intuitive Benutzeroberfläche mit einer möglichst kurzen Reaktionszeit.

Ajax ermöglicht die Umsetzung von Rich-Client-Anwendungen über konventionelle Web-Technologien. Folgende Abbildung verdeutlicht den Unterschied zwischen den einzelnen Client/Server-Modellen.

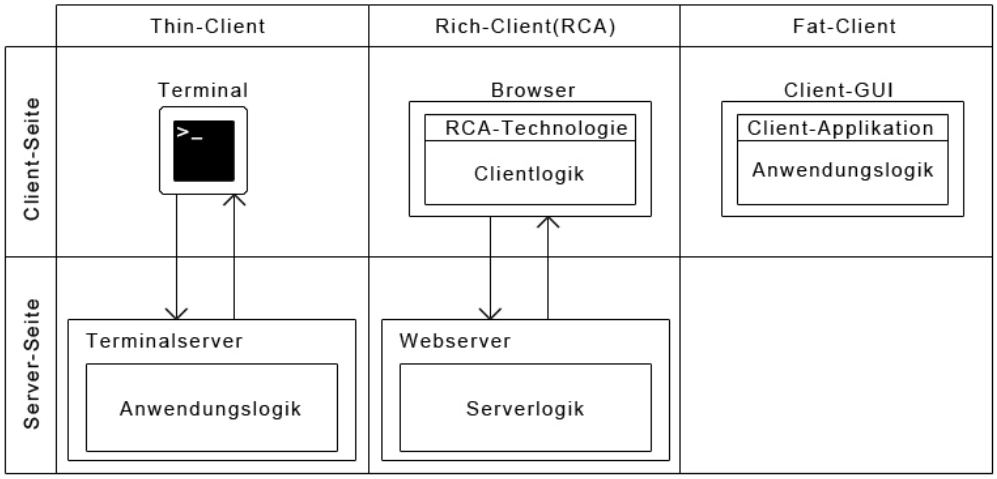

Abbildung 6 Vergleich von Client-Server-Architekturen

2.2. Ajax

Ajax ist ein Ansatz für die Implementierung von Rich-Internet-Anwendung. Im Gegensatz zu proprietären Technologien können Ajax-Anwendung über die Verwendung standardisierter Webtechnologien realisiert werden. Folgende Abschnitte erläutern die grundlegenden Technologien die bei der Umsetzung Ajax-basierter Anwendungen zum Einsatz kommen.

2.2.1. HTML und CSS

Die Hypertext Markup Language (HTML) ist eine vom World Wide Web Consortium (W3C) standardisierte Beschreibungssprache. Über die HTML kann die Struktur einer Webseite beschrieben werden. Die HTML kann von jedem gängigen Webbrowser wie dem Internet-Explorer oder Firefox interpretiert und dargestellt werden. Die Hauptaufgabe von HTML ist die Beschreibung des Layouts bzw. des Aufbaus einer Webseite.

Webdesigner haben zusätzlich die Möglichkeit das Aussehen einer Webseite über sogenannte Cascading-Style-Sheets (CSS) zu beeinflussen. CSS wurde ebenfalls über das W3C standardisiert und hat direkten Einfluss auf die Darstellung der HTML-Elemente einer Webseite. So werden in professionellen Webseiten beispielsweise die Farben und die Größe von Texten innerhalb eines HTML-Dokuments über ein CSS definiert.

HTML und CSS dienen also der Beschreibung der Benutzeroberfläche einer Webanwendung. Folgende Abschnitte vermitteln die Grundlagen dieser beiden Technologien.

Grundlagen von HTML

Ein HTML-Dokument setzt sich aus sogenannten Tags zusammen, welche ineinander verschachtelt werden können. Man unterscheidet zwischen alleinstehenden Tags ohne Inhalt und Tags mit Inhalten. Ein Tag mit Inhalt muss immer mit einem sogenannten End-Tag abgeschlossen werden. Einem Tag können mehrere Attribute zugewiesen werden. Code-Snipped 1 HTML-Grundstrukturzeigt die Grundstruktur eines einfachen HTML-Dokuments mit einer Überschrift und einem Text-Paragraph.

Das Tag *html* dient als Wurzel-Tag und leitet das Dokument ein. Innerhalb des *head*-Tags können u.a. Dateien wie CSS- oder JavaScript-Dateien in die Webseite eingebunden oder der Titel der Webseite definiert werden. Die darzustellenden Inhalte der Webseite werden innerhalb des *body*-Tags definiert. Hier befindet sich der eigentliche Inhalt der Webseite wie Tabellen, Texte und Bilder.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Beschreibung der Seite</title>
  </head>
<body>
  <h1>Eine Überschrift</h1>
  <p>Ein Textabschnitt</p>
</body>
</html>
```

Code-Snipped 1 HTML-Grundstruktur

Grundlagen von CSS

Eine CSS-Definition kann entweder direkt innerhalb eines HTML-Dokuments oder außerhalb des Dokuments über ein eigenes CSS-Dokument erfolgen. Die Attribute einer CSS-Definition beziehen sich auf ein oder mehrere HTML-Elemente. Es ist ebenfalls möglich eine CSS-Definition direkt innerhalb eines HTML-Tags über das *style*-Attribut an das jeweilige HTML-Element zu binden. Folgendes Beispiel zeigt eine simple CSS-Definition, welche die Farbe und die Schriftgröße der Überschriften eines HTML-Dokuments festlegt.

```
h1 {
    color: black;
    font-size:1em;
}
```

Code-Snipped 2 CSS - Definition

Die Definition beginnt mit der Angabe einer Liste von Selektoren. Die Selektoren legen fest, für welche HTML-Elemente die Definition gültig ist. Innerhalb des Blocks können Attributwerte angegeben werden, welche die Darstellung der HTML-Elemente beeinflusst.

HTML und CSS bilden die grundlegenden Technologien für die Darstellung der grafischen Oberfläche einer Ajax-basierten Webanwendung. Die Darstellung einer Webseite kann von der Clientlogik der Webanwendung durch die Manipulation von HTML-Elementen oder CSS-Definitionen beliebig angepasst werden. Die Clientlogik einer Ajax-Anwendung ist somit für die Darstellung der asynchron ermittelten Daten mit Hilfe von HTML und CSS zuständig. Die Manipulation der Darstellung erfolgt innerhalb einer Ajax-Anwendung über die clientseitige Skriptsprache JavaScript.

2.2.2. JavaScript und DOM

Innerhalb eines Browsers wird das HTML-Markup einer Webseite als baumartige Struktur abgebildet. Diese Struktur wird als Document-Object-Model bezeichnet. Das DOM dient innerhalb moderner Webanwendungen als zentrale Schnittstelle für den Zugriff auf die einzelnen Bestandteile einer Webseite (Gamperl, 2006). Mit Hilfe der clientseitigen Skriptsprache JavaScript kann die Struktur des DOM manipuliert werden, indem beispielsweise neue HTML-Elemente hinzugefügt oder bestehende Elemente entfernt werden. JavaScript bietet hierfür eine Schnittstelle für die Navigation innerhalb des DOM-Baumes. Über die Manipulation des DOM werden Änderungen der Darstellung einer Webseite ermöglicht, ohne die Webseite neu laden zu müssen. Die Logik für solche Änderungen wird innerhalb von JavaScript-Funktionen bereitgestellt und kann jederzeit von der Client-Logik einer Webanwendung ausgeführt werden. Der folgende Ausschnitt einer HTML-Seite zeigt die Manipulation des DOM über die Skriptsprache JavaScript anhand eines einfachen Beispiels.

```
<script type="text/javascript">
  function changeContent() {
    var firstText= document.getElementById("firstText");
    var parent = firstText.parentNode;
    parent.removeChild(firstText);
    parent.style.backgroundColor = "black";
  }
</script>

<div>
  <p id="firstText">Erster Absatz</p>
  <p>Zweiter Absatz</p>
  <a href="#" onclick="changeContent();return false;" >Test</a>
</div>
```

Code-Snipped 3 JavaScript Dom-Manipulation

In diesem Beispiel ist eine JavaScript Funktion zu sehen, die zunächst einen Textparagraphen innerhalb eines *div*-Elements entfernt und die Hintergrundfarbe des *div*-Elements verändert. Dazu wird zunächst das DOM-Element des zu entfernenden Paragraphen über die Funktion *getElementById()* selektiert und das Vater-Element über das Attribut *parentNode* ermittelt. Im nächsten Schritt wird der Textparagraph über die Funktion *removeChild()* aus dem DOM-Baum entfernt. Im nächsten Schritt wird die Hintergrundfarbe des *div*-Elements über das *style*-Attribut und dem Namen des CSS-Attributs *backgroundColor* angepasst.

Asynchrone Serveraufrufe über JavaScript

Neben der Manipulation des DOM wird JavaScript innerhalb einer Ajax-Anwendung dazu eingesetzt, asynchrone Anfragen an einen Server zu senden. Zu diesem Zweck steht einem Webentwickler das XMLHttpRequest-Objekt zur Verfügung. Dieses Objekt besitzt u.a. Funktionen für das Senden von asynchronen Anfragen und dem Abfangen der Serverantworten. Aus Gründen der Sicherheit dürfen Ressourcen mit Hilfe von JavaScript nur von der eigenen Domäne der Webanwendung geladen werden. Das XMLHttpRequest-Objekt durchläuft während einer Serveranfrage folgende Zustände(W3C, 2010):

Zustand	Beschreibung
UNSET	Das Objekt wurde konstruiert
OPENED	Die *open* Methode wurde aufgerufen. In diesem Zustand können beispielsweise die Anfrage-Parameter gesetzt werden.
HEADERS_RECEIVED	Alle HTTP-Response-Header wurden vom Client empfangen.
LOADING	Der Response-Body wird geladen
DONE	Der Request wurde vollständig beantwortet. Die Anfrage wurde entweder erfolgreich verarbeitet, oder es sind Fehler bei der Verarbeitung aufgetreten.

Tabelle 3 XMLHttpRequest Zustände

Dem XMLHttpRequest-Objekt kann vor dem Senden einer Serveranfrage ein Eventhandler übergeben werden. Dieser Eventhandler wird im Falle einer Zustandsänderung des XMLHttpRequest-Objekts automatisch aufgerufen. Der Event-Handler könnte beispielsweise bei einer erfolgreichen Verarbeitung der Serveranfrage die Inhalte der Serverantwort in die Darstellung der Webseite integrieren. Das Absenden von asynchronen Serveranfragen und die Manipulation des DOM-Baumes über JavaScript sind fundamentale Techniken, die bei der Implementierung einer Ajax-Anwendung eingesetzt werden.

2.2.3. XML

Die Logik einer Ajax-basierten Anwendung ist aufgeteilt in die Serverlogik und Clientlogik. Obwohl die Programmiersprache auf dem Server Anwendungsdaten wie beispielsweise Produktdaten anders repräsentiert als die Clientlogik, müssen diese Daten zwischen dem Client und dem Server ausgetauscht werden. Dies geschieht innerhalb von Ajax-basierten Anwendungen über ein plattform- und sprachenunabhängiges Nachrichtenformat. Die Verwendung der Extensible Markup Language (XML) stellt eine Möglichkeit dar, Daten über ein plattformunabhängiges Format abzubilden.

„Die Extensible Markup Language, abgekürzt XML, ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur. XML definiert dabei die Regeln für den Aufbau solcher Dokumente. (Gamperl, 2006, S. 127)“

Da XML-Dokumente ähnlich wie HTML-Dokumente eine baumartige Struktur besitzen, kann für den Zugriff auf XML-Inhalte ebenfalls das Document-Object-Model verwendet werden.

Abbildung 7 verdeutlicht die Rolle von XML innerhalb einer Ajax-Anwendung. In der Abbildung sendet ein Client eine asynchrone Serveranfrage über das XMLHttpResponse-Objekt an den Server. Auf der Seite des Servers wird zunächst die Geschäftslogik ausgeführt und anschließend die zu übertragenden Daten in ein XML-Format umgewandelt. Das Umwandeln eines Objekts in ein anderes Format wird auch als Marshalling bezeichnet(Springsource, 2006). Die XML-Daten werden im nächsten Schritt innerhalb des HTTP-Response-Bodys an den Client gesendet. Nach dem Empfangen der Antwort wird der Eventhandler aufgerufen, der für die Verarbeitung der Serverantwort konfiguriert wurde. Der Eventhandler hat nun die Aufgabe die Daten aus der XML-Antwort zu interpretieren und die Darstellung der Webseite über die Manipulation des DOM anzupassen.

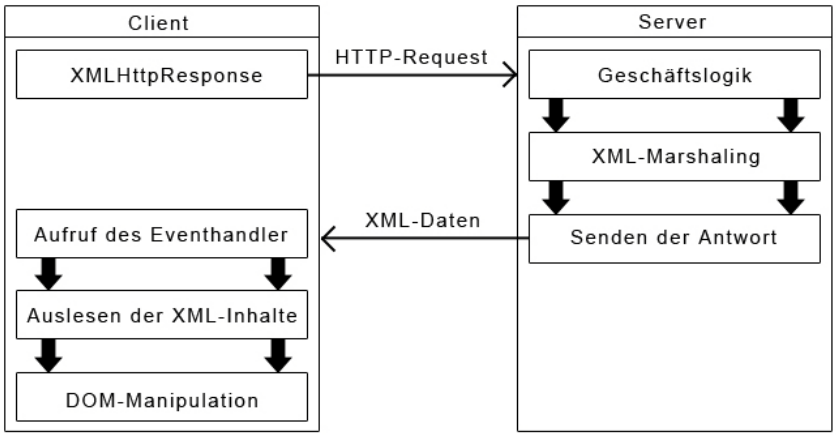

Abbildung 7 Ablauf einer asynchronen Anfrage mit XML

In der Praxis können neben XML weitere plattformunabhängige Datenformate wie JSON zum Einsatz kommen. JSON steht für JavaScript Object Notation und wird innerhalb von JavaScript für die Darstellung verschachtelter Objekte verwendet.

2.3. Web-Frameworks

Die Aufgabe eines Web-Frameworks ist die Bereitstellung von vorgefertigten Komponenten oder Mechanismen, welche die Umsetzung typischer Funktionen einer Webanwendung erleichtern sollen. Serverseitige Web-Frameworks werden unterteilt in aktionsbasierte- und komponentenbasierte-Frameworks. Des Weiteren unterscheidet man zwischen Client-zentrischen und Server-zentrischen

Frameworks. Die verschiedenen Arten von Web-Frameworks werden in den folgenden Abschnitten erläutert (Wähner, 2011).

2.3.1. Komponenten-basierte und aktionsbasierte Web-Frameworks

Komponentenbasierte Web-Frameworks

Viele serverseitige Web-Frameworks wie Java-Server-Faces bilden die einzelnen Bestandteile der Benutzeroberfläche einer Webanwendung auf ein serverseitiges Komponentenmodell ab. Änderungen der Benutzeroberfläche, wie beispielsweise das Hinzufügen eines Tabelleneintrags, werden dabei über die Anpassung des Komponentenmodells auf dem Server getätigt. Die Komponenten einer Webanwendung besitzen definierte Funktionen und werden innerhalb eines Render-Prozesses in HTML-Markup umgewandelt und in die Oberfläche der Anwendung integriert.

Aktionsbasierte Web-Frameworks

Aktions-basierte Web-Frameworks generieren die Benutzeroberfläche einer Webseite ohne die Haltung eines serverseitigen Komponentenmodells. Die Aktionen einer solchen Anwendung werden somit nicht von den Komponenten einer Anwendung sondern von einem Web-Controller behandelt, der die Anfrage eines Clients an eine bestimmte Serverressource wie einer JSP-Datei weiterleitet. In diesem Fall ist eine View-Technologie wie beispielsweise Java Server Faces oder Faclets für das komplette Rendering der Webseite verantwortlich.

2.3.2. Client-zentrische und Server-zentrische Web-Frameworks

Client-zentrische Web-Frameworks

Beim Client-zentrischen Ansatz befindet sich die gesamte GUI-Logik auf der Clientseite einer Webanwendung. Die GUI-Logik wird in diesem Fall beim initialen Laden der Webseite auf den Browser mit geladen. Neue Daten können in Form von XML oder einem anderen Format vom Server der Anwendung abgefragt und in die Webseite integriert werden. Über diesen Ansatz kann der Datenaustausch zwischen dem Client und dem Server minimiert werden, was vor allem bei langlaufenden Anwendungen von Vorteil ist.

Server-zentrische Web-Frameworks

Beim Server-zentrischen Ansatz geschieht die Generierung der Benutzeroberfläche auf der Seite des Servers. In diesem Fall werden nicht nur konkrete Geschäftsdaten, sondern auch GUI-Daten zwischen dem Server und dem Client ausgetauscht. Über diesen Ansatz kann die Ausführung der Geschäftslogik und die Generierung der Benutzeroberfläche zentral über die Server-Logik geregelt werden.

2.3.3. Ajax-Frameworks

Als Ajax-Framework können serverseitige, sowie rein clientseitige Web-Frameworks bezeichnet werden, die eine asynchrone Client/Server-Kommunikation über Ajax ermöglichen.

Die verschiedenen Ajax-Frameworks auf dem Markt verfolgen unterschiedliche Ansätze um die Entwicklung von Ajax-basierten Webanwendungen zu erleichtern. Das Google-Web-Toolkit (GWT) versucht beispielsweise die Entwicklung von Ajax-basierten Rich-Internet-Anwendungen ohne die Programmierung von JavaScript-Kode zu ermöglichen. Andere Web-Frameworks wie JQuery oder Dojo stellen reine clientseitige JavaScript-Frameworks mit erweiterten Ajax-Funktionen dar.

Ein serverseitiges Ajax-Framework spezifiziert die Kommunikation zwischen der Client- und der Server-Logik. Manche serverseitige Ajax-Frameworks stellen zudem vorgefertigte Ajax-Komponenten wie beispielsweise Validatoren oder Drag-And-Drop-Komponenten zur Verfügung.

Die Funktionsweise eines Ajax-Frameworks wird nun anhand des Web-Frameworks Java-Server-Faces verdeutlicht.

Das serverseitige Web-Framework Java-Server-Faces (JSF)

Seit Version 6 der Java Enterprise Edition ist das serverseitige Java-Framework Java-Server-Faces (JSF) ein fester Bestandteil des JEE-Standards. JSF wird als Framework innerhalb der Darstellungsschicht einer webbasierten JEE-Anwendung eingesetzt. Die HTML-Bausteine einer JSF-Webanwendung werden serverseitig als Komponentenbaum abgebildet. JSF kann somit als komponentenbasiertes Web-Framework bezeichnet werden. Manipulationen der serverseitigen Komponenten haben direkten Einfluss auf die Darstellung der Webanwendung (Prof. Müller, 2006, S. 10).

JSF-Komponenten-Modell

Abbildung 8 verdeutlicht das Komponenten-Konzept von JSF. Auf der Abbildung ist ein span-Element zu erkennen, welches für die Darstellung eines Textes auf dem Browser verwendet wird. Serverseitig wird dieses Element als sogenannte *UIComponent* über die Java-Klasse *HtmlOutputText* abgebildet. Der Wert des Textfelds ist an eine sogenannte Bean gebunden. Eine Bean ist in diesem Fall eine konventionelles Java Objekt, welches sich im Kontext von JSF befindet. Verändert sich der Wert dieses Java-Objekts, wird automatisch der Text des span-Elements auf der Seite des Clients angepasst.

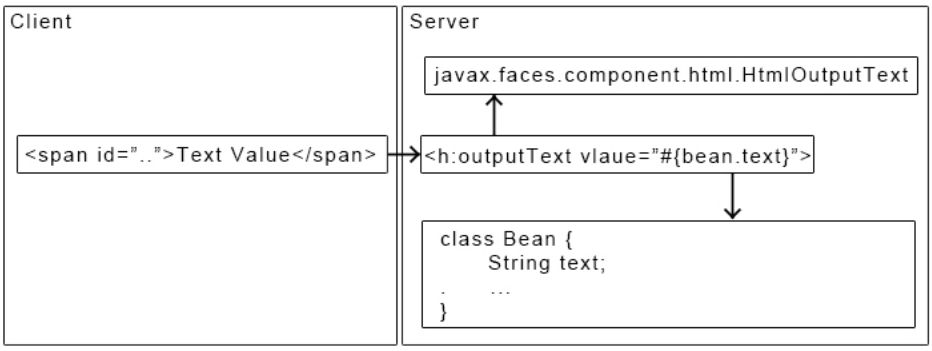

Abbildung 8 Java Server Faces Komponente

Das f:ajax-Tag

Das JSF-Framework bietet seit der Version 2.0 zusätzlich die Möglichkeit, die Komponenten einer Webanwendung mit Ajax-Funktionalität auszustatten. Dies geschieht über die Angabe des neu eingeführten JSF-Tags f:ajax. Über dieses Tag können Ajax-Events an die Komponenten einer JSF-Anwendung gebunden werden. Folgender Code-Ausschnitt zeigt eine JSF-Komponente, welche über das f:Ajax Tag erweitert wurde (Irian.at).

```
<h:selectBooleanCheckbox id="useCreditCard"
    value="#{customerBean.customer.useCreditCard}"
    valueChangeListener="#{customerBean.useCreditCardChanged}">
  <f:ajax render="ccData" />
</h:selectBooleanCheckbox>
```

Code-Snipped 4 JSF - Komponente mit f:ajax Funktion

In diesem Beispiel ist eine JSF-Komponente *selectBooleanCheckbox* zu sehen. Die Komponente selectBooleanCheckbox repräsentiert ein einfaches HTML-Kontrollkästchen. Die Komponente ist an einen serverseitigen Wert gebunden, der sich in einer Bean mit dem Namen *customerBean* befindet. Über das Attribut *valueChangeListener* kann ein Listener definiert werden, der automatisch von JSF aufgerufen wird, falls sich der Wert der Checkbox ändert. Ohne die Angabe des f:ajax Tags würde dies über einen synchronen Serveraufruf geschehen. Das f:ajax Tag sorgt dafür, dass bei einer Änderung der Checkbox ein asynchroner Serveraufruf abgesendet wird. Das Attribut *render* enthält die ID einer Serverkomponente, die nach der Abarbeitung des asynchronen Aufrufs neu generiert werden soll.

Das clientseitige JavaScript-Framework JQuery

Das JavaScript-Framework JQuery bietet neben anderen Spracherweiterungen zusätzliche Funktionen für die Verarbeitung von asynchronen Serveranfragen. Verantwortlich für das Senden einer Ajax-Anfrage ist die Funktion *jQuery.ajax()*. Als Argument erwartet die Funktion jQuery.ajax() ein Konfigurationsobjekt. Über die Konfiguration des Ajax-Aufrufes können die Eigenschaften wie die URL der Anfrage oder der erwartete Response-Typ definiert werden. Der Folgende Code-Ausschnitt zeigt einen einfachen Ajax-Aufruf mit Hilfe der jQuery.ajax()-Funktion.

```
$.ajax({
   url: “ajax/test.html”,
   success: function(data) {
     //Verarbeitung der Daten
   }
});
```

Code-Snipped 5 JQuery ajax Beispiel

In diesem Beispiel wird ein Ajax-Aufruf an den relativen URL-Pfad *„ajax/test.html“* gebunden. Des Weiteren wird ein Handler konfiguriert, der bei einer erfolgreichen Verarbeitung des Requests von jQuery aufgerufen wird. In der Regel ist ein solcher Handler für die Interpretation und die Darstellung der empfangenen Daten verantwortlich.

Neben dem Success-Handler können weitere Handler definiert werden, die während des Lebenszyklus einer Ajax-Anfrage von JQuery aufgerufen werden. Die möglichen Ereignisse eines JQuery-Ajax-Aufrufes werden in folgender Tabelle beschrieben (docs.jquery.com).

Zustand	Beschreibung
beforeSend	Dieses Ereignis wird direkt vor dem Senden einer Ajax-Anfrage ausgeführt. Innerhalb des Handlers für diesen Event können beispielsweise die HTTP-Header der Anfrage angepasst werden.
success	Dieses Ereignis wird aufgerufen, falls der Ajax-Request erfolgreich beantwortet werden konnte. Innerhalb des Handlers für den success Event wird in der Regel die Serverantwort verarbeitet.
error	Der error-Handler wird aufgerufen, falls bei der Kommunikation mit dem Server ein Fehler aufgetreten ist.
complete	Dieser Event wird am Ende des Request-Lebenszyklus aufgerufen.

Tabelle 4 Ereignisse einer JQuery-Ajax-Anfrage

3. Anforderungsdefinition und Einordnung

Das zu entwickelnde Ajax-Framework soll als Grundlage für die Entwicklung eigener Seitenkomponenten (Widgets) dienen. Dabei soll das Framework keine eigenen Seitenkomponenten in Form einer Komponentenbibliothek bereitstellen, sondern die Entwicklung solcher Komponenten erleichtern. Des Weiteren soll das Framework einfach in die bestehende Architektur einer JEE-Anwendung integriert und bei Bedarf wieder entfernt werden können.

Die Anforderungen an das Framework werden in den folgenden Absätzen vorgestellt. Die Anforderungen gelten sowohl für das zu entwickelnde Kommunikationsprotokoll als auch für die Implementierung des Frameworks.

Unabhängigkeit zu anderen serverseitigen Frameworks und Bibliotheken

Die Server-Logik des Frameworks soll auf standardisierten JEE-Technologien wie Java Server Pages und Servlets aufsetzen. Das Framework soll keine Abhängigkeiten zu anderen Bibliotheken außerhalb des JEE-Standards besitzen. Dies erleichtert die Integration des Frameworks, da einerseits keine Kompatibilitätsprobleme zwischen den verwendeten Bibliotheken entstehen können und andererseits keine weiteren Frameworks oder Bibliotheken in die bestehende Architektur eines Projekts integriert werden muss.

Des Weiteren sollte das Framework an keine spezielle View-Technologie gebunden sein. Dies ermöglicht die Entwicklung von Seitenkomponenten über unterschiedliche View-Technologien. Denkbar wäre das Rendering einer Serverantwort über Technologien wie JSP/JSPX, Servlets, Spring-View oder Facelets.

Keine Framework-spezifischen HTML-Tags oder Attribute

Viele Frameworks wie beispielsweise das clientseitige Framework Dojo binden eigene Framework-spezifische Tags oder Attribute innerhalb des HTML-Markup einer Webanwendung ein. Solche Tags oder Attribute sind zum einen nicht HTML-konform und erhöhen zum anderen die Komplexität des Markups. Das Framework soll die Gestaltung des HTML-Markups, ohne die Verwendung eigener HTML-Tags oder Attribute, ermöglichen (ApacheWicket).

Leichte Erweiterung und Anpassung des Frameworks an eigene Bedürfnisse

Das Framework soll einfach an die Bedürfnisse eines Projekts angepasst werden können. Dazu muss das Framework dem Entwickler bestimmte Freiheiten bei der Integration des Frameworks und der Entwicklung der Ajax-Komponenten lassen. So unterscheidet sich beispielsweise der Aufbau einer URL oder die Verarbeitungsweise einer Serveranfrage von Anwendung zu Anwendung. Das

Framework sollte so konzipiert sein, dass ein Entwickler bei der Integration des Frameworks nicht die bestehende Architektur des Projekts an das Framework, sondern vielmehr das Framework an die Projektarchitektur anpassen kann. Zudem sollten dem Entwickler über das Framework mehrere Lösungsmöglichkeiten für ein bestimmtes Problem breitgestellt werden.

Das Framework bietet grundlegende Ajax-Funktionen und vereinheitlicht die Kommunikation zwischen der Client- und Server-Logik. Diese grundlegenden Funktionen des Frameworks sollten von einem Entwickler über bestimmte Mechanismen erweitert werden können. Somit kann ein Entwickler eigene Funktionalität in den spezifizierten Ablauf des Frameworks integrieren und das Framework somit erweitern. Über diese Anforderungen soll auch die Entwicklung eigener Frameworks auf Basis des Ajax-Frameworks dieser Bachelorarbeit entwickelt werden können.

Bereitstellung eines Template-Mechanismus

Das Framework sollte über einen eigenen Template-Mechanismus verfügen, der die Definition von wiederverwendbaren GUI-Bausteinen ermöglicht. Ein Template dient dabei als eine Vorlage für bestimmte Teile einer Webseite. Das Template enthält Platzhalter, die innerhalb des Render-Prozesses mit Inhalten gefüllt werden. Mit Hilfe des Template-Mechanismus muss das HTML-Markup einer Webseite nicht redundant an verschiedenen Stellen einer Webseite eingefügt werden.

Effiziente Request-Verarbeitung

Das Framework sollte sowohl auf der Client- als auch auf der Server-Seite bestimmte Mechanismen bereitstellen, um die Leistungsfähigkeit einer Webseite zu steigern. Dies kann beispielsweise über eine Optimierung des serverseitigen Render-Prozesses oder das Caching von Templates auf dem Client geschehen.

XML-Kommunikation

Daten eines Servers sollten unter Anderem in Form von XML-Objekten an den Client übertragen werden. Neben der Übertragung von XML-Objekten sollte auch das Kommunikations-Protokoll auf XML basieren.

Plattformunabhängiges Kommunikationsprotokoll

Das zu entwickelnde Kommunikationsprotokoll sollte keine Abhängigkeiten zu einer bestimmten Programmierplattform besitzen. Dies wird durch die Verwendung des plattformunabhängigen Nachrichten-Format XML gewährleistet. Das plattformunabhängige Kommunikationsprotokoll ermöglicht eine Implementierung des Frameworks auf verschiedenen Plattformen, so könnte beispielsweise die Implementierung der Client-Logik mit verschiedenen serverseitigen Framework-Implementierungen kommunizieren.

Möglichkeit der Manipulation von Inhalten

Das Framework muss bestimmte Mechanismen oder Pattern bereitstellen, mit deren Hilfe die Benutzeroberfläche der Web-Anwendung geändert werden kann.

4. Spezifikation des Frameworks

Dieses Kapitel befasst sich mit der Spezifikation des Frameworks. Die folgenden Abschnitte erläutern das Kommunikationsprotokoll sowie die bereitgestellten Funktionen und Mechanismen des Frameworks. Im weiteren Verlauf dieser Arbeit wird das Framework als Ajax-Kommunikations-Framework (AKF) und das Kommunikationsprotokoll als Ajax-Kommunikations-Protokoll (AKP) bezeichnet.

4.1. Das Kommunikationsprotokoll AKP

Die folgenden Abschnitte beschreiben das Kommunikationsprotokoll, welches als Grundlage für die Implementierung des Ajax-Frameworks dienen soll.

Das AKP setzt auf dem Hyper-Text-Transport-Protocol (HTTP) auf und dient der Vereinheitlichung der Client/Server-Kommunikation über ein plattformunabhängiges Nachrichtenformat. Es soll die Entwicklung des Frameworks auf verschiedenen Plattformen ermöglichen. Das Protokoll definiert den Ablauf der Kommunikation und die Struktur der zu übertragenden Nachrichten. Als Nachrichtenformat des Protokolls wurde die XML gewählt.

Das AKP-Protokoll definiert, wie bestimmte Aktionen auf der Seite des Servers aufgerufen werden können. Des Weiteren spezifiziert das Protokoll Möglichkeiten für die Steuerung der clientseitigen Benutzeroberfläche mit Hilfe einer AKP-Response-Nachricht.

4.1.1. Der AKP-Request

Die AKP-Anfrage initiiert eine asynchrone Client/Server-Kommunikation mit Hilfe einer HTTP-Anfrage. Hierbei kann zwischen den beiden HTTP-Methoden POST und GET gewählt werden. Der Anfrage können zusätzliche Parameter hinzugefügt werden, welche für die Verarbeitung der Serveranfrage benötigt werden. Über die Angabe eines Anfrage-Parameters könnte beispielsweise die Aktion definiert werden, welche auf der Seite des Servers ausgeführt werden soll.

Der HTTP-Accept-Header einer AKP-Anfrage sollte den Mime-Type „application/xml“ bzw. „text/xml“ enthalten, da der Inhalt einer AKP-Response-Nachricht aus einem XML-Dokument besteht.

Der Anfrage-Parameter async

Einem AKP-Request muss ein HTTP-Parameter *async* mit dem Wert *true* angefügt werden. Dieser Parameter markiert den Aufruf als AKP-Request. Der Parameter *async* muss im Request enthalten sein, da asynchrone AKP-Aufrufe anders verarbeitet werden als konventionelle Serveraufrufe. So können AKP-Requests im Gegensatz zu synchronen Serveranfragen ausschließlich mit einer

bestimmten AKP-Response beantwortet werden. Mit Hilfe dieses Parameters können AKP-Requests von anderen Request-Arten unterschieden werden. Auf den Parameter *async* kann nur verzichtet werden, falls die Implementierung des serverseitigen Frameworks eine andere URL-Struktur vorsieht.

Da das zu implementierende Ajax-Framework eine einfache Integration in die Architektur bestehender Projekte ermöglichen soll, gibt das AKP keine Vorgaben über die Struktur bzw. den Aufbau einer Request-URL. Bis auf den Parameter *async* und weiteren optionalen Parameter kann die Request-URL frei gestaltet werden.

Das folgende Beispiel zeigt eine mögliche URL eines AKP-GET-Requests mit Angabe eines zusätzlichen Request-Parameters *test*.

```
http://www.beispiel.de/ressourcenpfad?async=true&testt=wert
```

Code-Snipped 6 URL einer AKP-GET-Anfrage

4.1.2. Der AKP-Response

Eine AKP-Request-Nachricht kann mit einer AKP-Response-Nachricht beantwortet werden. Der AKP-Response ist jedoch optional. Dadurch wird eine einseitige asynchrone Kommunikation mit dem Server ermöglicht, was die Netzwerkbelastung der Webanwendung schont. Diese Technik kann für die Übertragung von Nachrichten verwendet werden, für die keine Antwort erwartet wird. Abbildung 9 verdeutlicht den Ablauf einer AKP-Kommunikation mit einem AKP-Response.

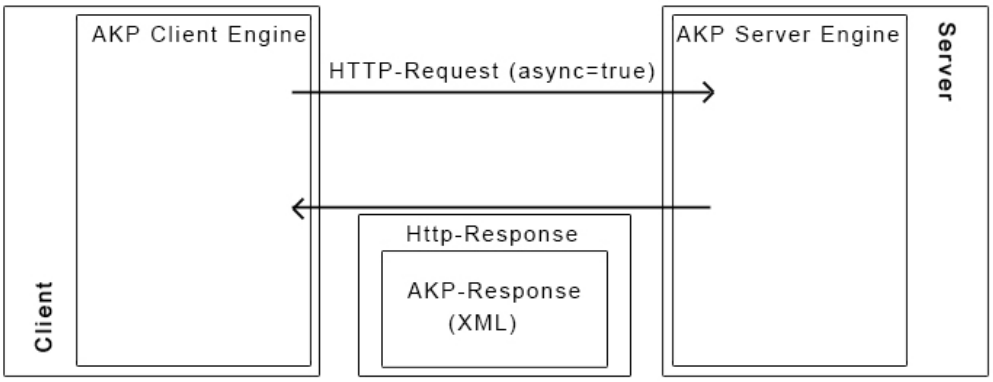

Abbildung 9 AKP-Kommunikation

Der AKP-Response befindet sich innerhalb des Body-Teils einer HTTP-Response-Nachricht. Der AKP-Response besteht aus einem XML-Dokument. Ähnlich einer HTTP-Nachricht enthält eine AKP-Response-Nachricht einen Head- und einen Body-Teil. Über die Aufteilung der Response-Nachricht in einen Head- und einen Body-Teil, können Meta-Informationen der Kommunikation wie Fehlermeldungen von den eigentlichen Inhalten und Anweisungen getrennt werden.

Der folgende Ausschnitt zeigt die Grundstruktur einer AKP-Response-Nachricht.

```
<ajaxresponse>
    <head>
        <!—META-Informationen →
    </head>
    <body>
        <!—Anweisungen und Templates →
    </body>
</ajaxresponse>
```

Code-Snipped 7 AKP - Ajaxresponse Grundsturktur

Das Tag *ajaxresponse* ist das Wurzelelement einer jeden AKP-Response-Nachricht. Der Head-Teil der Response-Nachricht enthält Informationen über die Verarbeitung der Serveranfrage. Der Body-Teil ist optional und beinhaltet Anweisungen und andere Daten, welche von der Clientlogik für die Anpassung der Benutzeroberfläche benötigt werden.

Der AKP-Response-Head

Der Head-Teil einer AKP-Response-Nachricht kann aus einem Fehlerschalter und mehreren Fehler-Nachrichten bestehen. Über den Fehlerschalter kann ermittelt werden, ob bei der Verarbeitung einer Serveranfrage Anwendungsfehler aufgetreten sind. Anwendungsfehler können zum Beispiel aufgrund fehlerhafter Anfrage-Parameter auftreten. Die Fehler müssen von der Anwendungslogik des Servers erkannt und in die Response-Nachricht eingefügt werden.

Um bestimmte Anwendungsfehler zu kategorisieren, kann einer Fehlermeldung ein Fehlerkode zugewiesen werden. Dieser anwendungsspezifische Fehlerkode ermöglicht der Client-Logik, auf bestimmte Anwendungsfehler zu reagieren. Die möglichen Fehlerkodes müssen von den Entwicklern der Webanwendung definiert werden. Die Angabe von Fehlerkodes ist vor allem für die Erweiterung bzw. Anpassung des Frameworks wichtig.

Abbildung 10 zeigt den Aufbau des Head-Teils einer AKP-Response-Nachricht.

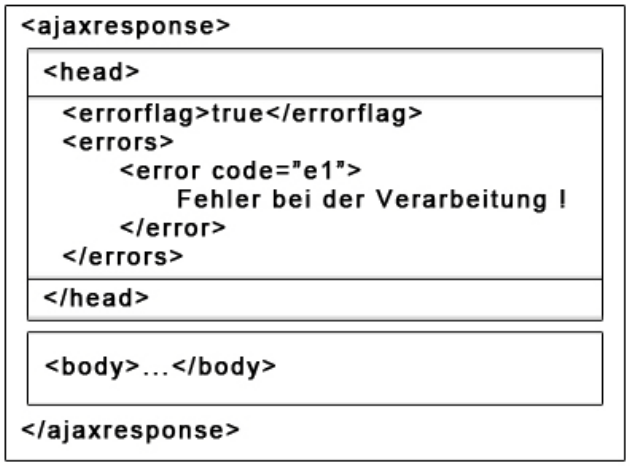

Abbildung 10 AKP-Head

Das *errorflag* ist in diesem Beispiel auf *true* gesetzt. Über diese Angabe wird der Client-Logik der Webanwendung vermittelt, dass ein Anwendungsfehler bei der Verarbeitung der Anfrage aufgetreten ist. Ist kein errorflag angegeben, oder ist das *errorflag* auf „*false*“ gesetzt, wurde die Serveranfrage erfolgreich verarbeitet.

Die Fehlermeldungen können innerhalb des Tags *errors* eingefügt werden. Jeder aufgetretene Fehler wird durch ein eigenes *error*-Tag repräsentiert. Der Fehlerkode einer Fehlermeldung kann über das Attribut *code* definiert werden.

Eine reine Bestätigungsnachricht ohne zusätzliche Inhalte könnte wie folgt aussehen:

```
<ajaxresponse>
    <head>
        <errorflag>false</errorflag>
    </head>
</ajaxresponse>
```

Code-Snipped 8 Head Teil einer AKP-Response

Der AKP-Response-Body

Der Body Teil einer AKP-Response-Nachricht kann Anweisungen (Actions) und Vorlagen (Templates) in Form von XML enthalten. Actions sind Anweisungen von Seiten des Servers, welche von der Client-Logik ausgeführt werden sollen. Mit Hilfe von Templates können Layout-Vorlagen als Grundlage für die Generierung von HTML-Seitenkomponenten entworfen werden. Die Action- und Template-Mechanismen werden in den folgenden Abschnitten im Detail beschrieben. Abbildung 11 zeigt die Grundstruktur des Body-Teils einer AKP-Response-Nachricht.

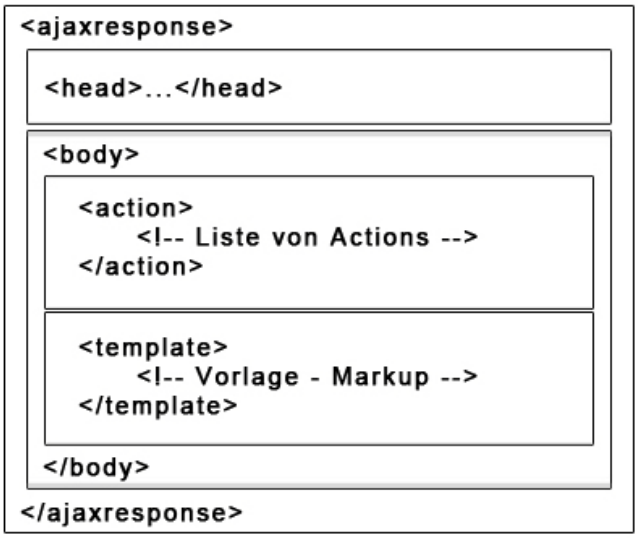

Abbildung 11 Body-Teil einer AKP-Response-Nachricht

4.1.3. Der Lebenszyklus einer AKP-Anfrage

Eine AKP-Anfrage durchläuft einen festgelegten Ablauf. Während dieses Ablaufs verändert sich der Zustand der Anfrage. Der Anfragezustand bestimmt unter Anderem welche Daten aus dem Anfragekontext abgefragt werden können und wie weit die Anfrage ausgeführt wurde. Die Entwickler einer Webanwendung haben die Möglichkeit über die Definition eigener Handler-Funktionen auf Änderungen des Anfrage-Zustands zu reagieren.

Jede AKP-Anfrage wird mit dem Zustand *beforeRequest* initialisiert. Während diesem Zustand können einer Anfrage zusätzliche Header und Parameter hinzugefügt werden. Direkt Nach dem Zustand *beforeRequest* wird die Anfrage an den Server gesendet. Die folgenden Abschnitte erläutern den Ablauf einer Anfrage, für unterschiedliche Ereignisse.

Normaler Ablauf einer Anfrage

Bei einem erfolgreichen Ablauf geht eine Anfrage, nach dem Senden der Anfrage und dem Empfang der Serverantwort, in den Zustand *afterResponse* über. Der Handler hat innerhalb dieser Phase Zugriff auf den Header- und Body-Teil der AKP-Response-Nachricht. Der Zustand afterResponse tritt vor der Verarbeitung des AKP-Response-Bodys auf. Während diesem Zustand wurden die Anweisungen der Serverantwort also noch nicht ausgeführt. Die Ausführung der Anweisungen des AKP-Response-Bodys findet direkt nach dem *afterResponse*-Zustand statt. Der Ablauf einer-Anfrage wird immer über den Zustand *afterProcessing* beendet. Folgende Abbildung zeigt den Ablauf einer erfolgreichen Anfrage.

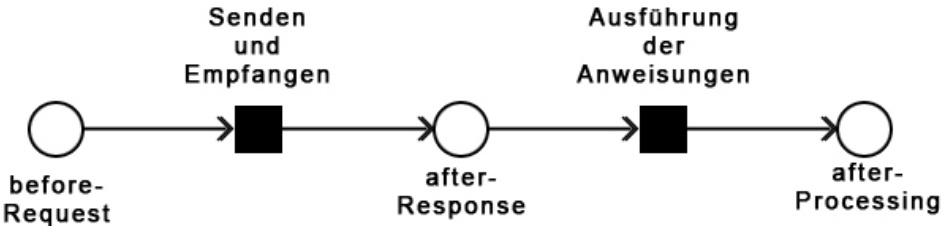

Abbildung 12 Zustandsübergänge einer erfolgreichen Anfrage

Anfragefehler

Falls bei der Kommunikation mit dem Server ein Fehler auftritt, geht die Anfrage von dem Zustand *beforeRequest* in den Zustand *requestError* über. Die Handler-Funktion hat während diesem Zustand die Möglichkeit genauere Fehlerinformationen über den Anfragekontext zu erhalten. Falls die Anfrage innerhalb der Handler-Methode abgebrochen wird, geht die Anfrage in den Zustand *requestAbort* über. Falls kein Abbruch stattfindet, wird die Anfrage automatisch mit dem Zustand *afterProcessing* beendet. Folgende Abbildung verdeutlicht den Ablauf einer Anfrage bei Auftreten eines Anfragefehlers.

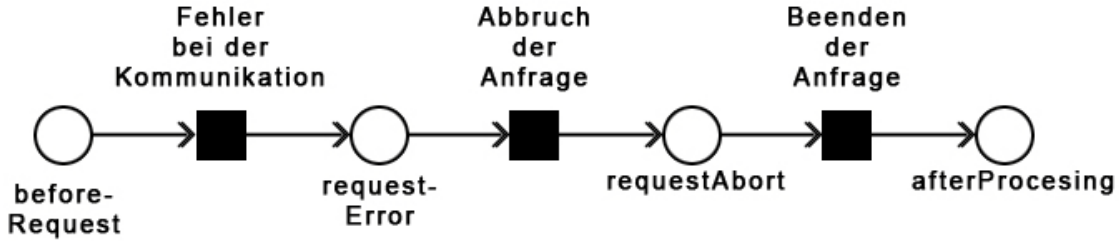

Abbildung 13 Zustandsübergänge bei Auftreten eines Anfrage-Fehlers

Anwendungsfehler

Nach dem Empfangen der Serverantwort, geht eine Anfrage in den Zustand *appError* über, falls der Fehlerschalter der AKP-Response-Nachricht auf „*true*“ gesetzt ist. Während dieses Zustands können die Fehlermeldungen des Head-Teils der Anfrage ausgelesen werden. Eine Anfrage kann während dieses Zustands über die Handler-Funktion der Anfrage abgebrochen werden. In diesem Fall wird die Ausführung des AKP-Response-Bodys übergangen. Folgende Abbildung verdeutlicht die Zustandsübergänge bei Auftreten eines Anwendungsfehlers.

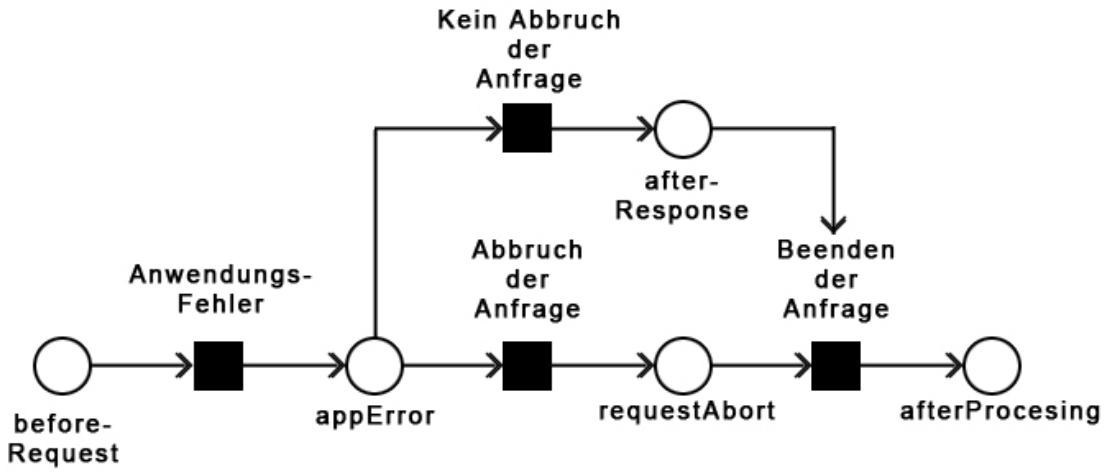

Abbildung 14 Zustandsübergänge bei Auftreten eines Anwendungsfehlers

4.1.4. Der Anfragekontext

Jede AKP-Anfrage besitzt einen Anfragekontext, der innerhalb der Client-Logik der Webanwendung gehalten wird. Dieser Kontext enthält alle Informationen wie beispielsweise den Zustand einer Anfrage und die AKP-Response-Nachricht. Alle beteiligten Funktionen wie die Handler-Funktion der Anfrage oder die Funktion der Script- bzw. Custom-Anweisung (siehe 4.1.5 AKP-Anweisungen), haben Zugriff auf diesen Kontext.

Der Kontext enthält ebenfalls das Element, welches eine AKP-Anfrage ausgeführt hat. Dieses Element wird als *caller*-Objekt bezeichnet. Das caller-Objekt kann beispielsweise dazu verwendet werden, das Vater-Element oder die Tabellenzeile des caller-Objekts innerhalb einer Script-Anweisung zu entfernen.

Des Weiteren ermöglicht der Kontext den Zugriff auf die AKP-Response-Nachricht der Anfrage. Somit kann jederzeit überprüft werden, ob ein Fehler aufgetreten ist, oder ob die Serverantwort einen Body-Teil besitzt.

Über den Anfragekontext, kann die Anfrage zudem Abgebrochen werden, wodurch die Anfrage in den Zustand *requestAbort* übergeht.

Einen Detaillierten Überblick über die Funktionen und Attribute des Anfragekontexts folgt in Kapitel 5.2.1 Das JavaScript-Objektmodell.

4.1.5. AKP-Anweisungen

Mit Hilfe von AKP-Anweisungen (Actions) kann die Server-Logik einer AKP-basierten Anwendung bestimmte Aktionen der Client-Logik auslösen. Über die Verwendung bestimmter Anweisungen können beispielsweise neue Inhalte in die Darstellung der Webanwendung eingefügt oder ein Redirect auf eine andere Webseite erzwungen werden. Die AKP-Anweisungen dienen also der Steuerung der Benutzeroberfläche.

Die Actions einer AKP-Response-Nachricht können innerhalb des Tags *action* im Body der Nachricht gebündelt werden, wobei die Client-Logik die einzelnen Anweisungen von oben nach unten abarbeitet. Als Standardanweisungen des AKP wurden die vier Aktionen *Script*, *Addcontent*, *Redirect* und *Custom* entworfen, welche im weiteren Verlauf dieses Kapitels vorgestellt werden.

Auf Basis der unterschiedlichen AKP-Anweisungen können Entwickler einer Webanwendung bestimmte Ajax-Pattern wie das Display-Manipulation-Pattern (ajaxpatterns.org) implementieren. Über die Kombination verschiedener Anweisungen soll die Umsetzung komplexer Anpassungen der Benutzeroberfläche ermöglichen.

Script-Anweisung

Die Script-Anweisung wir dazu verwendet einzelne JavaScript-Funktionen auf der Seite des Clients auszuführen. Über diese Anweisung kann ein Entwickler eigene Funktionen in den Verarbeitungsablauf der Response-Nachricht einfügen. Die definierte Funktion könnte beispielsweise Änderungen in der Benutzeroberfläche erwirken.

In diesem Fall muss ein Script-Tag mit einem Attribut *function* an den Client übertragen werden. Innerhalb des Attributs *function* wird dabei der Name der auszuführenden Funktion festgelegt.

Das folgende Beispiel verdeutlicht die Funktionsweise der Script-Anweisung. Abbildung 15 zeigt ein Test-Widget, mit dem die Zeilen einer Tabelle gelöscht, hinzugefügt und markiert werden können. Der Link „Add new Row“ besitzt einen Zähler, welcher die Anzahl der Tabellenzeilen angibt.

RowName	Time	Delete
Initial Request	Thu Mar 03 21:43:08 CET 2011	Delete - HighLight
Ajax Request	Thu Mar 03 21:43:10 CET 2011	Delete - HighLight

Add new Row (2)

Abbildung 15 Script-Anweisung Beispiel

Der folgende Kode-Ausschnitt zeigt eine AKP-Response-Nachricht, die das Entfernen einer Tabellenzeile und das Aktualisieren des Zählers erwirkt. Die JavaScript-Funktion *refreshRowCount* ist dabei für die Aktualisierung des Zählers verantwortlich. Der angegebenen Methode werden automatisch bestimmte Informationen wie z.B. der Request-Kontext der Anfrage übergeben. Über

diesen Kontext hat die Methode Zugriff auf bestimmte Informationen wie beispielsweise den Inhalt der Request- oder Response-Nachrichten.

```
...
<body>
    <action>
        <custom type=”deleteCallerRow” />
        <script function=“refreshRowCount“/>
    </action>
</body>
...
```

Code-Snipped 9 Script-Anweisung innerhalb einer AKP-Response

Redirect-Anweisung

Diese Anweisung kann dazu verwendet werden einen Redirect auf dem Client zu erzwingen. Die Redirect-Anweisung sollte immer am Ende einer Anweisungsliste angehängt werden, da der Redirect die Verarbeitung der Anweisungsliste unterbricht. Das XML-Element der Redirect-Anweisungen ist *redirect*, die URL auf die der Client Umgeleitet werden soll muss über das Attribut *url* definiert werden. Das *url*-Attribut muss angegeben sein, um den Redirect ausführen zu können.

```
...
<body>
    <action>
        <redirect url=“www.google.de“/>
    </action>
</body>
...
```

Code-Snipped 10 Redirect-Anweisung innerhalb einer AKP-Response

Addcontent-Anweisung

Die Addcontent-Anweisung dient dazu neue Inhalte in die Benutzeroberfläche der Webanwendung zu integrieren. Eine Addcontent-Anweisung besteht dabei immer aus einem Addcontent-Tag mit dem Attribut containerid und einem Render-Typ. Der zu integrierende Inhalt ist innerhalb des Addcontent-Tags enthalten. Die containerid der Anweisung bestimmt das Vater-Element, in das der neue Inhalt kopiert werden soll. Der Render-Typ gibt an, wie der enthaltene Inhalt zu generieren ist.

Der zu integrierende Inhalt kann dabei über die drei Render-Typen *markup, method* und *template* generiert werden. Jeder Render-Typ verfolgt einen anderen Ansatz bei der Generierung des HTML-Markups. Tabelle 5 stellt die verschiedenen Eigenschaften der einzelnen Render-Typen gegenüber.

Render-Typ	Inhalt	Generierung	Pflichtattribute
markup	HTML	Server	containerid
method	Benutzerdefiniert	Client	containerid, renderfunction
template	XML	Client	containerid, templateid

Tabelle 5 Eigenschaften der Addcontent-Render-Typen

Der Render-Typ markup

Über den Render-Typ *markup* kann der Inhalt, der in die Webseite eingefügt werden soll, als HTML-Markup innerhalb des Addcontent-Tags übertragen werden. Abbildung 16 verdeutlicht den Ablauf des Render-Prozesses bei Angabe des Render-Typs *markup.* Bei diesem Render-Typ ist der Server für das Rendering des HTML-Markup verantwortlich. Die Client-Logik hat die Aufgabe, das Markup aus der Serverantwort auszulesen und in die Webseite einzufügen. Mit Hilfe dieses Render-Typs ist die Umsetzung des HTML-Message-Pattern (ajaxpattern.org - Message Pattern) möglich.

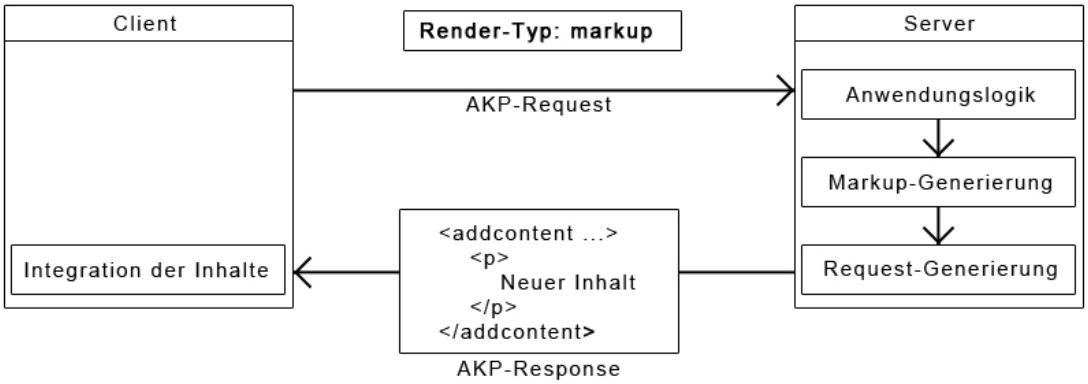

Abbildung 16 Render-Typ markup

Der folgende Kode-Ausschnitt zeigt eine Addcontent-Anweisung mit Angabe des Render-Typ *markup*. In diesem Fall wird der Inhalt des Addcontent-Tags in das HTML-Element mit der Id „*divContainer*“ kopiert.

Da HTML-Markup nicht XML-konform ist, muss der Inhalt in einen CDATA-Abschnitt eingebettet werden. Der CDATA-Abschnitt sorgt dafür, dass der XML-Parser des Browsers nicht versucht den HTML-Inhalt als XML zu interpretieren. Der CDATA-Abschnitt könnte im Falle von XHTML-Inhalten entfernt werden.

```
...
<body>
    <action>
        <addcontent containerid=“divContainer“ rendertype=”markup”>
            <![CDATA[ <p>Neuer Inhalt</p> ]]>
        </addcontent>
    </action>
</body>
...
```

Code-Snipped 11 Addcontent-Anweisung innerhalb einer AKP-Response

Render-Typ method

Über den Render-Typ *method* kann das Rendering des Inhalts mit Hilfe einer Render-Funktion geschehen. Hierbei muss dem Addcontent-Tag zusätzlich der Name einer Render-Funktion über das Attribut *renderfunction* übergeben werden. Die Render-Funktion ist dabei für das generieren des HTML-Markup zuständig. Bei diesem Render-Typ kann der Inhalt des Addcontent-Tags beliebig gestaltet werden. Auch in diesem Fall dürfen nur XML-konforme Inhalte ohne die Angabe eines CDATA-Abschnittes eingefügt werden. Abbildung 17 verdeutlicht den Render-Prozess des Render-Typ *method*. Dieser Render-Typ ermöglicht die Umsetzung des XML-Message-Pattern (ajaxpatterns.org - XML_Message).

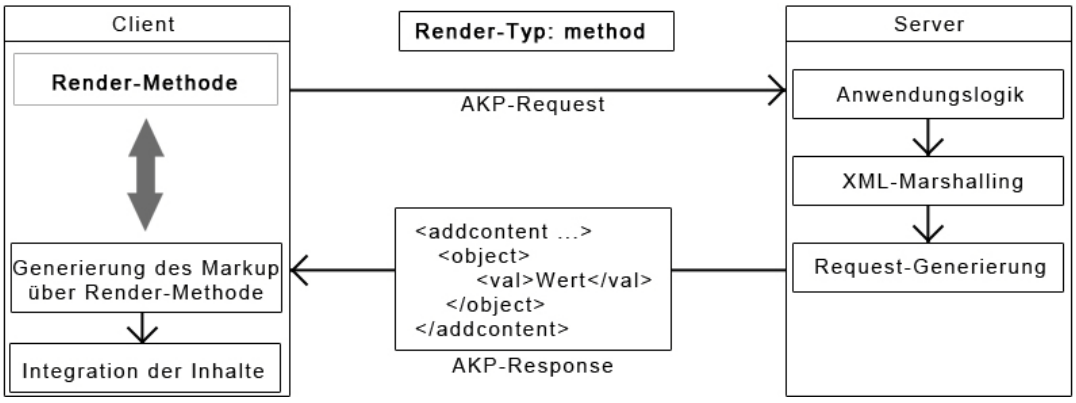

Abbildung 17 Render-Typ method

Folgendes Beispiel zeigt eine Addcontent-Anweisung mit dem Render-Typ *method.* Als Inhalt wird in diesem Beispiel ein Artikel-Objekt in Form von XML übergeben. Die JavaScript-Methode *renderArticleXML* ist dafür verantwortlich den XML-Inhalt zu parsen und in HTML umzuwandeln. Der generierte Inhalt wird dann von der Client-Logik in den angegeben Container eingefügt. Der Render-Typ *method* kann beispielsweise für komplexe XML-Objekte, JSON-Objekte oder andere Formate verwendet werden.

```
...
<body>
    <action>
        <addcontent containerid=“divContainer“ rendertype=”method”
            renderfunction=”renderArticleFromXML”>
            <Article>
                <id>15</id>
                <name>Articlename</name>
            </Article>
        </addcontent>
    </action>
</body>
...
```

Code-Snipped 12 Addcontent-Anweisung mit Render-Typ method

Render-Typ template

Über den dritten Render-Typ *template* kann die Generierung des HTML-Markups mit Hilfe einer Template-Vorlage geschehen. Die Id des zu Verwenden Templates muss dabei über das Attribut *templateid* angegeben werden. Dies ermöglicht die Umsetzung des Browser-Side-Templating-Pattern (ajaxpatterns.org - Templating).

Der Inhalt des Addcontent-Tags wird in diesem Fall immer in Form eines XML-Objekts übergeben. Der Inhalt des XML-Objekts wird zunächst von der Client-Logik der Webanwendung in das definierte Template eingefügt und dann in die Webseite integriert.

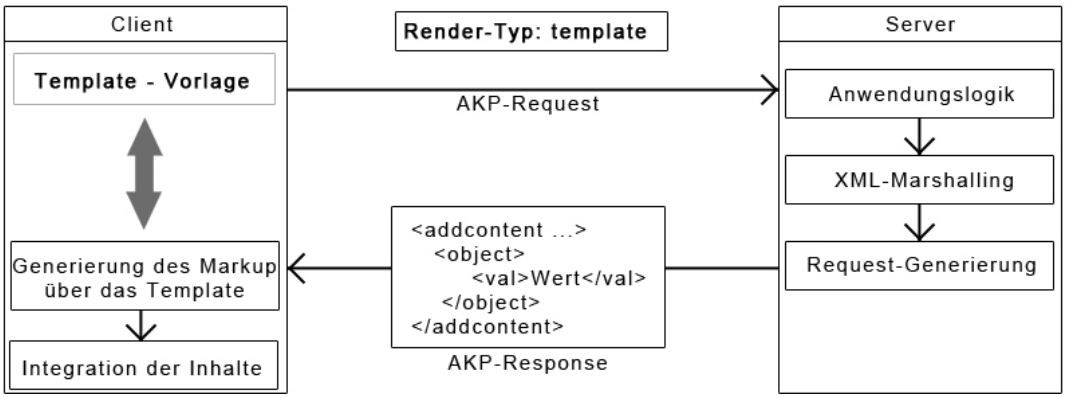

Abbildung 18 Render-Typ template

Der folgende Kode-Ausschnitt zeigt eine Addcontent-Anweisung mit dem Render-Typ *template.* In diesem Beispiel wird ebenfalls ein Artikel-Objekt in Form von XML innerhalb des Addcontent-Tags

übergeben. Der Inhalt des XML-Objekts wird auf der Seite des Clients in eine HTML-Vorlage eingefügt.

```
...
<body>
    <action>
        <addcontent containerid=“divContainer“ rendertype=”template”
            templateid=”articleTemplate”>
            <Article>
                <id>15</id>
                <name>Articlename</name>
            </Article>
        </addcontent>
    </action>
</body>
...
```

Code-Snipped 13 Addcontent-Anweisung mit Render-Typ template

Render-Methoden

Neben der Angabe eines Render-Typ, der für die Generierung der Inhalte verantwortlich ist, kann zusätzlich eine Render-Methode angegeben werden. Diese Render-Methode legt fest, wie der generierte Inhalt in die Webseite integriert werden soll. Das AKP unterscheidet zwischen den Render-Methoden *override*, *append* und *prepend*.

Render-Methode	Beschreibung
override	Überschreibt den Inhalt des Vater-Elements mit dem neuen Inhalt
append	Hängt den neuen Inhalt an die Kind-Elemente des Vater-Elements
prepend	Fügt den neuen Inhalt als erstes Kind-Element des Vater-Elements ein.

Tabelle 6 Beschreibung der Render-Methoden der Addcontent-Anweisung

Abbildung 19 verdeutlicht den Unterschied zwischen den einzelnen Render-Methoden. Falls keine Render-Methode definiert ist, wird der Inhalt automatisch mit Hilfe der Render-Methode *override* in die Seite eingefügt.

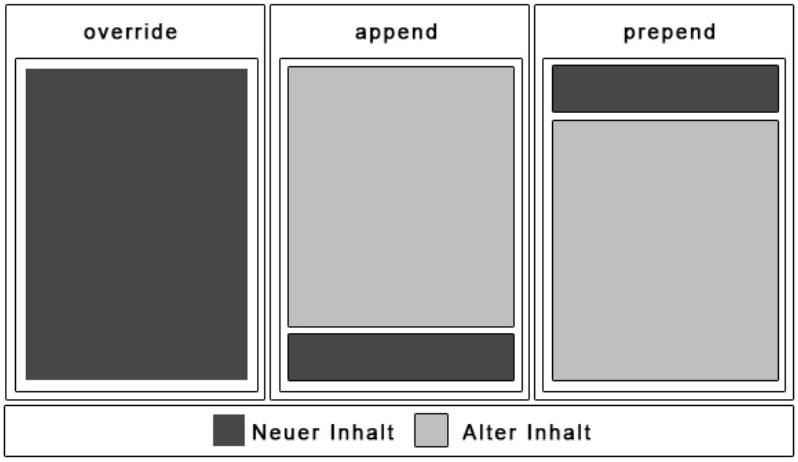

Abbildung 19 Unterscheidung der Render-Methoden einer Addcontent-Anweisung

Custom-Anweisung

Über Custom-Anweisungen können zusätzliche Anweisungen in das Framework integriert werden. Hierzu muss zunächst eine Anweisung in Form einer JavaScript-Funktion innerhalb der Client-Logik registriert werden. Die Funktion einer Custom-Anweisung wird mit einer eindeutigen Typbezeichnung in das Framework eingebunden. Anhand dieser Typbezeichnung kann die Funktion einer Custom-Anweisung ermittelt und ausgeführt werden.

Der Funktion der Custom-Anweisung werden zwei Argumente übergeben. Ein Argument ist der Request-Kontext, über den die Funktion Zugriff auf den Inhalt der Request- und Response-Nachricht hat. Das zweite Argument ist das Action-Objekt, welches Informationen über die Anweisungen selbst enthält.

Diese Anweisung bietet neben der Script-Anweisung eine zusätzliche Möglichkeit eigene Funktionen in das Framework aufzunehmen. Der Vorteil gegenüber der Script-Anweisung ist die feste Integration einer der Anweisung in das Framework. Dem Server muss in diesem Fall nicht mehr der Namen der JavaScript-Funktion, sondern die Bezeichnung der Anweisung bekannt sein. Dies unterstützt die Trennung der Client- und Server-Logik.

Der Custom-Anweisung können zusätzliche Parameter in Form von Schlüssel-Wert-Paaren angefügt werden. Dies geschieht über die Angabe von Property-Tags innerhalb des Tags *custom*. Die Property-Informationen können über das Action-Objekt innerhalb der Funktion einer Anweisung ausgelesen werden.

Das folgende Beispiel zeigt eine Custom-Anweisung vom Typ *changeColorAction.* Der Anweisung wurde eine Property-Information hinzugefügt, welche der Anweisung eine Farbangabe übergibt.

```
...
<body>
    <action>
        <custom type=“changeColorAction“>
            <property key=”color” value=”red” />
        </custom>
    </action>
</body>
...
```

Code-Snipped 14 Custom-Anweisung innerhalb einer AKP-Request-Nachricht

4.2. Besonderheiten des Frameworks

4.2.1. Der Template-Mechanismus

Ein Template-Mechanismus dient der Wiederverwendung von HTML-Bausteinen über die Definition von HTML-Vorlagen. Eine HTML-Vorlage besteht dabei aus der Grundstruktur eines HTML-Bausteins und mehreren Platzhaltern, die während des Render-Prozess mit konkreten Inhalten gefüllt werden. Der Template-Mechanismus des Frameworks ermöglicht das Übertragen von XML-Objekten an die Client-Logik der Webanwendung. Die Client-Logik ist für das Füllen der Templates mit den Inhalten des XML-Objekts zuständig.

Bei dem Template-Mechanismus des AKF unterliegt die Struktur eines übertragenen XML-Objekts bestimmten Vorgaben. Die Inhalte des XML-Objekts müssen innerhalb eines direkten Kind-Elementes des Wurzel-Elementes enthalten sein. Folgender Kode-Ausschnitt zeigt ein XML-Objekt, welches die genannten Vorgaben erfüllt. Das XML-Objekt enthält die Inhalte *id*, *name* und *imgUrlPath*.

```
<Article>
    <id>5A3G</id>
    <name>USB-Stick</name>
    <imgUrlPath>/img/usb.jpg</imgUrlPath>
</Article>
```

Code-Snipped 15 Artikel-XML-Objekt

Das folgende Template stellt eine mögliche Vorlage für das obige XML-Objekt in Form einer HTML-Tabellenzeile dar. Ein Platzhalter des Templates muss den exakten Namen eines der Kind-Elemente des XML-Objekts besitzen. Der Platzhalter wird mit einem $-Zeichen eingeleitet und abgeschlossen.

```
<try>
    <td>$id$</td>
    <td>$name$</td>
    <td>$imgUrlPath$</td>
</try>
```

Code-Snipped 16 Beispiel Template für das Artikel-XML-Objekt

Das zu Verwendende Template muss dabei über die Client-Logik ermittelt werden können. Ein Template kann über drei verschiedene Arten ermittelt werden.

Embedded-Template

Das Embedded-Template wird der Client-Logik des Frameworks mit dem Laden der Webseite übergeben. Da das Template auch aus unvollständigen HTML-Fragmenten bestehen kann, befindet sich ein Embedded-Template immer innerhalb eines HTML-Script-Tags. Als Typ des Skript-Tags muss *auf-template* angegeben werden. Dieser Typ unterscheidet das Script beispielsweise von JavaScript-Definitionen. Zusätzlich muss dem Script-Tag eine Id zugewiesen werden, über die das Template ermittelt werden kann. Diese Id darf innerhalb einer Webseite höchstens einmal verwendet werden. Der folgende Kode-Ausschnitt zeigt das obige Template in Form eines Embedded-Templates.

```
<script type=”act-template” id=”articleTemplate”>
    <try>
        <td>$id$</td>
        <td>$name$</td>
        <td>$imgUrlPath$</td>
    </try>
</script>
```

Code-Snipped 17 Beispiel eines Embedded-Template

Request-Template

Die zweite Möglichkeit der Client-Logik eine Template-Vorlage zu übermitteln ist die Übertragung des Templates innerhalb einer AKP-Request-Nachricht. Für diesen Zweck kann der Body-Teil einer AKP-Request-Nachricht neben den Anweisungen auch Templates innerhalb eines *template*-Tags enthalten. Nicht XML-konforme Templates müssen auch hier in einen CDATA-Abschnitt eingebettet werden. Über diesen Ansatz kann eine Addcontent-Anweisung mit dem Render-Typ template und das

dazugehörige Template innerhalb einer Response-Nachricht übertragen werden. Der folgende Ausschnitt aus einer AKP-Request-Nachricht zeigt die Übertragung des XML-Artikel-Objekts innerhalb einer Addcontent-Anweisung und dem zugehörigen Template.

```
...
<body>
  <action>
    <addcontent containerid=”divContainer” rendertype=”template”
        templateid=”articleTemplate” />
      <Article>
        <id>5A3G</id>
        <name>USB-Stick</name>
        <imgUrlPath>/img/usb.jpg</imgUrlPath>
      </Arrticle>
    </addcontent>
  </action>
  <template id=”articleTemplate”>
    <tr>
      <td>$id$</td>
      <td>$name$</td>
      <td>$imgUrlPath$</td>
    </tr>
  </template>
</body>
...
```

Code-Snipped 18 Beispiel eines Request-Templates

Remote Template

Falls die Client-Logik das Template nicht innerhalb des HTML-Markups in Form eines Embedded-Templates oder in Form eines Request-Templates innerhalb der Serverantwort ermitteln kann, versucht die Client-Logik das Template vom Server anzufragen. Die URL über die ein Template angefragt werden soll, kann auf der Seite des Clients konfiguriert werden. Über diese Konfiguration kann der Remote-Template-Mechanismus der Frameworks an den anwendungsspezifischen URL-Aufbau angepasst werden. Das Framework spezifiziert nicht, wie das Template auf der Seite des Servers geladen wird. Falls das Template nicht über den Remote-Mechanismus geladen werden kann, wird die Verarbeitung der Anfrage mit Hilfe eines Fehlers abgebrochen. Die Serverantwort der Anfrage eines Remote-Templates enthält eine AKP-Request-Nachricht mit dem eingebetteten Template.

Template-Caching

Um die Verarbeitung der Templates zu beschleunigen, werden Templates auf der Seite des Clients automatisch zwischengespeichert. Das Caching von Templates spart zum einen Zeit, falls die Templates einer Webanwendung über den Remote-Template-Mechanismus geladen werden. Des Weiteren kann das Netzwerk entlastet werden, da Remote-Templates nur einmalig von der Client-Logik geladen werden muss. Über das Template-Caching kann außerdem der Overhead verringert werden, der für die Aufbereitung der Templates anfällt.

4.2.2. Das Rendering von Komponenten

Das Framework stellt einen Komponenten-Mechanismus zur Verfügung, über den bestimmte Teile einer Webseite generiert werden können, ohne den Rest einer Webseite generieren zu müssen. Als Komponente, wird in diesem Fall, ein Bestandteil einer Webseite bezeichnet, die mittels einer AKP-Anfrage vom Server abgefragt werden kann. Der Komponenten-Mechanismus des AKF ist nicht mit dem Komponentenmodell eines komponentenbasierten Webframeworks vergleichbar, da die Komponenten im Fall von AKF nicht auf dem Server gespeichert werden. Eine AKF-Komponente ist ein Teil einer Webseite, der unabhängig von anderen Webseiten-Teilen über eine AKP-Anfrage abgefragt werden kann. Dieser Mechanismus spart die Rechenleistung, die für die Generierung der anderen Webseitenteile benötigt werden würde.

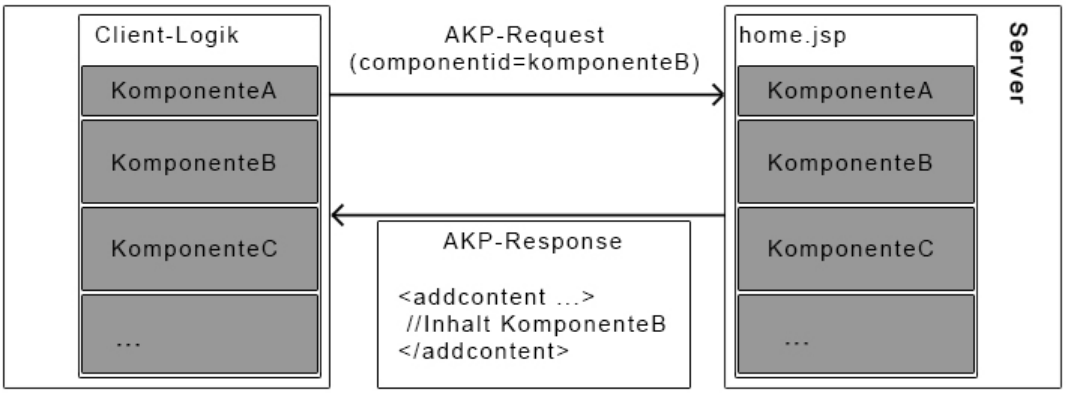

Abbildung 20 Darstellung des Komponenten-Mechanismus des AKF

Um das Rendering von einzelnen Komponenten auf der Seite des Servers zu erzwingen kann einer AKP-Anfrage der Anfrageparameter *componentid* mitgegeben werden. Die Server-Logik ist bei der Verarbeitung der Anfrage dafür verantwortlich die Komponenten mit der angegebenen Komponenten-Id zu rendern und als Addcontent-Anweisung in die Serverantwort zu integrieren.

4.2.3. Umgang mit nicht JavaScript-fähigen Clients

Da die Bedienung einer Webanwendung, vor allem im Bereich E-Commerce, über jeden Webclient möglich sein soll, sieht das AKF einen Mechanismus für den Umgang mit nicht JavaScript-fähigen Clients vor. Über diesen Mechanismus können Bereiche einer Webseite definiert werden, die nur für JavaScript-fähige Clients sichtbar sind. Die Entwickler einer Webseite haben die Möglichkeit alternative Inhalte für Webclients zu definieren, die JavaScript nicht unterstützen oder deaktiviert haben.

Die Teile einer Webseite, welche nur für JavaScript-fähige Clients sichtbar sein sollen, müssen in ein HTML-Element mit der Klasse „akfNSAC“ eingebettet sein. Alternative Inhalte, die für nicht JavaScript-fähige Clients sichtbar sein sollen, müssen in ein HTML-Element mit der Klasse „akfSAC“ eingebettet werden. Der Name dieser Element-Klasse kann über das Framework konfiguriert werden.

Folgendes Beispiel zeigt einen HTML-Ausschnitt mit einem definierten Inhalt für JavaScript-fähige Clients und einem alternativen Inhalt für nicht JavaScript-fähige Clients. Der Inhalt besteht aus einem Link, der die Tabelle eines Warenkorbs aktualisieren soll. Der Inhalt des ersten Div-Elements ermöglicht die Aktualisierung des Warenkorbs über eine JavaScript-Funktion. Der alternative Inhalt ermittelt die gesamte Webseite mit dem Warenkorb über einen konventionellen Hyperlink. Die Client-Logik des Frameworks ist dafür verantwortlich den alternativen Inhalt unsichtbar zu machen und den JavaScript-fähigen Inhalt über das setzen des *display*-Attributs sichtbar zu machen, falls der Webclient JavaScript aktiviert hat.

```
...
<div class=”akfSAC” style=”display:none”>
    <a href=“#“ onclick=”refreshCart();return false;”>
        Aktualisieren
    </a>
</div>
<div class=”akfNSAC”>
    <a href=”cart.jsp”>
        Aktualisieren
    </a>
</div>
...
```

Code-Snipped 19 Definition eines JavaScript-fähigen Inhalts und eines alternativen Inhalts

5. Details der Implementierung

Dieses Kapitel gibt einen Einblick in die clientseitige sowie serverseitige Implementierung des Frameworks. Die Funktionen der Server-Logik werden innerhalb der folgenden Kapitel anhand der View-Technologie JSP/JSPX erläutert.

5.1. Beschreibung der Entwicklungsumgebung

Für die Implementierung des Frameworks wurde die Entwicklungsumgebung Eclipse Helios mit der Webtool-Plattform (WTP) verwendet. Die Eclipse Webtool Plattform enthält erweiterte Funktionen für die Entwicklung von Java-EE Webanwendungen. Das WTP-Plugin erlaubt beispielsweise die Kode-Vervollständigung innerhalb von JSP-Seiten oder JavaScript-Dokumenten. Zusätzlich zu dieser Erweiterung wurde eine SVN-Erweiterung installiert, welches die Synchronisation der Projektdateien mit einem externen Server über die Eclipse-Oberfläche ermöglicht.

Für den Erstellungsprozess (Build-Prozess) des Projekts wurde das Build-Werkzeug Apache-Ant gewählt. Apache-Ant ist ein auf Java basierendes Build-Werkzeug, mit dem beispielsweise der Quellkode eines Projekts kompiliert, archiviert und kopiert werden kann.

Um die einzelnen Funktionen des Frameworks testen zu können wurde der Webserver Apache-Tomcat installiert. Apache-Tomcat ist ein leichtgewichtiger Webserver für die Ausführung Java-basierter Webanwendungen.

Zusätzlich wurde ein Web-Projekt innerhalb von Eclipse aufgesetzt, welches die Dateien des Frameworks und einer Testanwendung enthält.

Grundstruktur des Projekts

Abbildung 21 zeigt die Grundstruktur des gesamten Projekts in Eclipse. Die folgenden Abschnitte erläutern die wichtigsten Ordner des Projekts.

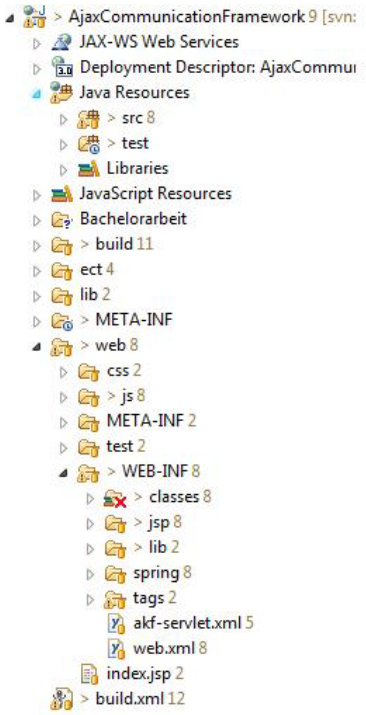

Abbildung 21 Projektstruktur

src:	Dieser Ordner enthält die Quell-Dateien der des Frameworks
test	Dieser Ordner enthält die Quell-Dateien der Testanwendung
lib:	Der lib-Ordner enthält alle benötigten Java-Bibliotheken, die für das Kompilieren der Quelldateien benötigt werden.
META-INF	Das META-INF Verzeichnis enthält den Taglibrary-Deskriptor, der die Tags des Frameworks beschreibt
css:	Dieser Ordner enthält Style-Sheet Dokumente
js:	Dieser Ordner enthält JavaScript-Dokumente
jsp:	Dieser Ordner enthält alle JSP- sowie JSPX-Dokumente
WEB-INF:	Der WEB-INF-Ordner enthält innerhalb einer Java-EE-Webanwendung alle Ressourcen, auf die nicht direkt über das Web Zugegriffen werden kann.
spring	Dieser Ordner enthält Konfigurationsdateien des Spring-Frameworks, welches für die Testanwendung eingesetzt wird.

Tabelle 7 Wichtige Ordner der Projektstruktur

Testen der Anwendung (Nach der Implementierung)

Die AKF-Konsole

Für die Entwicklungs- bzw. Testphase einer Webanwendung stellt das AKF eine eigene Konsole bereit. Über die Funktion *akf.console.log()* können Log-Mitteilungen auf der Konsole ausgegeben werden. Über die Konfiguration des Wertes *akf.console.selector* kann der Selektor des HTML-Elements der Konsole konfiguriert werden. Die Standardeinstellung des Selektors ist „*akfConsole*“. Neben der Ausgabe der Log-Mitteilungen auf der Webseite werden die Nachrichten zusätzlich auf der Browserinternen Konsole über den Befehl *console.log()* mit geloggt, falls diese Funktion auf dem Browser zur Verfügung steht. Der Inhalt der Konsole kann über den Befehl *akf.console.clear()* wieder entfernt werden. Über den Konfigurationseintrag *akf.debugmode* kann das Logging aktiviert bzw. deaktiviert werden.

Implementierung von Testfällen

Die Testanwendung dient dem Testen der einzelnen Funktionen des Frameworks. Hierzu wurde eine Testanwendung auf der Basis des Spring-Frameworks implementiert. Das Spring-Web-Framework bietet erweiterte Funktionen für die Implementierung von Java-basierten Webanwendungen. Die Testanwendung besteht aus einer Datei index.jsp, einem einfachen Spring-Controller und einer Style-Sheet-Datei. Die index.jsp stellt die grafische Oberfläche der Testanwendung dar. Der Controller empfängt die Serveranfragen der Testanwendung und lässt je nach Test-Anfrage eine bestimmte JSPX-Datei mit einer AKP-Response-Nachricht generieren. Die folgende Auflistung erläutert die wichtigsten Dateien der Testanwendung.

build.xml	Dies ist die Ant-Build-Datei. Über diese Datei kann die Testanwendung kompiliert und auf dem Tomcat-Server installiert werden.
index.jsp	Die index.jsp stellt die grafische Oberfläche der Testanwendung bereit.
web.xml	Die web.xml ist die Konfigurationsdatei der gesamten Webanwendung
akf-servlet.xml	Diese Datei enthält die Spring-Konfiguration

Tabelle 8 Wichtige Dateien der Testanwendung

Der Ablauf eines Tests wird nun anhand eines Beispiels verdeutlicht. Abbildung 22 zeigt einen Ausschnitt aus der Testanwendung. Innerhalb dieses Tests wird die Template-Funktion des Frameworks getestet. Nach einem Click auf den Link „Add new Row“ wird ein asynchroner AKP-Request an den Server gesendet. Dieser Request wird im Anschluss von einem Test-Controller empfangen. Der Controller interpretiert den Request und generiert den AKP-Response über eine JSPX-Datei. Die JSPX-Datei enthält dabei den XML-Inhalt der Response-Nachricht. Alle Ereignisse wie das Senden der Nachricht oder Fehlermeldungen werden dabei auf der Konsole ausgegeben.

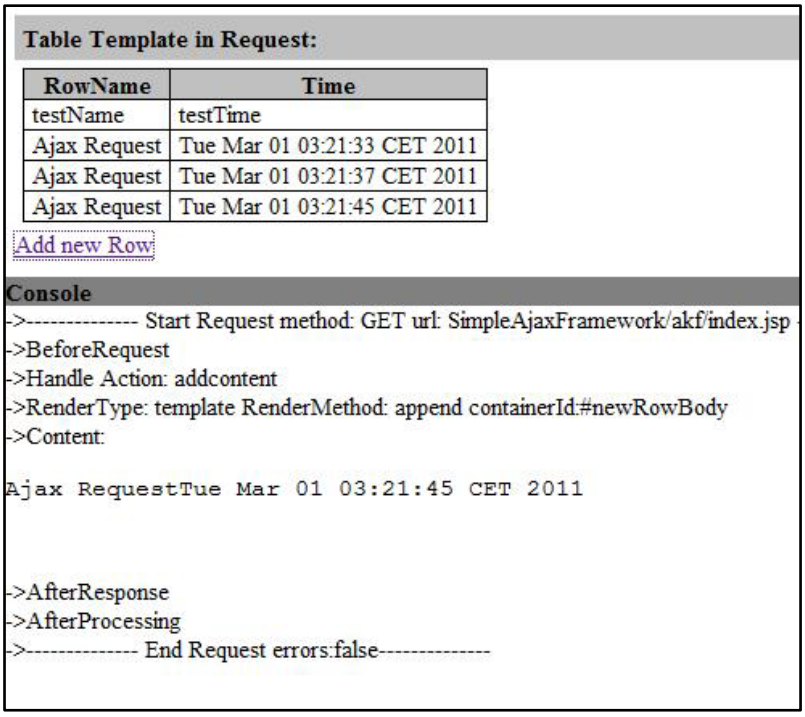

Abbildung 22 Ein einfaches Testszenario

5.2. Grundlagen der Client-Logik

Die Client-Logik des Ajax-Kommunikations-Framework wurde mit Hilfe des Javascript-Frameworks JQuery implementiert. Das JQuery-Framework bietet unter Anderem hilfreiche Funktionen für die Manipulation von XML und HTML Dokumenten. Des Weiteren wurde die Ajax-Funktionalität des JQuery-Frameworks für die Client/Server-Kommunikation des AKF verwendet. Im weiteren Verlauf dieser Arbeit, werden die Wichtigsten Funktionen und Objekte der Client-Logik vorgestellt.

5.2.1. Das JavaScript-Objektmodell

Als Grundlage der clientseitigen Logik wurde ein JavaScript-Objektmodell entwickelt, welches die einzelnen Objekte und Funktionen der Client-Logik bereitstellt. Ein JavaScript-Objekt besteht aus verschachtelten Attributen und Funktionen. Die Attribute und Funktionen besitzen dabei eindeutige Namen und einen zugehörigen Wert. Da die Skriptsprache JavaScript nicht typsicher ist, können einem Attribut verschiedene Typen zugewiesen werden.

Folgender Ausschnitt aus der Client-Logik des Frameworks verdeutlicht die Struktur eines verschachtelten Objektes in JavaScript. Der Ausschnitt zeigt ein übergeordnetes Objekt mit dem Namen akf. Die Definition eines übergeordneten Objekts, welches die Funktionalität des Frameworks enthält, dient unter Anderem der Unterscheidung von Funktionen und Attributen des Frameworks zu denen eines anderen Javascript-Frameworks. Das untergeordnete Objekt *state* enthält die einzelnen Zustände eines AKF-Request. Der Wert des Zustands *beforeRequest* kann beispielsweise mit Hilfe des Ausdrucks *akf.state.beforeRequest* überprüft werden. Das Beispiel zeigt zudem die Definition einer

Methode *addAction*. Die Methode *addAction* wird für das Hinzufügen von Custom-Anweisungen verwendet und besitzt zwei Argumente. Mit Hilfe des *this*-Zeigers können die Funktionen und Attribute des umgebenen Objekts aufgerufen werden. Im Fall der *addAction*-Funktion wird dem Array *customAction* ein neues Action-Objekt angefügt, welches aus den Attributen *type* und *handler* besteht.

```
var akf= {
  state: {
    beforeRequest:0,
    afterResponse:1,
    requestAbort:3,
    requestError:4,
    appError:5,
    afterProcessing:6
  },
  ...
  customAction:new Array(),
  addAction: function(actionType, actionHandler) {
    this.customActions.push({type:actionType, handler:actionHandler});
  }
  ...
}
```

Tabelle 9 Ausschnitt aus dem akf-Objektmodell

5.2.2. Die Konfiguration des akf-Objektes

Bestimmte Einstellungen des akf-Objektes können über die Client-Logik konfiguriert werden. Dies kann beispielsweise bei der Integration des Frameworks hilfreich sein. Tabelle 10 verschafft einen Überblick über die einzelnen Konfigurationseinträge des akf-Objektes.

Konfiguration	Beschreibung
akf.debugMode	Wenn der Konfigurationseintrag *akf.debugMode* auf *true* gesetzt können Debug-Ausgaben auf der Konsole ausgegeben werden.
akf.templatePath	Über diese Konfiguration kann der relative Pfad definiert werden, über den die Client-Logik Remote-Templates vom Server anfragen kann.
akf.scriptCheck.noScriptSelector	Dieser Konfigurationseintrag legt die Klasse der HTML-Elemente fest, welche für nicht Javascript-fähige Clients sichtbar sind.
akf.scriptCheck.activeScriptSelector	Dieser Konfigurationseintrag legt die Klasse der HTML-Elemente fest, welche nur für Javascript-fähige Clients sichtbar sein sollen.
akf.message	Über das Message-Objekt können bestimmte Nachrichten wie Debug-Ausgaben oder Fehlermeldungen definiert werden.

Tabelle 11 Konfigurationseinträge des AKF

5.2.3. Die Client/Server-Kommunikation

Die Client-Logik bietet vier verschiedene Funktionen für eine asynchrone Kommunikation mit dem Server.

Schnittstelle für die Client/Server-Kommunikation

Über die Funktion *akf.asyncGet* wird ein AKP-GET-Request an den Server gesendet und die Serverantwort von der Client-Logik verarbeitet. Die Verarbeitung des AKP-Request-Lebenszyklus verläuft bei Verwendung der Funktion *akf.asyncGet* nebenläufig. Mit Hilfe einer nebenläufigen Ausführung, kann die Client-Logik nach dem Absenden der AKP-Request-Nachricht weitere Funktionen ausführen, ohne auf die Verarbeitung der Serverantwort warten zu müssen. Die Funktion akf.asyncPost kann dazu verwendet werden eine nebenläufige AKP-POST-Anfrage an den Server zu senden.

Die Funktionen *akf-syncGet* und *akf.syncPost* werden nicht nebenläufig ausgeführt. Die Ausführung dieser beiden Funktionen unterbricht die Anweisungskette der Client-Logik, bis die Antwort der Serveranfrage verarbeitet wurde.

Folgende Tabelle beschreibt die Argumente der vorgestellten Funktionen der Client-Logik.

Argument	Beschreibung
url	Dieses Argument definiert die URL des AKP-Requests
requestparameters	Über dieses Argument kann dem AKP-Request zusätzliche Request-Parameter in Form von Javascript Schlüssel/Wert-Paaren mitgegeben werden.
caller	Das Caller-Objekt ist das HTML-Element, über das der Request ausgeführt wurde.
stateHandler	Dieses Argument definiert den Namen einer Handler-Funktion, welche bei einem Zustandswechsel des Request-Kontexts aufgerufen wird.

Tabelle 12 Argumente für die Request-Methoden

Verarbeitung einer Serveranfrage

Abbildung 23 zeigt die Verarbeitung eines erfolgreichen AKP-GET-Requests anhand eines Sequenzdiagramms. Intern delegieren die vier vorgestellten Request-Funktionen die Ausführung der Anfragen an die Funktion *akf.loadAjaxResponse.* Diese Funktion initialisiert den Request-Kontext über die Funktion *akf.createInitContext* und sendet die Anfrage über die JQuery-Funktion *jQuery.ajax* an den Server.

Zunächst wird der *beforeRequest*-Handler der Funktion *jQuery.ajax* aufgerufen. Dieser Handler setzt den AKP-Requests auf den Zustand *beforeRequest*. Bei jeder Zustandsänderung wird der Statehandler benachrichtigt, der dem Request zugewiesen wurde.

Als nächstes wird der *dataFilter*-Handler der jQuery.ajax Funktion aufgerufen, falls der Request erfolgreich an den Server gesendet und eine Serverantwort empfangen wurde. Dieser Handler dient dem Aufbau des AjaxResponse-Objekts, welches im weiteren Verlauf dieses Kapitels näher beschrieben wird.

Nach dem Aufbau des Response-Objekts wird der Success-Handler der *jQuery.ajax* Funktion aufgerufen. Der Success-Handler setzt den Zustand des AKP-Requests auf *afterResponse* und führt die Anweisungen der AKP-Response-Nachricht über die Funktion akf.handleContextActions aus.

Nach der Verarbeitung der Serverantwort wird der Complete-Handler der jQuery.ajax Funktion ausgeführt, welcher den *afterProcessing*-Zustand der Anfrage setzt.

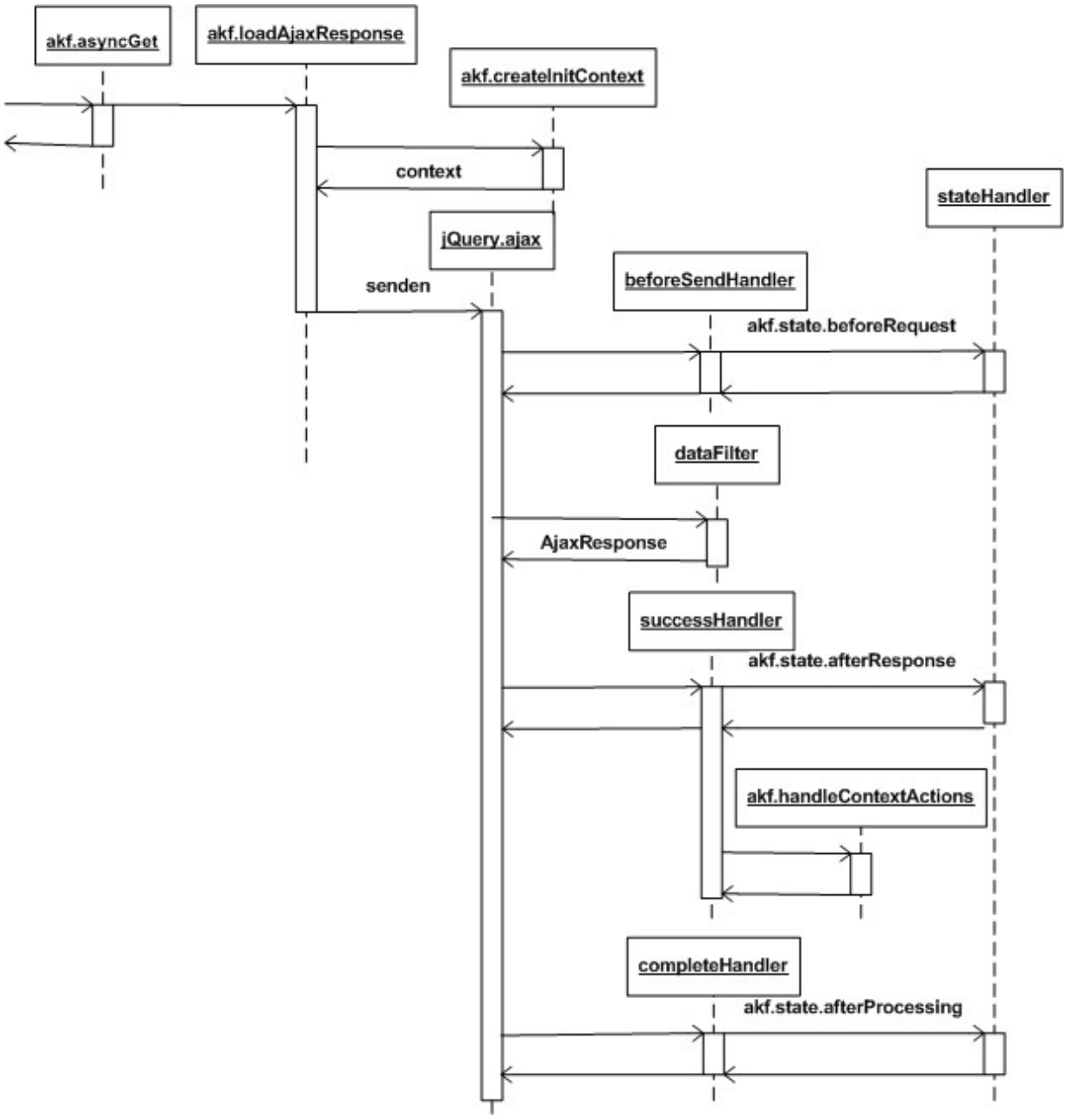

Abbildung 23 Sequenzdiagramm der Verarbeitung einer Serveranfrage

5.3. Grundlagen der Server-Logik

Die Server-Logik des AKF vereinfacht die Formulierung von AKP-Response-Nachrichten und implementiert zusätzlich Funktionen, wie das Rendering von Komponenten und dem Template-Mechanismus. Folgende Abschnitte erläutern den Aufbau der Server-Logik des AKF.

5.3.1. Das Ajax-Response-Objektmodell

Die Server-Logik bietet dem Entwickler einer AKF-Anwendung ein Objektmodell für die Definition einer AKP-Response-Nachricht. Über dieses Objektmodell kann eine AKP-Response-Nachricht in Form eines Java-Objektes erstellt und bei Bedarf in das XML-Format umgewandelt werden.

Abbildung 24 verdeutlicht den Aufbau und die Beziehungen der einzelnen Klassen des Objektmodells. Das Objektmodell bildet die Struktur einer AKP-Response-Nachricht ab. Die Klasse AjaxResponseMarshaller ist für die Umwandlung des Objektes in das XML-Format zuständig.

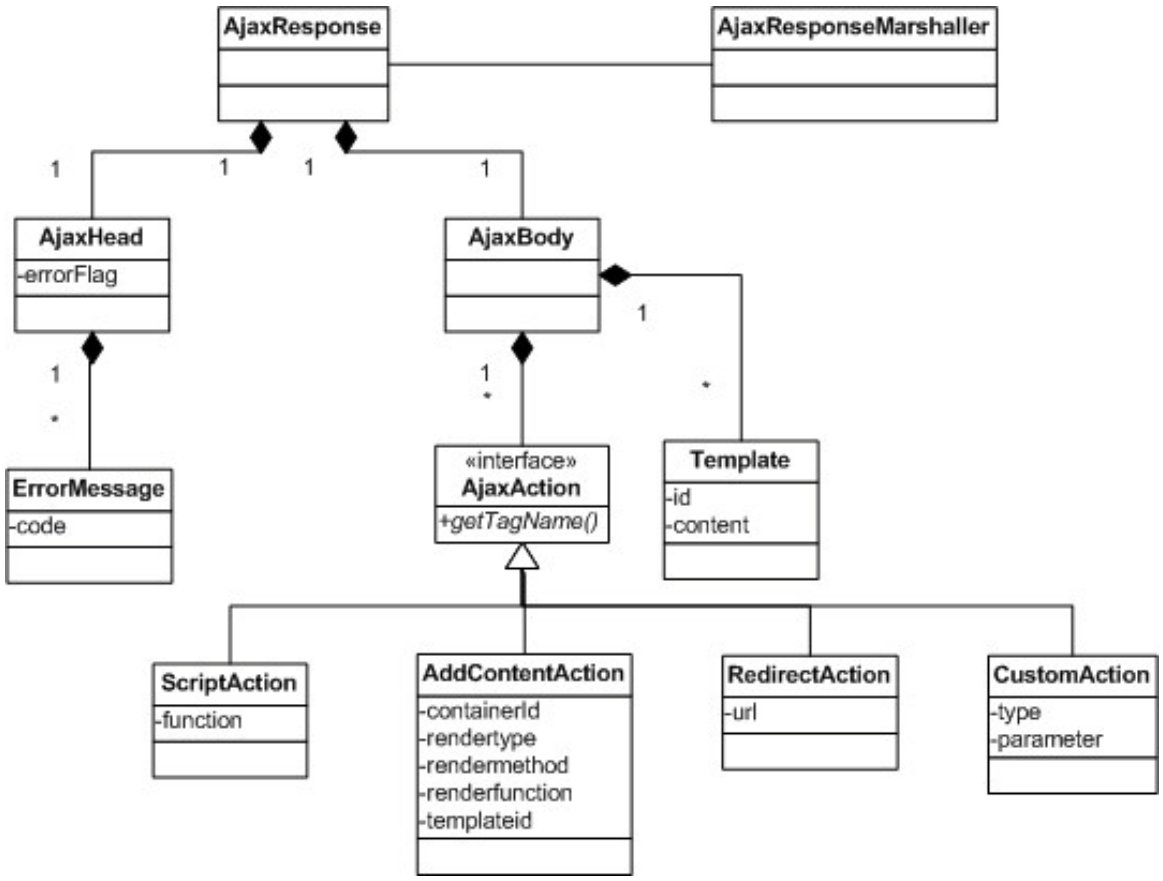

Abbildung 24 Klassendiagramm des Ajax-Response-Objektmodells

5.3.2. Die Ajax-Response-Tags

Neben der Formulierung einer Response-Nachricht mit Hilfe von einfachen Objekten, bietet das AKF zusätzlich die Möglichkeit eine AKF-Response-Nachricht über Tag-Klassen zu erstellen. Tag-Klassen können innerhalb von Webanwendungen dazu verwendet werden, die Logik einer Webanwendung von der Darstellung zu trennen. Ein Tag besteht dabei aus einer Tag-Handler-Klasse, welche innerhalb eines Tag-Library-Deskriptors (TLD) registriert wird. Der TLD enthält den Namen, die Klasse und die Attribute der bereitgestellten Tags. Das AKF bietet verschiedene Tags um eine AKP-Response-Nachricht innerhalb von View-Technologien wie JSPX oder Facelets zu erstellen. Diese Tags dienen

zudem der Kode-Vervollständigung innerhalb der Entwicklungsumgebung. Abbildung 25 zeigt ein Klassendiagramm mit allen AKF-Response-Tags.

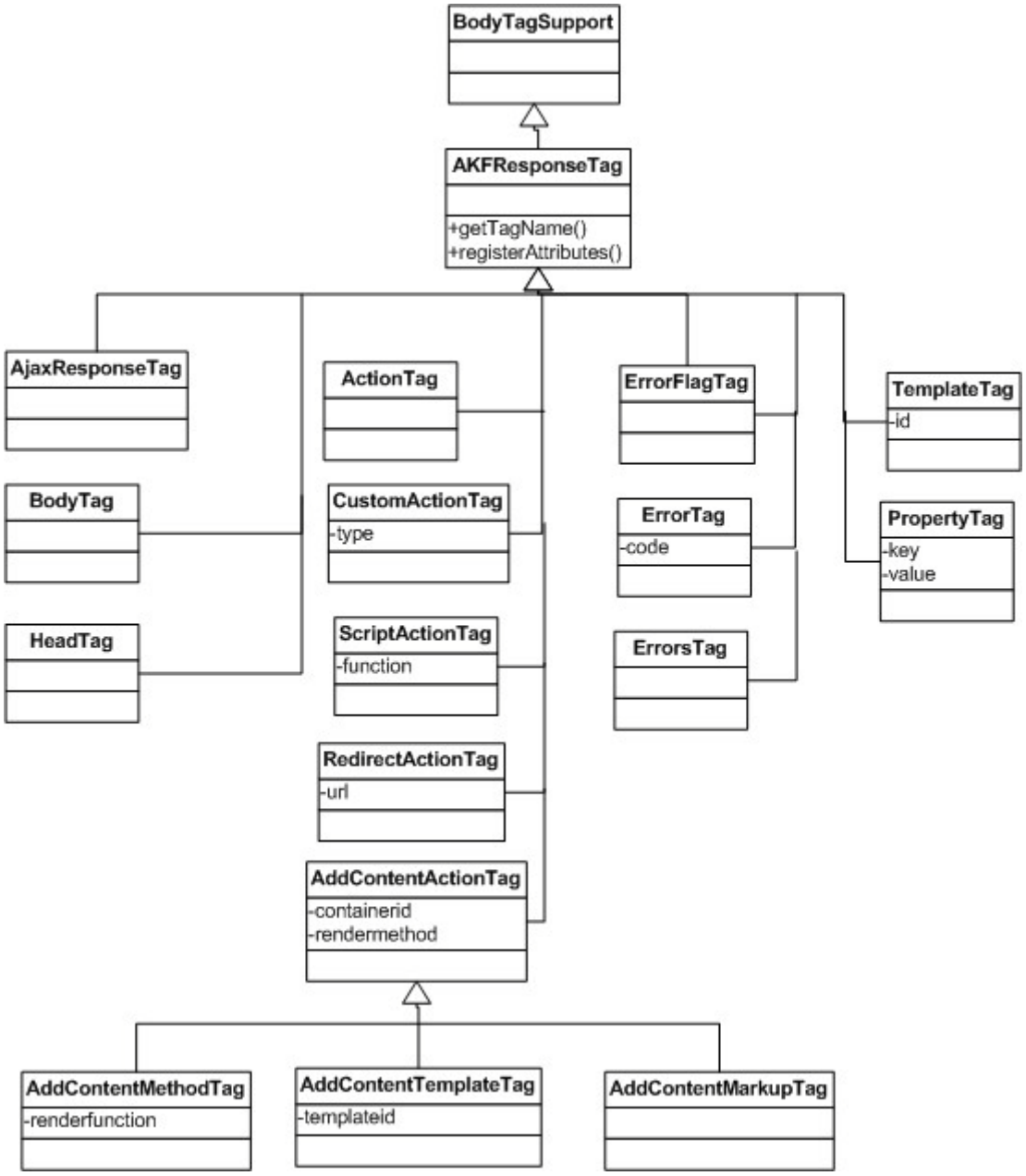

Abbildung 25 Klassendiagramm für die AKP-Response-Tags

Die einzelnen Tag-Klassen können beispielsweise innerhalb einer JSPX-Datei verschachtelt werden, um die Struktur einer AKP-Response-Nachricht zu abzubilden. Die Aufgabe der Tag-Klassen ist die Generierung einer AKP-Response-Nachricht in Form von XML. Folgendes Beispiel zeigt den Inhalt einer einfachen JSPX-Datei, über die eine AKP-Response-Nachricht generiert werden kann. Das Ergebnis des Render-Prozesses dieser JSPX-Datei ist eine AKP-konforme Response-Nachricht.

```
<jsp:root xmlns:jsp="..." xmlns:akf="...">
  <akf:AjaxResponse>
    <akf:ResponseHead>
      <akf:ErrorFlag>false</akf:ErrorFlag>
    </akf:ResponseHead>
    <akf:ResponseBody>
      <akf:ResponseAction>
        <akf:ScriptAction function="doFunction" />
      </akf:ResponseAction>
    </akf:ResponseBody>
  </akf:AjaxResponse>
</jsp:root>
```

Code-Snipped 20 Eine JSPX-Datei mit der Definition einer AKP-Response-Nachricht

5.4. Implementierung des Template-Mechanismus

5.4.1. Die Server-Logik

Der Template-Mechanismus wurde serverseitig über die Verwendung von Tag-Handler-Klassen gelöst. Ein Template kann innerhalb einer JSP-Datei über das Tag *AjaxTemplate* und dem Attribut *id* eingeleitet werden. Innerhalb des *AjaxTemplate*-Tags wird der Inhalt des Templates definiert. Die Platzhalter eines Templates werden über das Tag *TemplateElement* in das Template eingebunden. Code-Snipped 21 AjaxTemplate Beispiel zeigt die Definition eines Templates innerhalb eines JSP-Dokuments.

Über die Tags *AjaxTemplate* und *TemplateElement* kann der Inhalt einer Webseite generiert werden, wobei einzelne Teile der Webseite gleichzeitig als Template in die Webseite integriert werden. Der Inhalt des *AjaxTemplate*-Tags wird dabei zweimal durchlaufen. Beim ersten Durchlauf werden die Platzhalter, die über das Tag *TemplateElement* eingeleitet werden, mit dem Wert des Attributs *tmplExrpession* gefüllt und als Template auf der Webseite registriert. Das *tmplExpression*-Attribut muss dabei den Namen des Template-Platzhalters besitzen. Der zweite Durchlauf generiert den normalen Inhalt des *TemplateElement*-Tags und fügt diesen in die Webseite ein. Über diese Technik kann beispielsweise eine Tabelle generiert werden, deren Zeilenstruktur ebenfalls als Template bereitgestellt wird.

```
<table>
  <thead>
    <tr><td>ID</td></tr>
  </thead>
  <tbody id=”templateBody”>
    <akf:AjaxTemplate id=”tableTemplate”>
      <tr>
        <td>
          <akf:TemplateElement tmplExpression=”articleId”>
            $(article.id)
          </akf:TemplateElement>
        </td>
      </tr>
    </akf:AjaxTemplate>
  </tbody>
</table>
```

Code-Snipped 21 AjaxTemplate Beispiel

Die Registrierung der Templates einer Webseite geschieht über das AjaxPage-Tag. Das AjaxPage-Tag umgibt das gesamte JSP-Dokument einer AKF-Webseite. Während der Generierung des JSP-Dokuments werden die Templates der Webseite innerhalb der Tag-Handler-Klasse des *AjaxPage*-Tags gesammelt und müssen am Ende der Webseite über die Angabe eines *PrintTemplate*-Tags in die Webseite integriert werden. Das PrintTemplate Tag ist dafür Verantwortlich die Templates mit Hilfe von HTML-Script-Elementen in die Webseite einzufügen.

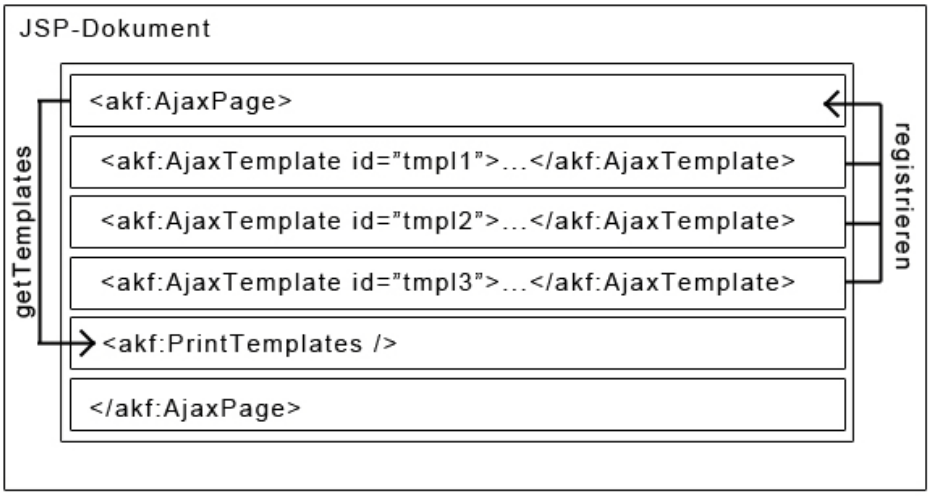

Abbildung 26 Aufbau einer JSP-Datei mit einem AjaxTemplate

5.4.2. Die Client-Logik

Das Template-Objekt

Innerhalb der Client-Logik werden Templates in Form von JavaScript-Objekten gehalten. Ein Template-Objekt besitzt dabei die Template-ID und den unbearbeiteten Template-Text. Zusätzlich verfügt das Template-Objekt über eine render-Funktion. Die render-Funktion dient der Generierung der Inhalte und erwartet ein XML-Objekt als Argument. Während des render-Prozesses werden die Inhalte des XML-Objekts in das Template eingefügt und das Ergebnis zurückgegeben.

Ermittlung von Templates

Über die Funktion *akf.getTemplate* kann eine Template-Instanz ermittelt werden. Die Funktion *akf.getTemplate* sucht das Template zunächst im aktuellen Request-Kontext und im Template-Cache. Falls das Template nicht ermittelt werden konnte, wird die Webseite nach Embedded-Templates durchsucht. Über die Funktion *akf.loadRemoteTemplate* können zusätzlich Remote-Templates vom Server angefragt werden.

5.5. Implementierung des Komponenten-Renderings

Das Rendering von Komponenten wurde serverseitig ebenfalls mit Hilfe von Tag-Handler-Klassen gelöst. AKF-Komponenten können über das Tag *AjaxComponent* eingeleitet werden. Eine AKF-Komponente besitzt eine eindeutige ID und wird ähnlich wie ein Template innerhalb des AjaxPage-Tags registriert. Die Bestandteile einer Webseite, die nicht eigenständig über einen Ajax-Aufruf abgefragt werden, können innerhalb eines *NonAjaxComponent*-Tags eingebettet werden.

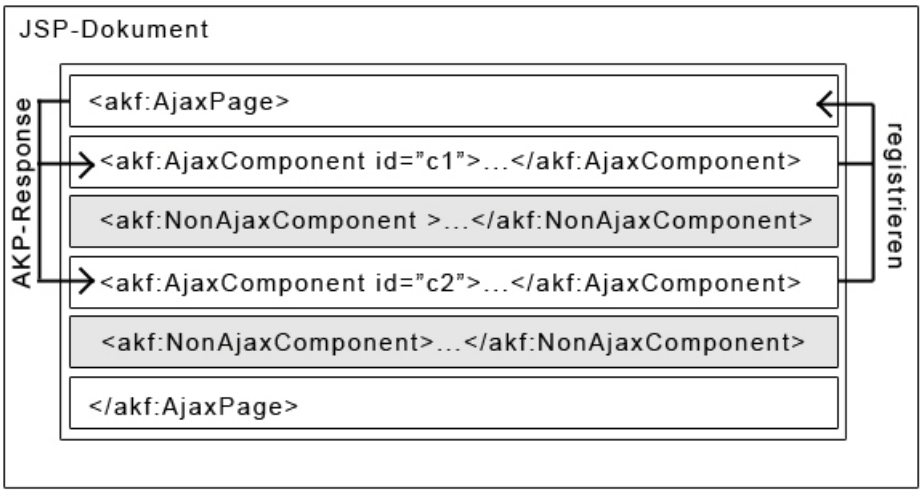

Abbildung 27 Rendering einer JSP-Datei mit AKF-Komponenten

Falls das AjaxPage-Tag den Request als AKP-Request identifiziert und der Request einen Parameter componenid.besitzt, werden nur die Inhalte der AjaxComponent-Tags generiert und über das AjaxPage-Tag registriert. In diesem Fall generiert das AjaxPage-Tag keine HTML-Seite, sondern eine

AKP-Response-Nachricht mit den angefragten AjaxComponenten. Somit wird die Generierung der NonAjaxComponenten umgangen.

5.6. Generierung von alternativen Inhalten für nicht JavaScript-fähige Clients

Um die Definition von alternativen Inhalten, für nicht Javascript-fähige Clients zu ermöglichen, wurde die JQuery-Funktion *$(document).ready* verwendet. Diese Funktion wird vom JQuery-Framework ausgeführt, falls alle Elemente einer Webseite geladen und dargestellt wurden. Innerhalb dieser Funktion kann auf alle Elemente einer Webseite zugegriffen werden.

Die Client-Logik des AKF verwendet diese Funktion um alle HTML-Elemente sichtbar zu machen, welche die eine bestimmte Klasse besitzen. Die Klasse für nicht JavaScript-fähige HTML-Elemente kann über den Konfigurationseintrag *akf.scriptCheck.activeScriptSelector* festgelegt werden. Des Weiteren werden alle HTML-Elemente entfernt, welche die Klasse des Konfigurationseintrages *akf.noScriptSelector* besitzen. Über diese Technik bleiben die Javascript-Elemente für nicht Javascript-fähige Clients verborgen, da in diesem Fall die Funktion *$(document).ready* nicht ausgeführt wird. Abbildung 28 verdeutlicht die Funktionsweise der *$(document).ready* Funktion im Zusammenhang mit nicht Javascript-fähigen Clients.

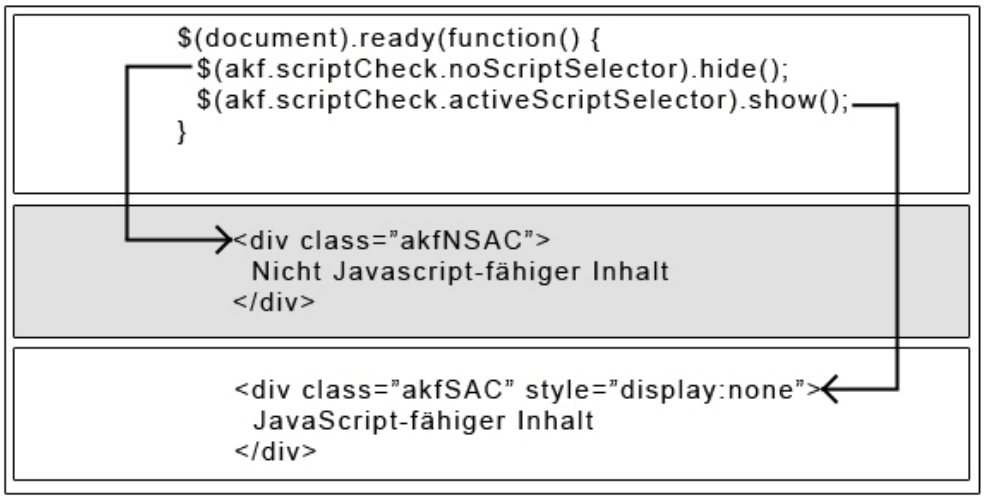

Abbildung 28 Funktionsweise der jQuery.ready Funktion im Zusammenhang mit nicht JavaScript-fähigen Clients

6. Integration des Frameworks

Dieses Kapitel beschreibt die Integration des Frameworks und die Umsetzung eines Widgets innerhalb einer E-Commerce-Anwendung. Die Anwendung in diesem Szenario stellt einen Web-Shop dar, der auf der Java Enterprise Edition und Spring basiert. Als View-Technologie der Webanwendung werden JSP und JSPX-Dokumente verwendet. Zudem verwendet die Anwendung das JQuery-JavaScript-Framework.

6.1. Beschreibung des Widgets

Das Widget, welches mit Hilfe des Frameworks umgesetzt werden soll, dient der Schnelleingabe von Artikeln in den Warenkorb. Ein Benutzer der Webanwendung soll über die Angabe einer Artikelnummer und der Menge, neue Artikel in den Warenkorb einfügen können. Das Layout und die Funktion für das Einfügen von Artikeln in den Warenkorb sind bereits im Shop enthalten.

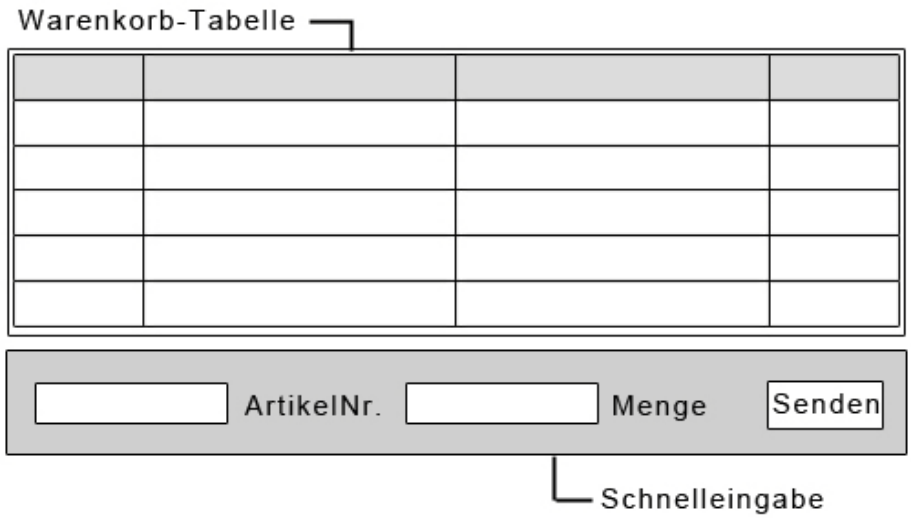

Abbildung 29 Layout des Widgets

Abbildung 29 zeigt das Layout der Seitenkomponente. Die Seitenkomponente setzt sich aus einer Tabelle mit Artikeldaten und der Schnelleingabemaske zusammen. Über den Link „Senden“ soll die Artikelnummer und die Mengenangabe an den Server übertragen werden und der zugehörige Artikel in den Warenkorb eingefügt werden. Falls der Artikel in den Warenkorb eingefügt werden konnte muss die Warenkorb-Tabelle von der Client-Logik aktualisiert werden. Falls der Artikel nicht eingefügt werden konnte muss eine entsprechende Fehlermeldung in die Webseite integriert werden.

6.2. Umsetzung des Widgets

Dieser Abschnitt erläutert die nötigen Schritte der Integration und der Umsetzung des vorgestellten Widgets.

Um das Framework in das vorhandene Projekt einzubinden, müssen zunächst die Quelldateien des Frameworks in Form einer Jar-Datei und einer JavaScript-Datei in die Projektstruktur kopiert werden.

Das Widget wurde im nächsten Schritt über den Komponenten-Mechanismus des Frameworks umgesetzt. Der Komponenten-Mechanismus ermöglicht dabei die Abfrage der gesamten Webseite, und die Abfrage des Inhaltes der Warenkorb-Tabelle. Des Weiteren sind über den Komponenten-Mechanismus nur geringfügige Änderungen in der JSP-Datei der Warenkorb-Seite nötig.

Um den Komponenten-Mechanismus des Frameworks nutzen zu können, musste das AjaxPage-Tag in das JSP-Dokument der Warenkorb-Seite eingefügt werden. Zudem wurde der Inhalt der Warenkorb-Tabelle in ein AjaxComponent-Tag eingebettet, um den Tabelleninhalt später über eine AKF-Anfrage aktualisieren zu können.

Der Link zum Senden der Schnelleingabe-Daten wurde mit einem alternativen Link für nicht JavaScript-fähige Clients erweitert. Somit kann die Funktionalität der Schnelleingabe auch ohne die Aktivierung von JavaScript verwendet werden. Abbildung 30 verdeutlicht die neue Struktur der JSP-Datei nach Einführung der AKF-Tags.

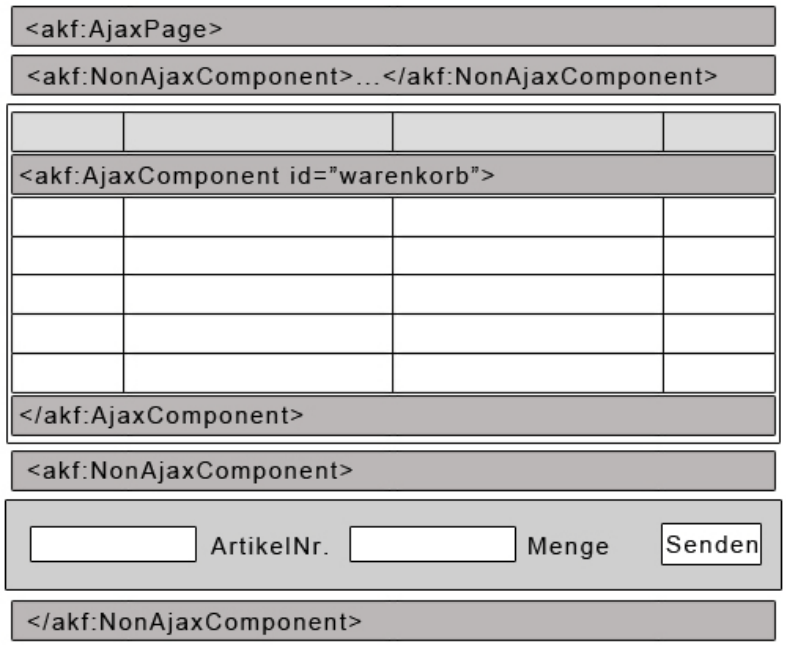

Abbildung 30 Struktur der Warenkorb-JSP-Seite nach Einführung des Frameworks

Falls der Artikel nicht in den Warenkorb eingefügt werden konnte, gibt die Controller-Logik der Anwendung nicht die JSP-Datei des Warenkorbs, sondern eine JSPX-Datei mit einer AKP-Response-Nachricht zurück. Die AKP-Response-Nachricht enthält in diesem Fall die Fehlernachrichten. Auf der Seite des Clients ist ein State-Handler für die Darstellung des Anwendungsfehlers zuständig.

Der Komponenten-Mechanismus des AKF ist nur ein möglicher Ansatz um die Funktionalität des beschriebenen Widgets umzusetzen. Das Widget hätte beispielsweise auch mit Hilfe des Template-Mechanismus oder einer Custom-Anweisung umgesetzt werden können.

7. Ergebnisse und Fazit

Das Ergebnis dieser Arbeit ist ein flexibles und erweiterbares Ajax-Framework, welches leicht an die Bedürfnisse eigener Projekte angepasst werden kann. Folgende Punkte fassen die Vorteile und Funktionen des Frameworks zusammen.

- Das Framework kann über die Definition eigener Custom-Anweisungen und Script-Anweisungen erweitert werden. Des Weiteren ermöglicht das AKF die Definition anwendungsspezifischer Fehler-Kodes, was die Behandlung von Anwendungsfehlern erleichtert.
- Über die Spezifikation des plattformunabhängigen Kommunikationsprotokolls AKP wird die Implementierung des Frameworks auf unterschiedlichen Plattformen ermöglicht.
- Der Ablauf asynchroner Serveranfragen kann über die Implementierung eigener State-Handler beeinflusst werden, wodurch eigene Funktionen in den spezifizierten Ablauf der Client-Logik eingefügt werden können.
- Mit Hilfe des AKF kann die Leistung einer Webseite über das serverseitige Rendering von Komponenten und dem Senden von reinen XML-Daten gesteigert werden. Über das Template-Caching kann zusätzlich Rechenleistung eingespart werden.
- Der Template-Mechanismus des AKF ermöglicht die Erstellung eines Templates innerhalb der Beschreibung des Webseiten-Layouts.
- Die Integration des Frameworks in ein bestehendes E-Commerce Projekt hat gezeigt, dass die Integration des AKF und die Umsetzung von Seitenkomponenten mit Hilfe weniger Schritte geschehen kann.

Mit Hilfe der Mechanismen des AKF konnten alle Anforderungen an das Framework eingehalten werden.

Literaturverzeichnis

ajaxpattern.org - Message Pattern. Abgerufen am 27. Januar 2011 von ajaxpattern.org - Message Pattern: http://ajaxpatterns.org/HTML_Message

ajaxpatterns.org. Abgerufen am 20. 12 2010 von ajaxpatterns.org: http://ajaxpatterns.org/Patterns

ajaxpatterns.org - Templating. Abgerufen am 24. 1 2011 von ajaxpatterns.org - Templating: http://ajaxpatterns.org/Browser-Side_Templating

ajaxpatterns.org - XML_Message. Abgerufen am 26. Januar 2011 von ajaxpatterns.org - XML_Message: http://ajaxpatterns.org/XML_Message

Basham, B., Sierra, K., & Bates, B. (2008). *Head First Servlets & JSP*. Sebastopol: O'Reilly.

docs.jquery.com. Abgerufen am 20. Januar 2011 von docs.jquery.com: http://docs.jquery.com/Ajax_Events

Gamperl, J. (2006). *Ajax - Web 2.0 in der Praxis.* Bonn: Galileo Press.

Jäger, K. (2008). *Ajax in der Praxis- Grundlagen, Konzepte, Lösungen.* Berlin: Springer-Verlag.

Locke, J. *ApacheWicket*. Abgerufen am 12. Januar 2011 von ApacheWicket: http://wicket.apache.org/meet/introduction.html

Marinschek, M., Kurz, M., Müllan, G., Petracek, G., Kröger, M., & Schnabl, A. (kein Datum). *Irian.at*. Abgerufen am 24. Februar 2011 von Irian.at: http://jsfatwork.irian.at/book_de/ajax.html

O'Reilly, T. (30. September 2005). *Oreilley.de*. Abgerufen am 28. 01 2011 von http://www.oreilly.de/artikel/web20.html

Prof. Dr. Andreas Meier, D. H. (2008). *Ebusiness & Ecommerce: Management Der Digitalen Wertschopfungskette* . Berlin: Springer-Verlag.

Prof. Müller, B. (2006). *Java Server Faces - Ein Arbeitsbuch für die Praxis.* München: Carl Hanser Verlag.

Ryan Asleson, N. T. (2006). *Foundations of Ajax.* New York: Springer-Verlag.

Springsource. (2006). *Springsource.org - Marshaller and Unmarshaller*. Abgerufen am 12. Februar 2011 von Springsource.org - Marshaller and Unmarshaller: http://static.springsource.org/spring-ws/docs/1.0-m3/reference/html/ch05s02.html

w3.org - HTTP. Abgerufen am 12. Januar 2011 von w3.org - HTTP: http://www.w3.org/Protocols

W3C. (3. August 2010). *World Wide Web Consortium - XMLHttpRequest Specification*. Abgerufen am 11. Februar 2011 von http://www.w3.org/TR/XMLHttpRequest/

Wähner, K. (2011). Webframeworks - eine Kategorisierung. *Java - Magazin* , 34-42.