Integrierung von Ajax in JSF-Anwendungen

Claudia Leeb

BACHELORARBEIT

Nr. 238-005-040-A

eingereicht am
Fachhochschul-Bachelorstudiengang

Medientechnik und -design

in Hagenberg

im Jänner 2008

Diese Arbeit entstand im Rahmen des Gegenstands

Web Applications

im

Wintersemester 2007/08

Betreuer:

DI Rimbert Sommer-Rudisch

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 28. Januar 2008

Claudia Leeb

Inhaltsverzeichnis

Erklärung **iii**

Vorwort **vii**

Kurzfassung **viii**

Abstract **ix**

1 Einleitung **1**

2 JSF und Ajax: eine gute Ehe? **3**
2.1 Einführung in JSF 3
2.1.1 Warum JavaServer Faces? 3
2.1.2 Das MVC-Pattern 3
2.1.3 Komponentenorientierung 4
2.1.4 Der JSF-Lifecycle 5
2.2 Einführung in Ajax 8
2.2.1 Das Kind hat jetzt einen Namen! 8
2.2.2 Die Ajax-Technologien 8
2.2.3 Funktionsweise von Ajax 9
2.3 JSF und Ajax kombinieren 10
2.3.1 Vorteile des Zusammenspiels 10
2.3.2 Fallstricke beim Einsatz von Ajax in JSF 11

3 Aktuelle Ajax-JSF-Frameworks **13**
3.1 Reichhaltige Open Source-Komponentenframeworks 14
3.1.1 MyFaces Tomahawk 14
3.1.2 MyFaces Sandbox 14
3.1.3 MyFaces Trinidad 15
3.1.4 JBoss RichFaces 15
3.1.5 ICEfaces 17
3.1.6 Frameworkübersicht 18
3.2 GUI-spezialisierte Open Source-Frameworks 19
3.2.1 MyFaces Tobago 19

3.2.2 RCFaces . . . 19
3.2.3 WebGalileo Faces . . . 19
3.2.4 Woodstock . . . 20
3.2.5 YUI4JSF . . . 20
3.2.6 Frameworkübersicht . . . 20
3.3 Gemischte Open Source-Komponentenframeworks . . . 21
3.3.1 Blueprints . . . 21
3.3.2 jenia4faces . . . 22
3.3.3 xulfaces . . . 22
3.3.4 Frameworkübersicht . . . 22
3.4 Open Source-Ajax-Frameworks . . . 23
3.4.1 AjaxAnywhere . . . 23
3.4.2 J4Fry . . . 24
3.4.3 jMaki . . . 25
3.4.4 jsf-extensions . . . 25
3.4.5 DWR . . . 26
3.4.6 Frameworkübersicht . . . 26
3.5 Bekannte kommerzielle Komponentenframeworks . . . 27
3.5.1 NetAdvantage . . . 27
3.5.2 QuipuKit . . . 28
3.5.3 Backbase . . . 28
3.5.4 Simplica . . . 29
3.5.5 Frameworkübersicht . . . 29
3.6 Weitere kommerzielle Komponentenframeworks . . . 29
3.6.1 BindowsFaces . . . 29
3.6.2 ZK . . . 30
3.6.3 JavaServer Faces Widget Library . . . 31
3.6.4 ILOG JSF Tools . . . 31

4 Probleme und Lösungsansätze 34
4.1 Layout und Kompatibilität . . . 34
4.1.1 Ajax-Frameworks einbinden . . . 34
4.1.2 Komponentenbibliotheken kombinieren . . . 35
4.2 JSF-Standards . . . 35
4.2.1 Potentielle Problemquellen in JSF-Frameworks . . . 36
4.2.2 Ausblick: JSF 2.0 . . . 37
4.3 Performance . . . 38
4.3.1 Serverseitige vs. clientseitige Statusspeicherung . . . 38
4.3.2 Facelets steigern die Performance . . . 39
4.3.3 Unterschiede zwischen JSF-Implementierungen . . . 39
4.3.4 Rendermethoden . . . 40

5 Schlussbemerkungen 42

A Frameworktabellen – Erläuterungen **44**
A.1 Allgemeines zu den Tabellen 44
A.1.1 Erklärung der angeführten Kriterien 44

B Inhalt der CD **46**

Literaturverzeichnis **47**

Vorwort

JavaServer Faces machen die Entwicklung von Webapplikationen um vieles einfacher, wenn man das Prinzip erst einmal durchschaut hat. Das war auch mein Gedanke, nachdem ich während eines Sommerpraktikums mein erstes kleines JSF-Projekt erledigt hatte – übrigens ausschließlich mit Standard-JSF und CSS-Stylesheets. Erst während der Realisierung des zweiten Teilprojektes entdeckte ich die Möglichkeiten, die sich darüber hinaus noch bieten – so schloss ich innige Freundschaft mit der MyFaces Tomahawk-Komponentenbibliothek.

Einmal auf den grünen Zweig gekommen und neugierig geworden, begann ich, mich in der großen, bunten JSF-Welt noch genauer umzusehen – diese stellte sich als noch viel umfangreicher heraus als ich erwartet hatte – und so kam ich auf die Idee, diese Erkenntnis doch als Basis für die vorliegende Bakkalaureatsarbeit aufzugreifen.

Das vorliegende Dokument soll eine Hilfestellung für all jene bieten, die ein Framework einsetzen möchten, sich jedoch nicht tagelang mit der Evaluierung von diversen Bibliotheken befassen können oder wollen. Natürlich kann es niemals das Lesen einer Dokumentation und das Ausprobieren von Demoapplikationen ersetzen, jedoch denke ich, dass es anhand der vorgenommenen Evaluierung und Kategorisierung auf jeden Fall möglich ist, eine Vorauswahl von zwei oder drei Frameworks zu treffen, die man sich anschließend im Detail zu Gemüte führen kann.

Im Laufe der Recherchen habe ich allerdings auch festgestellt, dass der Einsatz von Frameworks nicht zwingend „Friede, Freude, Eierkuchen“ bedeutet. An dieser Stelle geht mein besonderer Dank an Alexander Bell und Ganesh Jung, Mitgründer des Frameworks J4Fry, durch deren Website ich auf die tatsächlich vorhandenen Probleme aufmerksam geworden bin – und die mich bei der Vervollständigung des betreffenden Kapitels unterstützt haben.

Kurzfassung

Das bekannte Web-Framework JavaServer Faces liegt derzeit – nicht zuletzt durch die zunehmende Integrierung von Ajax-Features – sehr im Trend. Es vergeht kaum eine Woche, in der nicht neue Komponentenbibliotheken bzw. neue Features angekündigt werden. Dieser Umstand macht es nicht leichter, den Überblick zu wahren – vor allem nicht für Einsteiger in die JSF-Technologie.

Die vorliegende Arbeit bietet zu Beginn Grundlagenwissen sowohl über Ajax als auch JSF, welches für die weitere Lektüre unbedingt notwendig ist. Im Hauptteil werden die derzeit verfügbaren Ajax-JSF-Frameworks möglichst objektiv untersucht und kategorisiert. Besonderes Augenmerk wird dabei auf den Leistungsumfang und die Besonderheiten der einzelnen Frameworks gelegt, sowie auch auf eventuelle Inkompatibilitäten zwischen verschiedenen Bibliotheken.

Da die Verwendung von Frameworks – vor allem wenn dabei zwei so unterschiedliche Technologien wie das serverzentrierte JSF und das clientseitige Ajax verwendet werden – oft zu Problemen führen kann, werden auch potentielle Problemquellen näher beleuchtet. Desweiteren werden nach Möglichkeit auch Lösungsvorschläge bzw. Alternativen aufgezeigt.

Abstract

The server-based web-framework JavaServer Faces in connection with Ajax seems to be very trendy at the moment. Nearly every week you hear from new released frameworks or new implemented features in existing ones, that's why the world of JSF becomes more and more unclear – especially for beginners.

The present work gives a short introduction in JSF as well as in Ajax, because these basic principles are essential for the further chapters. The main part of this work tries to evaluate and categorize existing Ajax-JSF-frameworks in a reasonable manner, in order to facilitate getting an overview about existing resources as well as getting information about (in)compatibilities between different frameworks.

As the use of frameworks — especially if they consist of very different technologies like Ajax and JSF — can cause a lot of problems, this work will also deal with common problems in the development with Ajax-JSF-frameworks and methods of resolution that eventually exist.

Kapitel 1

Einleitung

JavaServer Faces ist ein bekanntes Java-Framework, das auf strikter Implementierung des MVC-Patterns basiert und besonders bei Enterprise Applikationen zur Anwendung kommt. Zu Beginn wurde das Projekt als wenig performant, schwerfällig und kompliziert abgetan – mittlerweile hat sich die Performance wesentlich verbessert und durch die steigende Akzeptanz entstanden mit der Zeit auch diverse Frameworks, die den Umgang mit JSF erleichterten. Spätestens seit der Integrierung in die Java Enterprise Edition (*Java EE*) erfreut sich das Framework stark wachsender Beliebtheit.

In dieser Zeit wurde auch JavaScript – in Form von Ajax – wieder entdeckt. Es war nur eine Frage der Zeit, bis JSF-Framework-Entwickler Ajax zu integrieren versuchten um mehr Leben in JSF-Anwendungen zu bringen. Seitdem geht es Schlag auf Schlag – ständig tauchen neue Ajax-JSF-Frameworks auf und bestehende Bibliotheken entwickeln sich ebenfalls kontinuierlich weiter. Gleichzeitig bleibt aber auch die Entwicklung des JSF-Standards selbst nicht stehen – was moderne Frameworks ebenfalls wieder berücksichtigen müssen. Es liegt also auf der Hand, dass es alles andere als einfach ist, den Überblick über das Angebot zu bewahren.

Genau dies versucht die vorliegende Arbeit – zumindest in Form einer Momentaufnahme. Im folgenden Kapitel werden die Grundzüge von JSF und Ajax erläutert, damit auch Leser, die mit der Materie nicht so vertraut sind, folgen können. Zudem wird auf Vor- und Nachteile von Ajax eingegangen.

Im anschließenden Hauptteil werden die derzeit bekannten Frameworks näher untersucht und kategorisiert. Besonderes Augenmerk liegt auf dem Funktionsumfang und der Kompatibilität zu anderen Frameworks. Zudem wurde versucht, die Besonderheiten jedes Frameworks herauszuarbeiten, um so die jeweilige Individualität[1] zu unterstreichen. Es sei ausdrücklich darauf hingewiesen, dass die Recherchen natürlich nur eine Momentaufnahme darstellen, daher wird bei jedem untersuchten Framework auch das Recherchedatum angegeben. Es kann mit ziemlich großer Sicherheit davon ausge-

[1] sofern vorhanden...

gangen werden, dass sich die Frameworklandschaft mittlerweile – also knapp zwei Monate später – bereits wieder um einiges verändert hat.

So beliebt die Arbeit mit Frameworks auch ist – durch ihre Verwendung entstehen oft Probleme, die allerdings von den Framework-Entwicklern gerne unter den Teppich gekehrt bzw. ignoriert werden. Das letzte Kapitel beleuchtet bekannte Probleme bzw. Fehlerquellen in der Arbeit mit Ajax-JSF-Frameworks und – sofern existent – auch mögliche Lösungen dazu.

Kapitel 2

JSF und Ajax: eine gute Ehe?

2.1 Einführung in JSF

2.1.1 Warum JavaServer Faces?

So viele verschiedene Frameworks es auch gibt – sie haben natürlich alle ein erklärtes, gemeinsames Ziel: dem Entwickler die Arbeit zu erleichtern. Auch JavaServer Faces machen da keine Ausnahme. Bei der Entwicklung mit gewöhnlichem JSP[1] muss sich der Entwickler um vieles selbst kümmern: Zustandsüberwachung, Sessionverwaltung, Internationalisierung... um nur einiges zu nennen. Entsprechend hoch ist der Implementierungsaufwand. JSF übernimmt den Großteil der Routineaufgaben des Programmierers und noch viel darüber hinaus. Wichtige Prinzipien von JSF sind in den folgenden Abschnitten dargestellt. Die volle Palette zu präsentieren würde allerdings den Rahmen dieser Arbeit sprengen – für eine vollständige Auflistung sei daher auf [2, Kap. 4] verwiesen.

2.1.2 Das MVC-Pattern

Das MVC-Pattern (*Model-View-Controller*) ist ein gängiges Entwurfsmuster in der Entwicklung von Webanwendungen, besonders wenn es um die Realisierung von graphischen Benutzeroberflächen (*GUI*) geht. Das Kernprinzip ist, dass die Applikation streng in die Bereiche Geschäftsmodell, Anwendungslogik und Darstellung getrennt wird.

Model

Das *Model* repräsentiert die Geschäftslogik der Anwendung und besteht aus Java-Klassen, die die fachliche Logik enthalten. Ein Beispiel ist eine Bean namens *Kunde*, die alle Daten zu einem Kunden speichert und eventuell kundenspezifische Methoden enthält.

[1] JSP, http://java.sun.com/products/jsp

View

Die *View* ist ausschließlich für die Darstellung der Applikationsinhalte vorgesehen und darf somit keinerlei Anwendungslogik enthalten. Standardmäßig ist für die View HTML in Verbindung mit JSP vorgesehen, aber auch andere Formate wie beispielsweise WML sind möglich.

Controller

Der Begriff *Controller* repräsentiert die Anwendungslogik und ist zum einen die Verbindung zwischen dem Geschäftsmodell und der Darstellung, zum anderen verantwortlich für die Navigation innerhalb der Applikation. In der Regel übernehmen diese Aufgaben sogenannte *Handler*, also Java-Klassen, die u. a. Methoden zur Steuerung von *Navigationsregeln*[2] und zum Datenabgleich von Model und View enthalten.

2.1.3 Komponentenorientierung

JavaServer Faces arbeiten komponentenbasiert. Das heißt, der Oberflächenentwickler setzt aus den verfügbaren „Bausteinen“ seine Anwendung zusammen. Sollte der Standardkomponentensatz aus irgendeinem Grund nicht ausreichen, kann der Entwickler entweder eigene Komponenten programmieren oder auf verfügbare Bibliotheken zurückgreifen. Auf letzteres wird im nächsten Kapitel näher eingegangen. Ein Vorteil der Komponentenbasierung ist, dass hiermit die Applikationsentwicklung sehr einfach auf verschiedenste Spezialisten aufgeteilt werden kann, falls das nötig ist. Solche Spezialgebiete sind beispielsweise

- Erstellen der View - also Komponenten zusammensetzen und Styling mittels CSS
- Implementierung der Anwendungslogik
- Entwicklung neuer Komponenten

Ein Java-Entwickler muss also kein HTML und CSS beherrschen, genauso wenig muss der Grafikspezialist eine Ahnung von fortgeschrittener Programmierung haben. Im übrigen sei gesagt, dass HTML in Verbindung mit JSF kaum mehr benötigt wird[3] – im Gegenteil: es soll sogar so weit als möglich vermieden werden. Manche der Bibliotheken, die im nächsten Kapitel vorgestellt werden, stellen sogar für JSF optimierte HTML Tags zur Verfügung. Sollte Interesse bzw. Bedarf bestehen, sei auf die jeweilige API/Taglib verwiesen.

[2] Rückgabewert ist ein String, der in der Datei *faces-config.xml* definiert ist.
[3] HTML muss nicht mehr programmiert werden, dieser Code wird von JSF erzeugt!

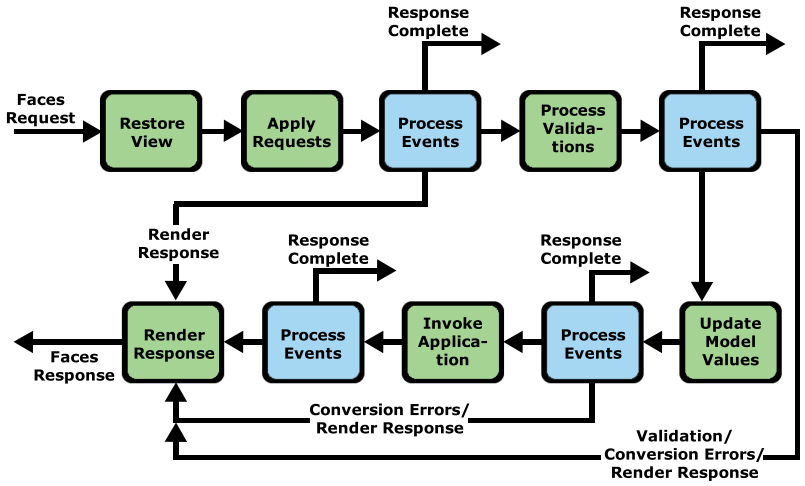


Abbildung 2.1: Lifecycle einer JSF-Anfrage
Quelle: http://java.sun.com/javaee/5/docs/tutorial/doc/bnaqq.html

2.1.4 Der JSF-Lifecycle

Das Herzstück einer JSF-Applikation ist die frameworkeigene Anfragebearbeitung, welche die gesamte Eventabarbeitung, Validierung und das Rendering übernimmt. Auch wenn es sich hierbei um einen relativ komplexen Prozess handelt sei festgehalten [2, S. 47]:

> JavaServer Faces basieren auf der Servlet API - und somit auf dem Request-Response-Modell des HTTP-Protokolls. Damit erben sie auch alle Eigenheiten von diesem, inklusive Zustandlosigkeit von HTTP [...].

Der Lifecycle gliedert sich, wie in Abb. 2.1 gezeigt, im Wesentlichen in sechs Phasen, zwischen denen meist Events (sofern vorhanden) an entsprechende EventListener übergeben und weitere Listener informiert werden. Nicht immer müssen alle Phasen durchlaufen werden, beispielsweise kann beim Auftritt eines Fehlers jederzeit zur letzten Phase gesprungen werden, um die Antwort sofort darzustellen. In den folgenden Abschnitten werden diese Phasen genauer beschrieben.

Wiederherstellung des Komponentenbaumes
Restore View

JSF-Komponenten einer Seite werden – ähnlich einem DOM-Baum – in einer Baumstruktur gespeichert. Die Wurzel dieses Baumes ist immer eine

<f:view>-Komponente, in der sich alle weiteren Komponenten befinden müssen. Beim erstmaligen Besuch einer JSF Seite wird dieser Baum völlig neu erstellt, d. h. es wird sofort die Antwort generiert. Gleichzeitig werden die Komponenten je nach Konfiguration entweder server- oder clientseitig gespeichert.

Wenn nun an eine bereits besuchte JSF Seite eine weitere Anfrage gesendet wird, wird der gesamte speicherintensive Komponentenbaum – also der Inhalt der View – aus den zuvor gespeicherten Komponenten wieder aufgebaut. Dieses System hilft Speicherplatz zu sparen, da die Zeit, die für eine Anfrage benötigt wird, im Normalfall um ein Vielfaches kürzer ist als die Zeit zwischen zwei Anfragen des selben Benutzers.

Übernahme der Anfragewerte
Apply Requests

Wird eine Anfrage abgesetzt, werden die Inhalte der diversen Eingabekomponenten als POST-Parameter kodiert. Der Zweck dieser Phase ist es, eben diese Inhalte zu dekodieren und den entsprechenden Komponenten im Baum zuzuweisen. Diese Zuweisung ist jedoch nur vorläufig, da die Werte durch eventuelle Fehlerfälle bei Konvertierung und Validierung in späteren Phasen noch verändert werden können.

Sollen jedoch Konvertierung und Validierung schon in dieser Phase stattfinden und die Zuweisung des Wertes einer bestimmten Komponente schon fix sein, so kann für jede Komponente extra das *immediate=true* Attribut gesetzt werden. Dies kann beispielsweise praktisch sein, wenn eine Eingabe richtungsweisend für alle anderen ist und daher schon vor diesen auf Korrektheit geprüft werden soll.

Validierung
Process Validations

Wie der Name schon sagt, werden in dieser Phase die zugewiesenen Werte durch die registrierten *Validatoren* auf Richtigkeit überprüft. Manche Werte müssen vor der Validierung in ein anderes Format, beispielsweise String in Integer, konvertiert werden. Dies geschieht durch sogenannte *Converter*.

Für beiderlei – Validatoren wie Converter – gibt es bereits einige Standardimplementierungen, z. B. für die Längenbeschränkung einer Eingabe, Minimum- und Maximumwert usw. Einige Erweiterungen wie die Komponentenbibliothek[4] Tomahawk bieten fortgeschrittene Converter und Validatoren an, beispielsweise für Kreditkartennummern und reguläre Ausdrücke. Mit Abschluss der Validationsphase besitzen alle Komponenten ihren endgültigen Wert.

[4]Diese werden im nächsten Kapitel behandelt

Aktualisierung der Modellobjekte
Update Model Values

Wird diese Phase erreicht, besitzen alle Werte ihre Gültigkeit, d. h. sie sind valide und vom richtigen Typ. Diese Werte werden nun ins Model – meist Java-Beans – übernommen.

Aufruf der Anwendungslogik
Invoke Application

Wenn alle Eingaben validiert und die Modell-Klassen aktualisiert wurden, kann – wenn gewünscht – Anwendungslogik, d. h. Handler-Methoden, aufgerufen werden. Dies geschieht über *ActionListener* und *Actionmethoden*, die über Buttons und Links registriert werden können.

Actionmethoden werden einfach über das *action*-Attribut einer Komponente registriert:

```
1  <h:commandButton id="testbutton" value="testen" action="#{Testhandler.
       testmethode}" />
```

Actionmethoden werden bevorzugt zur Steuerung der Navigation verwendet. Ist das Ziel der Navigation ohnehin klar und sind keine weiteren Handlungen notwendig, kann auf eine Actionmethode verzichtet und somit gleich ein fixer String gesetzt werden:

```
1  <h:commandButton id="testbutton" value="testen" action="weiter" />
```

ActionListener werden über das *actionListener*-Attribut einer Komponente registriert:

```
1  <h:commandButton id="testbutton" value="testen" actionListener="#{
       Testhandler.testmethode}" />
```

ActionListener werden verwendet, wenn auf die implementierenden Komponenten zugegriffen werden muss oder Parameter mitgegeben werden.

Darstellung der Antwort
Render Response

In dieser letzten Phase werden die Werte der Komponenten wieder kodiert und der Komponentenbaum gespeichert, damit er bei der nächsten Anfrage wiederhergestellt werden kann. Natürlich wird – wie der Name schon sagt – hier auch die Antwort generiert. Der Vorgang der Antwortgenerierung wird gewöhnlich als *Renderphase* bezeichnet, daher wird dieser Begriff im weiteren Verlauf der Arbeit ebenso verwendet.

2.2 Einführung in Ajax

2.2.1 Das Kind hat jetzt einen Namen!

Als der Begriff Ajax/AJAX[5] das Licht der Welt erblickte, war die dahinter liegende Technologie schon nichts Neues mehr. Ajax ist nicht einmal eine Technologie im klassischen Sinne, sondern eine Sammlung bewährter Technologien unter einem Marketingbegriff [5, S. 391], der 2005 von Jesse J. Garret erfunden wurde[6]. Ajax steht für *Asynchronous JavaScript and XML*, die enthaltenen Technologien werden im folgenden Abschnitt kurz erläutert.

2.2.2 Die Ajax-Technologien

(X)HTML und CSS

(X)HTML und CSS sind *die* Basistechnologien des Web. HTML ist eine *Markup*sprache, die zum logischen Aufbau einer Webseite benutzt und vom Browser interpretiert wird. CSS ist für die Designkomponente in Webapplikationen zuständig.

Document-Object-Model (DOM)

(X)HTML-Dokumente (und XML-Dokumente) bestehen aus ineinander verschachtelten Elementen (*Tags*). Um auf die einzelnen Elemente zugreifen zu können, wurde vom *World Wide Web Consortium*[7] DOM als Schnittstelle spezifiziert. Durch DOM – natürlich mit Hilfe einer geeigneten Scriptsprache wie JavaScript – ist es möglich, Teile einer Webseite zur Laufzeit und ohne völliges Neuladen der Seite auszutauschen bzw. zu verändern. Somit kommt mehr Dynamik ins Web.

XMLHttpRequest-Objekt

Der Ursprung des XMLHttpRequests ist bei Microsoft zu finden [5, S. 394], als eine Schnittstelle für Outlook Web Access entwickelt werden sollte, die im Hintergrund auf neue Mails prüft. Der XMLHttpRequest tut nichts anderes als Anfragen an den Server zu schicken und die Antwort auszuwerten.

XML/XSLT und JSON

Obwohl die Markupsprache XML im vollen Namen von Ajax vorkommt, wird sie in der Praxis kaum in Verbindung mit Ajax verwendet [5, S. 391]. Meist kommt das Austauschformat JSON (*JavaScript Object Notation*)[8] beim Da-

[5] Ajax im Sinne der Technologie natürlich, nicht des Putzmittels...

[6] *A New Approach to Web Applications*, http://www.adaptivepath.com/ideas/essays/archives/000385.php

[7] W3C, www.w3.org

[8] JSON Homepage, http://www.json.org/

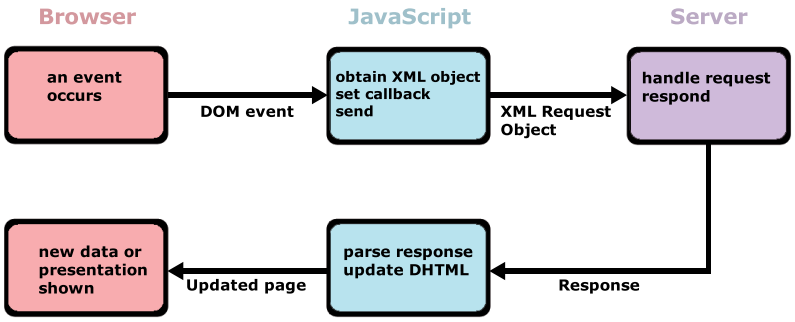


Abbildung 2.2: Ein typischer Ajax-Request.
Quelle: http://code.google.com/edu/client/ajax-tutorial.html

tentransfer zwischen Server und Client zum Zug.

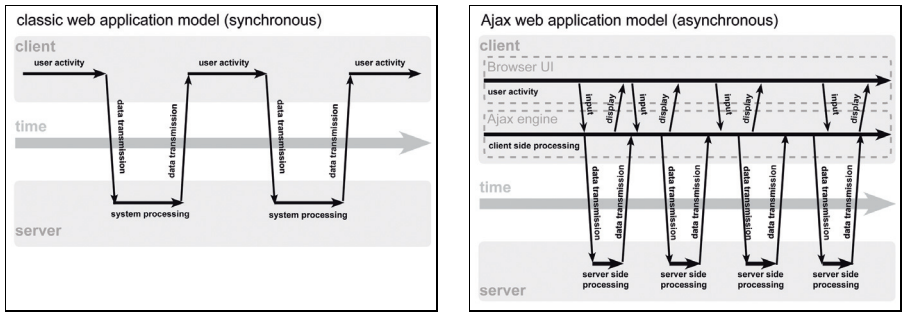
JavaScript

JavaScript wird benötigt, um die vorher genannten Technologien unter einen Hut zu bringen [2, S. 198]. Es spricht das XMLHttpRequest-Objekt an, kann mittels JSON übermittelte Daten einfach mit der *eval()*-Funktion auswerten und die so extrahierten Inhalte als HTML- und CSS-Code in den DOM-Baum einschleusen.

2.2.3 Funktionsweise von Ajax

Der wesentliche Punkt, der eine Ajax-Anwendung auszeichnet, ist die Tatsache, dass eine Webseite nicht ständig neu geladen werden muss – dies ist in Abb. 2.2 dargestellt. Beim erstmaligen Laden der Seite wird diese wie gewohnt aufgebaut – weitere Requests werden durch registrierte Events ausgelöst und basieren auf dem XMLHttpRequest-Objekt. XMLHttpRequest wartet anschließend auf die vom Server generierte Antwort, wertet diese aus und schleust im Erfolgsfall die erhaltenen Daten in das DOM der Webseite ein. Das heißt, nur die betroffenen Elemente der Seite werden neu gerendert.

Durch den Einsatz des XMLHttpRequests muss der Benutzer nicht warten bis die Seite neu geladen wird, sondern kann sofort weiterarbeiten – sofern der Entwickler asynchrone Requests verwendet. In Abb. 2.3 ist der Unterschied zwischen klassischer Interaktion im Web und einer Ajax-Applikation schematisch dargestellt.



(a) (b)

Abbildung 2.3: Interaktionsmodell einer klassischen HTTP-Seite (a) im Vergleich zu einer mit Ajax angereicherten Seite (b).
Quelle: http://www.adaptivepath.com/ideas/essays/archives/000385.php

2.3 JSF und Ajax kombinieren

Wie schon erwähnt, setzen JavaServer Faces auf dem HTTP-Protokoll auf, was bedeutet, dass sich eine klassische JSF-Anwendung aus Sicht des Benutzers nicht anders verhält als eine „normale“ HTML- oder PHP-Applikation. Soll mehr Interaktivität in einer Anwendung erzielt werden, lässt sich mit Ajax Abhilfe schaffen, jedoch gibt es auch einige Details, auf die achtzugeben ist.

2.3.1 Vorteile des Zusammenspiels

Bildschirmflackern gehört der Vergangenheit an

Wird klassisches JSF eingesetzt, wird bei jedem Request die ganze Seite neu geladen, auch wenn nur eine Zeile zu ändern ist. Das vollständige Neuladen der Seite führt jedoch zu dem bekannten, unangenehmen Flackern der Anzeige. Mit Ajax ist es möglich zu veranlassen, dass nur Teile der Seite neu gerendert werden, der Flackereffekt entfällt somit.

Verkürzung der Renderzeit

Der Lifecycle einer JSF-Anfrage ist ein komplexer, rechenintensiver Prozess, was sich naturgemäß negativ auf die Performance von JSF-Applikationen auswirkt. Da es mit Ajax – je nach verwendetem Framework – möglich ist, nur Teilbereiche einer Seite rendern zu lassen, kann sich die Wartezeit für den Benutzer erheblich verkürzen. Die schnellere Aktualisierung steigert natürlich die Interaktivität der Anwendung.

Clientseitiger Code entlastet den Server

Warum soll immer der Server die ganze Arbeit erledigen? So manche Aufgabe kann auch Ajax und damit clientseitiger Code übernehmen – beispielsweise Validierungsaufgaben. Werden diese zum Client verlagert, muss ein Formular und damit so manche Anfrage gar nicht erst fehlerhaft abgeschickt werden. Der Server wird also geringerer Last ausgesetzt.

Mehr Interaktivität durch asynchrone Requests

Mit Ajax kann die Applikation asynchrone Requests an den Server senden. Der Benutzer kann also weiterarbeiten, obwohl die Antwort des Servers noch gar nicht erfolgt ist. Dies steigert – zusammen mit den vorher genannten Punkten – das „Desktopfeeling“ der Applikation.

Ansprechendere Applikationen durch JavaScript-Effekte

Mit Ajax, besonders mit entsprechenden Bibliotheken wie Dojo oder Scriptaculous, ist es sehr einfach, eine statische Seite mit diversen Effekten zu beleben und damit attraktiver zu gestalten.

2.3.2 Fallstricke beim Einsatz von Ajax in JSF

Ajax scheint also die perfekte Ergänzung zu JSF zu sein. So einfach ist die Integrierung von Ajax in JSF allerdings nicht – es gibt einige Faktoren, auf die der Entwickler hierbei acht geben muss. Diese seien in diesem Abschnitt überblicksweise beschrieben.

Dieser Abschnitt befasst sich allerdings nur mit allgemeinen Problemen, die bei der Verwendung von reinem Ajax in JSF-Anwendungen auftreten können. Den Komplikationen, die bei der Verwendung von JSF-Ajax-Frameworks auftreten können, ist ein eigenes Kapitel gewidmet.

Browserinkompatibilitäten

In den 90ern wurde vom „Browserkrieg“ [5, S. 23] gesprochen, in dem auch JavaScript[9] eine große Rolle spielte, da sich die großen Browser Internet Explorer und Netscape Navigator[10] ständig mit allen möglichen Features übertrumpfen wollten. Beide Hersteller entwickelten eine eigene Version der Sprache, die letztendlich zu JavaScript vereinheitlicht wurde. Auch wenn es mittlerweile etablierte JavaScript-Standards gibt, unterscheiden sich die Implementierungen aller Browser immer noch in manchen Bereichen. Diese Unterschiede müssen natürlich auch bei der Entwicklung in Verbindung mit JSF beachtet werden.

[9] in sämtlichen Abwandlungen wie JScript, LiveScript...

[10] Der Netscape Navigator wurde vor kurzem mit allen Ehren zu Grabe getragen: http://blog.netscape.com/2007/12/28/end-of-support-for-netscape-web-browsers

Bottleneck: Datenbank-Verbindungen

Die Versuchung ist groß: Der Benutzer ändert einen Wert einer Selectbox und – zack! – aktualisiert die Applikation relevante Werte. Das ganze nun noch fünf oder sechs mal, mit einigen Autosuggest-Eingabefeldern zum Drüberstreuen... und fertig ist das Formular. Bei einer Website mit wenigen Benutzern mag das nicht weiter ins Gewicht fallen, bei viel besuchten Seiten sollte der Entwickler aber darauf achten, den Server bzw. die Datenbank nicht mit überflüssigen Requests zu überfordern, da sonst die Antwortzeiten darunter leiden – oder gar der Server abstürzt[11]. Ajax-Requests wahllos einzusetzen, nur weil es „in“ ist, sollte also vermieden werden.

JavaScript kann jeder deaktivieren!

Die große Unbekannte: der Benutzer. Ajax ist JavaScript, und JavaScript kann bekanntlich jederzeit deaktiviert werden. Für solche Fälle ist ein Workaround notwendig, und sei es nur eine simple Ausgabe mit diesbezüglicher Information in einem *<noscript>*-Tag.

Imkompatibilität mit dem JSF-Lifecycle

Der Ajax-Request an sich ignoriert den JSF-Lifecycle schlichtweg. Was geschieht nun mit der Antwort, die vom Server kommt? Da der Lifecycle ignoriert wird, wird auch der Komponentenbaum von JSF nicht aktualisiert. Dies stellt kein Problem dar, wenn nur Ausgabekomponenten aktualisiert werden. Betrifft die Antwort des Servers jedoch auch Eingabekomponenten, sind Workarounds, beispielsweise über einen PhaseListener, nötig. Für detailliertere Informationen über passende Strategien sei auf die Dokumentation von Blueprints[12] verwiesen.

Accessibility von Ajax-Anwendungen

Ajax und Accessibility sind zwei Begriffe, die nach wie vor nicht richtig miteinander in Einklang zu bringen sind. Dies liegt zum Einen daran, dass manche Browser – wie beispielsweise Lynx[13] – JavaScript gar nicht erst interpretieren, zum Anderen an der Tatsache, dass zahlreiche Internetnutzer JavaScript deaktiviert haben. An Letzterem ist in erster Linie der nach wie vor nicht all zu glänzende Ruf von JavaScript schuld, da zu den Anfangszeiten der Sprache – auch dank vieler Sicherheitslücken in den Browsern – viel Missbrauch damit getrieben wurde.

[11] Früher als sonst, wohlgemerkt...
[12] Blueprints, https://blueprints.dev.java.net/bpcatalog/ee5/ajax/usingJSFwithAJAX.html
[13] Lynx, http://lynx.browser.org

Kapitel 3

Aktuelle Ajax-JSF-Frameworks

Die JavaServer Faces-Technologie hat sich mittlerweile am Markt etabliert – und nicht zuletzt durch die Entwicklung von ansprechenden weiterführenden Komponentenbibliotheken. Die Integration von Ajax in diversen Projekten macht die Entwicklung von Weboberflächen mit JavaServer Faces noch um vieles leichter, da sich der Entwickler die Programmierung von JavaScript oft zur Gänze ersparen kann.

Eine Applikation, die nur aus den Standardkomponenten besteht, ist ohne erheblichen zusätzlichen Aufwand nicht sonderlich aufregend. Erst mit der Entstehung der Erweiterungen wurde es einfacher, solche Webapplikationen performancemäßig zu pushen, ansprechender zu gestalten und so beim Anwender ein „Desktopfeeling“ zu erzeugen. Kurz gesagt, die Kombination von JSF und Ajax liegt im Trend.

Fakt ist, JSF-Ajax-Frameworks schießen wie Pilze aus dem Boden. Über 20[1] Open Source und kommerzielle Frameworks sind derzeit verfügbar – natürlich erhebt jedes für sich den Anspruch auf absolute Genialität und Usability – und doch sehen auf den ersten Blick alle gleich aus. Um herauszufinden, was ein bestimmtes Framework wirklich auszeichnet, muss sich der Entwickler bei einigen Projekten ziemlich tief in die Dokumentation einlesen – und entsprechendes JSF- und Ajax-Wissen mitbringen.

Wie werden nun ein oder mehrere geeignete Frameworks für ein JSF-Projekt ausgewählt? Wichtig ist vor allem zu wissen, was mit der Applikation eigentlich erreicht werden soll, erst dann kann gezielt nach einem passenden Framework gesucht werden. Die folgende Sammlung von Frameworks und deren Beschreibung geht vor allem auf die Besonderheiten der jeweiligen Projekte und deren Eignung ein. Ich habe mich nach bestem Wissen und Gewissen bemüht, dabei objektiv vorzugehen und keinen Prüfkandidaten stiefmütterlich zu behandeln.

[1] Natürlich erhebt die von mir erstellte Sammlung keinen Anspruch auf Vollständigkeit!

3.1 Reichhaltige OS-Komponentenframeworks[2]

3.1.1 MyFaces Tomahawk

Erweiterte Standardkomponenten, GUI, Effekte, Validatoren

MyFaces Tomahawk[3] beinhaltet im ursprünglichen Sinne erweiterte Standardkomponenten. Diese sind fast durchwegs mit folgenden zusätzlichen Attributen ausgestattet:

- *User-role Awareness*: Komponenten werden nur gerendert, wenn der Benutzer bestimmte Berechtigungen hat. Letztere werden beispielsweise mit Hilfe eines Applikationsservers wie Tomcat[4] administriert.
- *DisplayValueOnly*: Formularfelder können deaktiviert werden.
- *forceId*: Die ID, die der Entwickler einer Komponente zuweist, wird nicht durch kontext-generierte JSF-ID ersetzt. Dadurch können Probleme bei der Ansprache von Elementen mittels CSS und JavaScript vermieden werden.

Ein besonderes Feature von Tomahawk sind die umfangreichen Validierungsmöglichkeiten wie z. B. Email- und Kreditkartenvalidatoren. Zudem bietet Tomahawk eine Integration des Dojo-Toolkits[5] an. Das Projekt wird beständig erweitert - und zwar durch die im folgenden Abschnitt beschriebenen Komponenten von MyFaces Sandbox. Tomahawk beinhaltete ursprünglich gar keine Ajax-Funktionalität, da die Sandbox-Komponenten allerdings großteils Ajax integrieren, zählt auch Tomahawk mittlerweile zu den JSF-Ajax-Frameworks.

3.1.2 MyFaces Sandbox-Komponenten

Die Sandbox[6] ist ein Subprojekt von Tomahawk. Hier befinden sich alle Komponenten, Validatoren, Converter, die das Test- und Entwicklungsstadium noch nicht abgeschlossen haben. Diese Komponenten können – „auf eigene Gefahr“, wohlgemerkt – bereits verwendet werden, jedoch besteht keine Garantie, dass sie jemals offiziell Teil von Tomahawk werden.

Im Sandbox-Stadium befinden sich hauptsächlich Ajax-lastige Komponenten, die auf dem Ajax-Framework Dojo aufsetzen. Dabei geht es in erster Linie um clientseitige Validierung, Benutzerinteraktion (Alertboxen, Messaging) und um Effekte aller Art.

---

[2] OS = Open Source

[3] MyFaces Tomahawk, http://myfaces.apache.org/tomahawk/

[4] Tomcat Realm,
http://tomcat.apache.org/tomcat-5.5-doc/realm-howto.html#WhatisaRealm?

[5] Dojo, http://www.dojotoolkit.org/

[6] MyFaces Sandbox, http://myfaces.apache.org/sandbox

3.1.3 MyFaces Trinidad

Erweiterte Standardkomponenten, GUI, Effekte, Validatoren

MyFaces Trinidad[7] ist ebenfalls ein Open Source Projekt der Apache Software Foundation. Ursprünglich wurden die Trinidad-Komponenten unter dem Namen ADF Faces veröffentlicht – ein Komponentensatz, der im Jahre 2006 von Oracle an die Apache Software Foundation gespendet wurde. Trinidad ist ein sehr umfangreiches Framework mit über 100 Komponenten, Validatoren usw.

Die NamingContainer API von JSF wird von Trinidad nicht implementiert, was den Vorteil bringt, dass JavaScript-Code leichter einzubinden ist. Der Entwickler muss allerdings darauf achten, dass sich die selbst vergebenen IDs nicht wiederholen. Der Unterschied ist in Programm 3.1 ersichtlich: das erste Formular wird mit der Standardkomponente realisiert, das zweite mit der Trinidad-Variante.

```
1  <h:form id="foo">
2  <!-- Diese Komponente hat die ID "foo:bar" -->
3    <tr:inputText id="bar"/>
4  </h:form>
5
6  <tr:form id="foo2">
7  <!-- Diese jedoch nur die ID "bar2" -->
8     <tr:inputText id="bar2"/>
9  </tr:form>
```

Programm 3.1: Beispiel: Standardimplementierung vs. MyFaces Trinidad

In allen Trindad Komponenten ist standardmäßig die Möglichkeit des PPR (Partial Page Rendering) enthalten. Des weiteren bietet das Framework einige Möglichkeiten zur clientseitigen Validierung von Benutzerdaten und bemerkenswerterweise einen Mediaplayer.

3.1.4 JBoss RichFaces

Erweiterte Standardkomponenten, GUI, Effekte

JBoss RichFaces[8] ist ein sehr umfangreiches Framework – nicht zuletzt durch die Zusammenarbeit von Red Hat und Exadel Software seit Anfang 2007, wodurch die beiden Frameworks von Exadel, RichFaces und Ajax4JSF, zusammengeführt wurden. Seit dem steht die Komponentenbibliothek Open Source bei JBoss zur Verfügung.

Das Framework beinhaltet sowohl Standardkomponenten als auch andere, erweiterte GUI-Komponenten und Ajax-Funktionalität und arbeitet

[7] MyFaces Trinidad, http://myfaces.apache.org/trinidad
[8] JBoss RichFaces, http://labs.jboss.com/jbossrichfaces/

laut Dokumentation auch mit den MyFaces Frameworks – mit Ausnahme Tobago – zusammen. Mit RichFaces lassen sich sehr einfach Effekte realisieren, zudem beinhaltet das Framework einen Mediaplayer, eine GoogleMaps-Komponente u. v. m.

Der Clou an der Sache ist, dass die beiden integrierten Frameworks immer noch getrennt einsetzbar sind. Es ist ohne weiteres möglich, „nur“ RichFaces-Komponenten zu verwenden, oder auch Komponenten der Standardimplementierungen mit Ajax4JSF aufzuwerten, ohne auch nur eine Zeile Code umschreiben zu müssen.

Ajax4JSF[9] bietet auch eine einfache Anwendungsmöglichkeit für PPR, wie in Programm 3.2 dargestellt:

```
1 <f:view>
2   <h:form>
3     <h:panelGrid columns="2">
4
5       <h:outputText value="Type the Text:" />
6
7       <h:inputText value="#{bean.text}">
8         <!-- Die Komponente mit dem Namen "repeater" wird bei Eintritt
                des Events der Inputkomponente einfach aktualisiert.-->
9         <a4j:support event="onkeyup" reRender="repeater" />
10      </h:inputText>
11
12      <h:outputText value="Text in the Ajax response:" />
13
14      <!-- Es muss nicht unbedingt eine Komponente sein. Auch das <a4j:
            region /> Tag kann die ID "repeater" haben. Dann werden alle
            Komponenten aktualisiert, die sich innerhalb der Region "
            repeater" befinden. -->
15      <h:outputText id="repeater" value="#{bean.text}" />
16
17    </h:panelGrid>
18  </h:form>
19 </f:view>
```

Programm 3.2: Beispiel: Partial Page Rendering mit Ajax4JSF

Wer jedoch unbedingt JavaScript programmieren will bzw. muss, kann mit den verschiedensten JavaScript Frameworks arbeiten und auch Komponenten mit Hilfe des *Component Development Kit* (CDK) entwickeln.

Ein mittlerweile ausgelagertes Unterprojekt von Ajax4jsf ist G4jsf. Dieses Projekt bietet eine Zusammenführung der JSF-Technologie mit dem Google Web Toolkit[10]. Google-Widgets werden in JSF-Komponenten „verpackt“[11].

[9] Ajax4jsf, http://labs.jboss.com/jbossajax4jsf/
[10] GWT, http://code.google.com/webtoolkit
[11] G4jsf, https://ajax4jsf.dev.java.net/nonav/ajax/gwt/gwt-cdk.html

3.1.5 ICEfaces

Erweiterte Standardkomponenten, GUI, Effekte

Das Framework ICEfaces[12] ist ein quelloffenenes JSF-Framework mit extrem starkem Fokus auf die Ajax-Technologie und beinhaltet erweiterte Standardkomponenten sowie GUI Komponenten. Jede Komponente von ICEfaces ist standardmäßig ajaxifiziert - was heißen soll, dass Änderungen in (fast allen) Input-Komponenten automatisch zum Update der Backing Bean führen - und auch zur Anpassung des Outputs, der von dieser Beanproperty abhängig ist. Natürlich lässt sich diese Funktionalität, genannt *PartialSubmit*, auf Wunsch (via web.xml) ausschalten.

PartialSubmit beschreibt einen asynchronen Ajax-Request, der immer nur das aktuelle Formularfeld validiert und die entsprechenden Daten aktualisiert. Durch diese partielle Formularvalidierung lässt sich verhindern, dass eine Fehlermeldung für Inputfeld X geworfen wird, dessen Inhalt vom Feld Y abhängt, dessen Inhalt wiederum aber erst durch das Ausfüllen von Feld Z generiert wird.

Im Anschluss wird der DOM-Baum serverseitig verändert und nicht völlig neu an den Client gesendet. Dazu wird der DOM-Baum, durch den Request veranlasst, durch den *Direct-to-DOM Renderer*(D2D) neu erstellt und mit dem auf dem Server zwischengespeicherten „alten“ DOM-Baum verglichen. Die Differenzen werden anschließend an den Client geschickt und eingesetzt. Die Kommunikation zwischen Server und Client erfolgt nach dem erstmaligen Laden der Seite über die Ajax-Bridge, die den Web-Server über Benutzeraktionen informiert und für den Abgleich zwischen serverseitigem und browserseitigem DOM sorgt. ICEfaces ermöglicht zudem auch *Ajax Push* – serverbasiertes asynchrones Nachladen von Inhalt, was auch als *Reverse Ajax* bezeichnet wird.

Diese Ajax-Lastigkeit ist eine Schlüsselfunktion von ICEfaces. Da aber Verbindungen zum Server und zur Datenbank ein sehr bekanntes Nadelöhr in der Entwicklung von Webapplikationen darstellen, beinhaltet das ICEfaces-Framework ein eigenes Connection-Management, welches auf diesbezügliche Fehler entsprechend reagieren kann.

ICEfaces erkennt außerdem, ob im verwendeten Browser Javascript aktiviert ist - wenn nicht, wird der Benutzer auf eine in der web.xml zu konfigurierende Seite (errorpage) weitergeleitet. Natürlich ist auch das nicht das Gelbe vom Ei, aber im Gegensatz zu den meisten anderen Frameworks wird der Benutzer zumindest auf die Fehlfunktion hingewiesen.

Wie auch bei MyFaces Tomahawk beschrieben, unterstützt ICEfaces das Prinzip der *User-role Awareness* sowie die Deaktivierung von Input-Komponenten, zudem gilt das Framework als eines der sichersten seiner Sorte[13].

[12] ICEfaces, http://www.icefaces.org
[13] ICEfaces, http://de.wikipedia.org/wiki/ICEfaces

3.1.6 Frameworkübersicht

Die vier Frameworks in Tab. 3.1 zählen, gemessen an dem Inhalt ihrer Komponentenbibliotheken und deren Gestaltungsmöglichkeiten, zu den Reichhaltigsten ihrer Art. Sie beinhalten nicht nur teils sehr ausgereifte GUI-Elemente, sondern auch Validatoren, Converter und einfach zu realisierende Effekte. Alle zeichnen sich durch eine große, aktive Community aus, was zu viel Dynamik in der Entwicklung, aber auch manchmal zu etwas konfusen und nicht hundertprozentig aktuellen Dokumentationen führt.

	Tomahawk Sandbox	Trinidad	ICEfaces	Richfaces Ajax4jsf
Recherchedatum:	29.11.2007	3.12.2007	4.12.2007	4.12.2007
Lizenz:	Apache 2.0	Apache 2.0	MPL	LGPL
JSF-Version:	JSF 1.1	JSF 1.1 und höher	JSF 1.1	MyFaces 1.1 Sun 1.1 und höher
Kompatibilität:	Trinidad ICEfaces RichFaces Backbase	Tomahawk	Tomahawk	Tomahawk Trinidad
JavaScript:	optional	optional	optional	optional
JavaScript Frameworks:	Dojo	Dojo	Scriptaculous	Scriptaculous jQuery GMaps API
Dokumentation:	gut	gut	sehr gut	gut
Baumstruktur:	ja	ja	ja	ja
Toggling:	ja	ja	ja	ja
Paging:	ja	ja	ja	ja
Sorting:	ja	ja	ja	ja
Menü:	ja	ja	ja	ja
Tabs:	ja	ja	ja	ja
Popup:	ja	nein	ja	ja
Toolbar:	nein	ja	nein	ja
Breadcrumbs:	nein	ja	nein	nein
Kontextmenü:	nein	nein	nein	nein
Fileupload:	ja	ja	ja	nein
HTML-Editor:	ja	nein	nein	nein
Kalender:	ja	ja	ja	ja
Duale Liste:	ja	ja	nein	nein
Autocomplete:	ja	nein	ja	ja
Dialoge:	nein	ja	nein	nein
Drag/Drop:	nein	nein	ja	ja
Chart:	nein	ja	ja	nein
Skinning:	nein	ja	ja	ja

Tabelle 3.1: Die vier bekanntesten Open Source-Frameworks im Überblick

3.2 GUI-spezialisierte OS-Frameworks

3.2.1 MyFaces Tobago
GUI

MyFaces Tobago[14] ist ein weiteres Subprojekt von Apache MyFaces und unterscheidet sich völlig von den anderen MyFaces-Projekten, da es ausschließlich für die Entwicklung von GUI-Oberflächen optimiert ist. Das Framework ist so weit abstrahiert, dass nicht einmal HTML-Kenntnisse benötigt werden.

Was die Kompatibilität mit anderen Frameworks anbelangt, so muss bei der Auswahl beachtet werden, dass Tobago nur mit Frameworks kompatibel ist, die keinen eigenen Renderer benötigen[15], da dieses Framework einen speziellen Renderer benötigt. In JSF darf es aber pro Applikation nur ein Renderkit geben[16], daher die Einschränkung.

Ein Plus von Tobago ist die Möglichkeit des Skinnings. Die verfügbaren Themes müssen dazu nur in einem in der *tobago-config.xml* angegebenen Pfad gespeichert werden.

3.2.2 RCFaces
Erweiterte Standardkomponenten, GUI, Validatoren

Die Komponentenbibliothek Rich Client Faces[17] enthält erweiterte Standardkomponenten und einige GUI-Features. Bemerkenswerte Features sind zum Beispiel die clientseitigen Validierungsmöglichkeiten, die live – also während der Eingabe – funktionieren und beispielsweise Buchstabeneingabe in Zahlen- oder Datumsfeldern unmöglich machen. Es ist sogar möglich, einzelne, verbotene Zeichen zu definieren! RCFaces besitzt zwar keine eigene Chart Komponente, ist jedoch kompatibel mit der Software von JFree[18].

Leider ist die Dokumentation zu RCFaces nicht gerade im besten Zustand. Es war mir nicht möglich, Informationen über unterstützte Implementierungen, Kompatibilität usw. zu bekommen.

3.2.3 WebGalileo Faces
Erweiterte Standardkomponenten, GUI

Dieses Framework war ursprünglich nur kostenpflichtig erwerbbar, wurde aber Mitte 2007 von den an der Entwicklung beteiligten Unternehmen – Softaspects und Jscape – unter Apache License und Open Source gestellt. WebGalileo Faces[19] ist primär für die Referenzimplementierung von Sun op-

[14] MyFaces Tobago, http://myfaces.apache.org/tobago/

[15] In der Regel verträgt sich Tobago also nur mit Ajax Erweiterungen wie AjaxAnywhere.

[16] Es sei denn, die Framework-Entwickler berücksichtigen Kompatibilität zu bestimmten anderen Frameworks

[17] RCFaces, http://www.rcfaces.org/

[18] http://www.jfree.org/jfreechart

[19] WebGalileo Faces,
http://www.javawebcomponents.com/content/products/webGalileoFaces.html

timiert, funktioniert laut Dokumentation aber auch mit der MyFaces Implementierung – unter Verwendung eines Workarounds[20].

Das Framework bietet nur wenige Komponenten, deren Anpassungsmöglichkeiten sind allerdings extrem umfangreich. Zu den verfügbaren Komponenten zählen unter anderem ein umfangreicher HTML-Texteditor sowie zum Teil editierbare Chart-Komponenten.

Die Einbindung des Frameworks ist etwas ungewöhnlich, da jede Komponente bei Anwendung extra eingebunden wird - und das auf jeder einzelnen Seite die diese Komponente verwendet. Ein Vorteil ist jedoch, dass fast alle Komponenten zur Laufzeit editierbar sind, und auch dynamisch erzeugt werden können. Dieser Vorgang ist in der Dokumentation inklusive Sourcecode genau beschrieben.

3.2.4 Woodstock

Erweiterte Standardkomponenten, GUI

Bei Woodstock[21] handelt es sich um ein Open Source Framework, welches Standardkomponenten sowie einige interessante Komponenten wie z. B. Userunterstützung in Form von Wizards und Seitenheadern (*webuijsf:masthead*) beinhaltet. Die meisten Komponenten des Woodstock Frameworks beinhalten noch keine Ajax Unterstützung, diese wird jedoch laufend erweitert und überarbeitet, um die etwas träge Performance zu verbessern.

3.2.5 YUI4JSF

GUI, Effekte

Dieses Framework setzt auf der Yahoo! UserInterface Library auf. YUI beinhaltet bereits ausgefeiltere Komponenten (sogenannte Widgets), die mittels YUI4JSF[22] JSF-tauglich gemacht werden. Bestehende YUI-Funktionen (Utilities, wie zB. DragDrop Utils für Sortierbare Listen) werden verwendet, um neue Komponenten mit YUI-Funktionalität anzureichern.

Wie der Name schon sagt, setzt YUI4JSF den Schwerpunkt auf GUI Komponenten. Das Framework wird laufend erweitert, in Entwicklung befinden sich derzeit ein Templating-System für JSF-Seiten, Skinning für Komponenten - wie in YUI bereits integriert - sowie Ajax-Formulare.

3.2.6 Frameworkübersicht

Die fünf Open Source-Frameworks in Tab. 3.2 sind, gemessen an der Art der vorhandenen Komponenten, primär für die Gestaltung von GUI-Applikationen vorgesehen – Usabilitiy rangiert vor der Realisierung von tollen Effekten.

[20] WebGalileo Faces Doku,
http://www.javawebcomponents.com/content/documentation/wgf/html/index.html

[21] Woodstock, https://woodstock.dev.java.net

[22] YUI4JSF, http://yui4jsf.sourceforge.net

Die Bibliotheken enthalten (meist fast) alles, was zur ergonomischen Gestaltung einer modernen grafischen Benutzeroberfläche dazugehört - also Menüs, Reiter, Baumstrukturen usw.

	Tobago	RCFaces	Webgalileo Faces	Woodstock	YUI4JSF
Recherchedatum:	29.11.2007	05.12.2007	10.12.2007	11.12.2007	10.12.2007
Lizenz:	Apache 2.0	LGPL	Apache 2.0	CDDL	BSD
JSF-Version:	JSF 1.1	–	JSF 1.1	JSF 1.2	–
Kompatibilität:	–	–	–	–	–
JavaScript:	nein	nein	optional	optional	optional
JavaScript Frameworks:	–	–	GMaps API Prototype	Dojo	YUI API
Dokumentation:	gut	mäßig	sehr gut	gut	gut
Baumstruktur:	ja	ja	ja	ja	ja
Toggling:	nein	ja	ja	ja	ja
Paging:	ja	ja	ja	ja	ja
Sorting:	ja	ja	ja	ja	ja
Menü:	ja	ja	ja	ja	ja
Tabs:	ja	ja	ja	ja	ja
Popup:	ja	nein	nein	nein	ja
Toolbar:	ja	nein	ja	nein	nein
Breadcrumbs:	nein	nein	nein	ja	nein
Kontextmenü:	nein	ja	ja	nein	nein
Fileupload:	ja	nein	ja	ja	ja
HTML-Editor:	nein	nein	ja	nein	nein
Kalender:	ja	ja	ja	ja	ja
Duale Liste:	nein	nein	ja	ja	nein
Autocomplete:	ja	ja	nein	nein	ja
Dialoge:	nein	nein	ja	ja	ja
Drag/Drop:	nein	nein	ja	ja	ja
Chart:	nein	nein	ja	nein	nein
Skinning:	ja	nein	ja	ja	nein

Tabelle 3.2: Bekannte JSF-Frameworks mit Schwerpunkt GUI

3.3 Unspezialisierte OS-Komponentenframeworks

3.3.1 Blueprints JSF Components

Diverse Komponenten

Blueprints[23] als Ganzes betrachtet ist ein Projekt von Sun, das versucht, Richtlinien für die Entwicklung von Web 2.0 in Java mit Ajax, JSF und co zu etablieren - dazu gehört unter anderem auch eine „Referenzapplikation“ namens *Pet Store*. Dieser Abschnitt bezieht sich allerdings lediglich auf die JSF-Komponenten von Blueprints.

[23] Blueprints, https://blueprints.dev.java.net/ajaxcomponents.html

Die oben erwähnte Komponentensammlung als Framework zu bezeichnen ist allerdings übertrieben. Vielmehr handelt es sich um eine Art exemplarischer Darstellung, was mit der JSF-Spezifikation und Ajax machbar ist.

3.3.2 jenia4faces

Effekte, Diverse Komponenten

jenia4faces[24] ist ein italienisches Open Source-Projekt, initiiert von jenia Software. Es besteht bereits seit 2005 und bietet diverse Erweiterungen zu den Standardkomponenten. Diese werden nicht ersetzt, sondern durch hinzuzufügende jenia-Tags erweitert. Des weiteren bietet jenia Komponenten zur Integration der Google Ajax Search API, für Ajax-Effekte, Charts und einfaches Templating à la Tiles[25].

Bei der Verwendung von jenia gibt es zwei Möglichkeiten. Entweder wird das gesamte Softwarepaket installiert, oder nur die Teile, die im Projekt verwendet werden sollen. Dazu ist jenia in (derzeit sechs) eigenständige Komponenten"familien" unterteilt.

3.3.3 xulfaces

Erweiterte Standardkomponenten, GUI

XUL (*XML Userinterface Language* ist eine XML-basierte Markupsprache für GUI-Oberflächen. Seit Mitte 2005 wird auch eine JSF-Version entwickelt – xulfaces[26] – die auf JSF, XUL und Ajax basiert.

XUL wurde ursprünglich von und für die Mozilla Foundation und deren Projekte (Firefox, Thunderbird...) entwickelt, wird aber mittlerweile auch von anderen Programmen genutzt. Damit XUL funktioniert, wird jedoch die Gecko Engine benötigt. Daher können xulfaces im Internet Explorer nicht dargestellt werden, was die Bedeutung der Komponentenbibliothek gehörig einschränkt. Zudem ist keine vernünftige Demoapplikation verfügbar, daher wurde von einer ausführlicheren Analyse Abstand genommen.

3.3.4 Frameworkübersicht

Die Frameworks in Tab. 3.3 sind nicht eindeutig zu klassifizieren, da sie keinen eindeutigen Schwerpunkt vorweisen können oder – im Falle von xulfaces – aufgrund mangelnder Kompatibilität nicht näher untersucht wurden.

[24] jenia4faces, http://www.jenia.org
[25] Apache Tiles, http://tiles.apache.org
[26] xulfaces, http://xulfaces.sourceforge.net

	Blueprints	jenia4faces	xulfaces
Recherchedatum:	11.12.2007	12.12.2007	15.12.2007
Lizenz:	BSD	Apache 2.0	LGPL
JSF-Version:	Lib 1: JSF 1.1 Lib 2: JSF 1.2	JSF 1.1	JSF 1.2
Kompatibilität:	–	–	–
JavaScript:	optional	optional	–
JavaScript Frameworks:	Dojo GMaps API	GMaps API	–
Dokumentation:	mäßig	gut	–
Baumstruktur:	nein	nein	–
Toggling:	nein	nein	–
Paging:	nein	ja	–
Sorting:	nein	nein	–
Menü:	nein	nein	–
Tabs:	nein	nein	–
Popup:	ja	ja	–
Toolbar:	nein	nein	–
Breadcrumbs:	nein	nein	–
Kontextmenü:	nein	nein	–
Fileupload:	ja	nein	–
HTML-Editor:	nein	nein	–
Kalender:	ja	ja	–
Duale Liste:	nein	nein	–
Autocomplete:	ja	nein	–
Dialoge:	nein	nein	–
Drag/Drop:	nein	nein	–
Chart:	nein	ja	–
Skinning:	nein	nein	–

Tabelle 3.3: Weitere Open Source-Komponentenframeworks

3.4 Open Source-Ajax-Frameworks

3.4.1 AjaxAnywhere

AjaxAnywhere[27] ist ein Open Source-Framework, das reine Ajax-Funktionalität beinhaltet. Es besteht ausschließlich aus „nicht sichtbaren“, regionenbasierten Komponenten. Es kann – da es keinen eigenen Renderer benötigt – in der Regel mit beliebigen Komponenten-Frameworks kombiniert werden. Setzt das zu kombinierende Framework auch Ajax ein, ist jedoch Vorsicht geboten, da es so zu Namensraumkonflikten, Kollisionen bei der Nutzung des XMLHttpRequest-Objektes durch zwei Frameworks usw. kommen kann.

Die Idee des Frameworks basiert auf sogenannten *reloadable Zones*, d. h. in der JSF-/JSP-Seite werden aktualisierbare Bereiche definiert. Davon kann es natürlich auch mehrere pro Seite geben. Zudem ist es möglich, bei einem

[27] AjaxAnywhere, http://ajaxanywhere.sourceforge.net

Request festzulegen, welche Zonen einer Seite nach dem Abschicken aktualisiert werden – es muss also nicht zwingend die Zone sein, die den Request abfeuert.

Die Programmierung erfolgt entweder clientseitig mit der AjaxAnywhere Javascript API oder serverseitig mittels AjaxAnywhere API für Java. Die Datenübertragung erfolgt via XML.

3.4.2 J4Fry

J4Fry[28] ist ein relativ junges Projekt – und unterscheidet sich wesentlich von allen anderen bisher genannten.

Die Grundidee ist, dass jegliche Erweiterung des JSF-Standards immer möglichst mit diesem konform gehen soll. Die meisten der bisher vorgestellten Projekte verändern den Standard wesentlich und entwickeln dazu auch ein eigenes Set von Standardkomponenten, also sollten sie sich laufend an den sich ändernden JSF-Standard anpassen. Doch was, wenn die Entwickler dies nicht tun?

J4Fry geht einen anderen Weg: die Standard-Komponenten, das Tomahawk-Projekt sowie CSS-Stylesheets für das Design bilden die Basis – dadurch ist ein großer Teil an möglicher und erwünschter Funktionalität abgedeckt. Möchte der Entwickler nun zusätzliche Ajax-Features hinzufügen, bedient er sich – wie in Programm 3.3 dargestellt – einfach der Ajax-Komponente, mit der jede beliebige Ajax-Funktionalität hinzugefügt werden kann.

```
1  <j4fry:ajax event='onChange'
2              reRender='idOfTheRerenderedTag'
3              action='#{myBean.myAction}' />
```

Programm 3.3: J4Fry: Einsatz der Ajax-Komponente

Die Ajax-Komponente macht es auch möglich, JavaScript-Code zu definieren, der entweder vor oder nach dem Ajax-Request ausgeführt wird. J4Fry unterstützt auch „echtes“[29] PPR, somit ist das Rendern des gesamten Komponentenbaumes nicht mehr notwendig. Laut den Entwicklern ist zur Zeit auch ein Partial Page Submit-Lösung – siehe ICEfaces, *Partial Submit* – in Arbeit.

Diese Art der Entwicklung mag aufwändiger sein, zukünftige Inkompatibilitäten mit dem sich weiterentwickelnden JSF Standard sind jedoch weniger zu befürchten, da in erster Linie auf die Standardimplementierung gesetzt wird.

[28] J4Fry, http://www.j4fry.org

[29] Der Begriff des „echten“ PPR wird im nächsten Kapitel erläutert.

Das Projekt bietet auch optimiertes Errorhandling, ein Lookup Framework für die Implementierung der Mehrsprachigkeit, welches auf Hibernate und PostgreSQL basiert, eine optimierte *panel*-Komponente sowie ein Framework zum Testen der Responsezeiten.

3.4.3 jMaki

Das Framework jMaki[30] stellt im Wesentlichen einen Wrapper für beliebigen JavaScript-Code dar. Von Sun entwickelt, soll jMaki in erster Linie die Einbindung von JavaScript (-Frameworks) in JSP und JSF erleichtern. jMaki bietet bereits vorgefertigte Schnittstellen für Dojo, Prototype und YUI usw., sowie immer mehr eigene Widgets.

Oft ist es notwendig, verschiedene JavaScript-Frameworks einzubinden, um die gewünschte Funktionalität einer Seite zu erreichen – hier kommt es allerdings in der Praxis oft zu Inkompatibilitäten zwischen den verschiedenen Frameworks – und hier kommt jMaki ins Spiel. Das Framework kapselt die benötigten Ressourcen, d. h. JavaScript-, CSS- und HTML-Dateien zu einem Widget, dass sich mit gewohnter JSP- bzw. JSF-Syntax einbinden lässt. Die manuelle Einbindung von irgendwelchen JavaScript-Dateien entfällt komplett.

3.4.4 jsf-extensions

jsf-extensions[31] bzw. DynaFaces ist am ehesten mit Projekten wie Ajax4jsf zu vergleichen[32]. Das Projekt beinhaltet keine neuen Komponenten, sondern erweitert bestehende mittels aktualisierbarer Zonen um Ajax-Funktionalität.

Eine *ajaxZone* hat zwei wichtige Attribute: *execute* und *render*: Im Beispiel muss „zone1“ zwar nicht neu gerendert werden, aber es wird ein onChange Event ausgelöst – daher muss der neue Wert in die Backing Bean geschrieben werden. Dazu wird der „ausführbare“ Teil des JSF-Lifecycles durchlaufen[33], wie in Programm 3.4 gezeigt.

„zone2“ ist ein reines Ausgabemedium und wird mit dem aktualisierten Wert der Backing Bean neu gerendert. Diese Komponente muss nur den Renderteil des JSF-Lifecycles durchlaufen, wie in Programm 3.5 dargestellt.

[30] jMaki, https://ajax.dev.java.net/

[31] jsf-extensions, https://jsf-extensions.dev.java.net/

[32] jsf-extensions Forum,
https://jsf-extensions.dev.java.net/servlets/ReadMsg?list=dev&msgNo=1

[33] jsf-extensions Doku,
https://jsf-extensions.dev.java.net/mvn/tutorial.html

```
1  <jsfExt:ajaxZone id="zone1" execute="zone1" render="zone2">
2    <h:selectOneMenu id="country" value="#{ApplicationBean.country}"
3      valueChangeListener="#{ApplicationBean.updateCapital}">
4      <f:selectItems value="#{ApplicationBean.countries}"/>
5    </h:selectOneMenu>
6  </jsfExt:ajaxZone>
```

Programm 3.4: Die auslösende Komponente.

```
1  <jsfExt:ajaxZone id="zone2">
2    <h:outputtext id="capital" value="#{ApplicationBean.stateCapital}"/>
3  </jsfExt:ajaxZone>
```

Programm 3.5: Die zu rendernde Komponente

3.4.5 DWR

DWR[34] (*Direct Web Remoting*) ist ein Framework, welches den Zugriff von Java auf JavaScript und umgekehrt ermöglicht, d. h. die Kommunikation funktioniert vom Client zum Server und auch umgekehrt(*Reverse Ajax*).

Um DWR zu nutzen, werden in einer XML-Datei die jeweiligen Java Beans eingetragen – deren Methoden werden anschließend durch das Framework clientseitig zugänglich gemacht und der Zugriffscode erzeugt. Diesen fügt der Entwickler an beliebigen Stellen in einem *script*-Tag ein. Dies ist in den Demoapplikationen sehr gut veranschaulicht[35]. Mit speziellen Klassen kann zudem einfach auf die Werte von Komponenten zugegriffen werden. Die Requests und Responses werden durch ein eigenes DWR Servlet verwaltet.

DWR wurde zwar nicht speziell für JSF entwickelt, besitzt aber mittlerweile eine entsprechende Schnittstelle. Allerdings gilt es zu beachten, dass DWR nicht auf den JSF-Lifecycle eingeht, was unter Umständen zu Problemen führen kann.

3.4.6 Frameworkübersicht

Die Frameworks in Tab. 3.4 und Tab. 3.5 beinhalten keine bzw. sehr wenige Komponenten, und sind in erster Linie darauf ausgerichtet, statische Komponenten mit Ajax-Funktionen anzureichern. Das ebenfalls noch existierende Framework Ajax4jsf scheint hier nicht auf, da dieses mittlerweile in RichFaces integriert wurde.

[34] DWR, http://getahead.org/dwr/
[35] DWR Demo, http://getahead.org/dwr/examples/text

	AjaxAnywhere	J4Fry	jsf-extensions
Recherchedatum:	6.12.2007	12.12.2007	17.12.2007
Lizenz:	Apache 2.0	Apache 2.0	CDDL
JSF-Version:	–	JSF 1.1	JSF 1.2
JavaScript:	optional	optional(?)	optional
JavaScript Frameworks:	–	–	–
Dokumentation:	gut	gut	sehr gut

Tabelle 3.4: Frameworks mit klassischer Ajax-Funktionalität

	jMaki	G4JSF	DWR
Recherchedatum:	14.12.2007	12.12.2007	17.12.2007
Lizenz:	BSD	CDDL	Apache 2.0
JSF-Version:	–	–	–
JavaScript:	optional	optional	ja
JavaScript Frameworks:	beliebig	–	OpenAjax Alliance[36]
Dokumentation:	gut	–	gut

Tabelle 3.5: Frameworks mit erweiterter Ajax-Funktionalität bzw. Einbindung anderer Ressourcen

3.5 Bekannte kommerzielle Frameworks

3.5.1 NetAdvantage

Erweiterte Standardkomponenten, GUI, Validatoren

NetAdvantage[37] – ein Produkt des Unternehmens Infragistics – ist ein kommerzielles Framework, welches sich auf GUI-Komponenten konzentriert. Es beinhaltet wenige, dafür jedoch sehr ausgereifte Komponenten. Hier einige Beispiele:

- *ig:gridView*: Tabelle mit sämtlichen Attributen für Blättermöglichkeit, Resizing, Customizing, Sortierung, CSV Export inkludiert.
- *ig:tabView*: Tabs sind validierbar und können so bei fehlerhaften Eingaben blockiert werden.
- *ig:chart*: NetAdvantage beinhaltet eine Unmenge an Darstellungsformen für diverse Daten - 2D, 3D, verschiedene Lichtquellen...

Die Komponenten von NetAdvantage sind in sieben jar-Dateien gekapselt. Diese können – je nach Erfordernis – auch einzeln eingebunden werden. Das Framework bietet zudem noch integrierte Browsererkennung, Mehrsprachigkeit und Unterstützung bei der Programmierung eigener Komponenten.

[37] NetAdvantage, http://www.infragistics.com/java/netadvantage/jsf.aspx

3.5.2 QuipuKit

Erweiterte Standardkomponenten, GUI, Validatoren

Die einzelnen – wenn auch nicht eben zahlreichen – Komponenten des kommerziellen Frameworks QuipuKit[38] erscheinen sehr gut durchdacht und warten mit vielen standardmäßig eingebauten Anpassungsmöglichkeiten auf.

Standardmäßig ist Ajax bei allen Komponenten aktiviert, dies lässt sich jedoch für jede einzelne Komponente konfigurieren. Sofern für die Komponente relevant, ist es möglich, die Art des Datenbezuges zu bestimmen.

- *normaler Request*: Den Bereich immer neu laden.
- *Ajax Request*: Beim erstmaligen Laden eines bestimmten Inhaltes wird dieser mit Ajax vom Server geladen, anschließend bleibt der Inhalt clientseitig erhalten.
- *Stack*: Die gesamte Struktur wird beim initialen Seitenaufbau geladen, so dass die Inhalte nur vom Client ausgetauscht werden müssen.

Viele Komponenten lassen sich – sofern sinnvoll – mit Hilfe der Tastatur steuern. QuipuKit bietet ein eigenes Validierungs-Framework, welches sowohl server- als auch clientseitig einsetzbar ist.

Das Framework inkludiert auch eine eigene Javascript API, die für jede Komponente spezielle Methoden enthält, welche in der ausführlichen Dokumentation sehr genau beschrieben sind.

3.5.3 Backbase

Erweiterte Standardkomponenten, GUI, Effekte

Backbase[39] ist ein von dem gleichnamigen Unternehmen vertriebenes, kommerzielles JSF-Framework. Es beinhaltet sowohl erweiterte Standardkomponenten als auch ausgefeiltere eigene Komponenten und ist eigentlich eine Erweiterung des bereits bestehenden Ajax-Frameworks (*Enterprise Ajax*).

Das gesamte Framework wird von der Ajax-Funktionalität bestimmt. Es ist auch möglich, bestehende Komponenten mittels Javascript zu erweitern oder gar eigene Komponenten mit Hilfe der vorgesehenen Entwicklertools zu schreiben. Eine interessante Möglichkeit, sich als Entwickler zu betätigen, ist der Backbase Explorer [40]. Hier können die Demos nicht nur betrachtet, sondern auch an Ort und Stelle verändert, ausprobiert, kopiert und debuggt werden.

Ein weiteres Feature, das für Backbase spricht, ist die bereits erfolgte Integration von anderen JavaScript APIs, wie zum Beispiel GoogleMaps oder YUI. Außerdem wird – zwecks Validierung von Formularen – explizit auf die

[38] QuipuKit, http://www.teamdev.com/quipukit/index.jsf
[39] Backbase, http://www.backbase.com
[40] Backbase Explorer, http://demo.backbase.com/explorer/#|examples/welcome.xml

Kompatibilität mit MyFaces Tomahawk hingewiesen. Backbase bietet auch eine einfache Einbindung von Flash-Applikationen und Applets.

3.5.4 Simplica
GUI

Simplica[41] ist ein kommerzielles Framework, das auf GUI-Komponenten spezialisiert ist. Damit ist es möglich, Applikationen zu entwickeln, die sich von Desktopanwendungen kaum mehr unterscheiden – nicht einmal mehr im Aussehen. Die Komponenten beinhalten – wann immer sinnvoll – Drag and Drop, Kontextmenü usw. Ein weiteres bemerkenswertes Feature wird die unternehmenseigene Ajax-Engine, die demnächst an den Start geht[42].

3.5.5 Frameworkübersicht

Die Frameworks in Tab. 3.6 werden – abgesehen von diversen Trial- und Communityversionen – ausschließlich kommerziell vertrieben. Sie zeichnen sich durchwegs durch eine ausgereifte, vielfältig adaptierbare Komponentenbibliothek mit GUI-Schwerpunkt und sehr gutem Support bzw. sehr guter Dokumentation aus.

3.6 Weitere kommerzielle Frameworks

3.6.1 BindowsFaces
GUI

Bei BindowsFaces[43] handelt es sich um ein Ajax-lastiges, kommerziell vertriebenes GUI-Framework. Den Ursprung von BindowsFaces bildet das objektorientierte Enterprise-Ajax-Framework Bindows, auf welchem die Komponentenbibliothek nahtlos aufsetzt. Diese ist zwar an sich kostenlos, der Entwickler ist jedoch gezwungen, das zugehörige und nicht gerade billige Ajax-Framework zu erwerben.

Mit BindowsFaces können Applikationen entwickelt werden, die von einer Windows-Oberfläche kaum mehr zu unterscheiden sind und – im Gegensatz zu den meisten anderen JSF-Frameworks – zu fast hundert Prozent clientseitig funktionieren. Dadurch lässt sich die Serverlast wesentlich reduzieren. Um eine Bindows-Applikation zu schreiben, werden weder JavaScript- noch HTML-Kenntnisse benötigt. Natürlich steht es jedem Entwickler frei, sich in die API einzuarbeiten und so zusätzliche Funktionalität zu kreieren.

Leider gibt es für BindowsFaces noch keine Demos oder eine Dokumentation, da die Bibliothek erst als Betaversion verfügbar ist. Die Funktionalität

[41] Simplica, http://www.simplica.com
[42] Simplica Produktinfo, http://www.simplica.com/products.htm
[43] BindowsFaces, http://www.bindows.net/BindowsFaces

	Netadvantage	QuipuKit	Backbase	Simplica
Recherchedatum:	10.12.2007	11.12.2007	11.12.2007	12.12.2007
Kosten:	ab USD 795.–	ab USD 450.–	„please contact our sales team...“	ab USD 995.–
JSF-Version:	MyFaces 1.1.5 Sun 1.0 und höher	JSF 1.2	MyFaces 1.1.5 Sun 1.1	MyFaces 1.1 MyFaces 1.3 Sun 1.1
Kompatibilität:	–	–	Tomahawk	–
JavaScript:	optional	optional	optional	optional
JavaScript Frameworks:	–	–	Dojo GMaps API FusionChart YUI	beliebig
Dokumentation:	sehr gut	exzellent	exzellent	exzellent
Baumstruktur:	ja	ja	ja	ja
Toggling:	ja	ja	ja	nein
Paging:	ja	ja	ja	ja
Sorting:	ja	ja	ja	ja
Menü:	ja	nein	ja	ja
Tabs:	ja	ja	ja	ja
Popup:	nein	ja	ja	ja
Toolbar:	nein	nein	ja	nein
Breadcrumbs:	nein	nein	nein	nein
Kontextmenü:	nein	nein	ja	ja
Fileupload:	nein	nein	ja	nein
HTML-Editor:	nein	nein	nein	nein
Kalender:	ja	ja	ja	ja
Duale Liste:	nein	ja	ja	nein
Autocomplete:	nein	ja	ja	ja
Dialoge:	nein	ja	nein	nein
Drag/Drop:	nein	nein	ja	ja
Chart:	ja	ja	nein	nein
Skinning:	ja	ja	ja	ja

Tabelle 3.6: Die bekanntesten kommerziell vertriebenen Komponentenframeworks

ist, laut Ankündigung auf der Website, jedoch die selbe wie die des Ajax-Frameworks – daher sei zu Demonstrationszwecken auf dieses verwiesen[44].

3.6.2 ZK

GUI, Effekte

Das Open Source-Ajax-Toolkit ZK[45] ist ein noch relativ junges, serverseitiges und strikt komponentenbasiertes Ajax-Framework. Die Komponenten-

[44] Bindows Ajax-Demo, http://www.bindows.net/demos/

[45] ZK, http://www.zkoss.org

auswahl ist sehr reichhaltig und es können auch eigene Komponenten erstellt werden. Die JSF-Komponenten stellen eine Erweiterung des Produktes dar und wurden erst im November 2007 zum Download freigegeben.

ZK unterstützt PPR. Wird beispielsweise ein Wert in eine Tabelle eingefügt, wird der Eintrag via Ajax an den Server geschickt – die Response besteht aus einer Tabellenzeile mit eben jenem Eintrag, der in die bestehende View eingesetzt wird. Die Seite wird dadurch – gerade bei großen Datenmengen – entsprechend schneller aufgebaut. Auf der anderen Seite stellt sich natürlich die Frage, wie sich ZK-Applikationen unter Last verhalten, da das Framework die meiste Arbeit auf dem Server erledigt.

Im deutschsprachigen Raum ist ZK ein noch relativ unbeschriebenes Blatt – für weitere Informationen sei auf die ZK-Website sowie auf einen Artikel aus dem Java Magazin [4] verwiesen, welcher außerdem einen guten Vergleich von JSF und ZK (letzteres damals noch ohne JSF Komponenten) bietet.

3.6.3 JavaServer Faces Widget Library
GUI, Effekte

Die JavaServer Faces Widget Library[46], kurz JWL, ist Teil der neuesten Version des *Rational Web Developers* von IBM. Das Framework beinhaltet verschiedenste GUI-Komponenten und eine Ajax-Engine für clientseitig Features. Die Standard-JSF-Komponenten müssen nicht extra hinzugefügt werden, da diese bereits im Application-Server – Websphere – enthalten sind.

3.6.4 ILOG JSF Tools
Charts, Diagramme

Seit mittlerweile zwanzig Jahren entwickelt die international tätige Firma ILOG kommerzielle Software, die den Unternehmensalltag erleichtert. Dazu gehören Tools zur Prozessoptimierung und zur firmeninternen Administration ebenso wie die umfangreiche – und ebenso teure – Visualisierungssoftware mit dem Überbegriff JViews.

JViews[47] bietet die Möglichkeit, komplexe Firmenrohdaten schnell und einfach zu visualisieren – sowohl am Desktop als auch im Web. Wie der Name schon vermuten lässt, basiert JViews auf Java. Zu den Features zählen:

- *JViews Diagrammer* – komplexe Diagramme mit vielen verschiedenen Layouts erstellen.
- *JViews Maps* – Landkartendarstellung und -bearbeitung à la Google Maps/Earth.

[46] IBM, http://www.ibm.com
[47] ILOG JViews, http://www.ilog.com/products/jviews

- *JViews Gantt* – Prozess-/Projektvisualisierungssoftware mit Hilfe des Gantt-Diagrammes[48].
- *JViews Charts* – Charts (auch Echtzeit) in verschiedensten Layouts anlegen.
- *JViews Telecom Graphic Objects (TGO)* – Echtzeittool, das für Fuhrparkmanagement etc. eingesetzt werden kann.

Frameworkübersicht

Die Frameworks in Tab. 3.7 stellen entweder Erweiterungen zu klassischen Komponentenbibliotheken dar oder konnten mangels existierender Demo oder Dokumentation nicht genauer untersucht werden[49].

[48] Gantt-Diagramm, http://de.wikipedia.org/wiki/Gantt-Diagramm

[49] Im Fall von ZK ist zwar eine Demo der JavaScript-Variante vorhanden, jedoch funktionierte diese in keinem Browser korrekt (Opera, IE7, FF).

	BindowsFaces	ZK	JWL	ILOG
Recherchedatum:	14.12.2007	15.12.2007	14.1.2007	6.12.2007
Kosten:	ab USD 195.–	ab USD 99.–	ab USD 2.290.–	ab USD 2.000.–
JSF-Version:	–	JSF 1.1	–	–
Kompatibilität:	–	–	–	–
JavaScript:	optional	optional	optional	–
JavaScript Frameworks:	–	–	–	–
Dokumentation:	–	–	sehr gut	–
Baumstruktur:	–	ja	–	–
Toggling:	–	nein	–	–
Paging:	–	ja	–	–
Sorting:	–	nein	–	–
Menü:	–	ja	–	–
Tabs:	–	ja	–	–
Popup:	–	ja	–	–
Toolbar:	–	nein	–	–
Breadcrumbs:	–	nein	–	–
Kontextmenü:	–	ja	–	–
Fileupload:	–	ja	–	–
HTML-Editor:	–	ja	–	–
Kalender:	–	ja	–	–
Duale Liste:	–	nein	–	–
Autocomplete:	–	nein	–	–
Dialoge:	–	ja	–	–
Drag/Drop:	–	ja	–	–
Chart:	–	ja	–	ja
Skinning:	–	ja	–	–

Tabelle 3.7: Andere kommerziell vertriebene Komponentenframeworks

Kapitel 4

Fallstricke beim Einsatz von JSF-Frameworks

Viele tolle Komponenten zusammenzubauen macht noch keine gelungene Webapplikation. Es ist wichtig, sich über die Auswahl und Kombination der eingesetzten Techniken und Frameworks viel Gedanken zu machen, um möglichst viele potentielle Fehlerquellen auszuschließen. Die folgenden Abschnitte stellen einen Überblick über die bekannten Risikofaktoren dar.

4.1 Layout und Kompatibilität

4.1.1 Ajax-Frameworks einbinden

Mittlerweile verfügen quasi alle bekannten JSF-Frameworks über diverse Features, die mit Ajax angereichert wurden – was im Klartext heißt, dass bereits irgend eine Ajax-Library im Framework eingebunden ist. Was praktisch für den Entwickler ist, der nur Komponenten zusammensetzen möchte, kann für denjenigen, der selbst JavaScript schreiben möchte, schnell zur Quelle eines Ärgernisses werden, falls das inkludierte und das selbst gewählte Ajax-Framework zueinander nicht kompatibel sind. Es ist also wichtig, sich bereits im Vorfeld über mögliche Inkompatibilitäten zu erkundigen.

Die sicherste Möglichkeit ist in jedem Fall, die „eingebaute“ Ajax-Library zu verwenden. Dies kann eine bekannte Library wie beispielsweise Dojo sein, aber auch – und das gar nicht so selten – eine eigens programmierte Engine. Für letzteres wird oft auch die API zur Verfügung gestellt. Im Falle von Simplica kann durch einen eigenen Namespace zudem jede beliebige Ajax-Library in die Applikation zusätzlich eingebunden[1] werden. Eine interessante Möglichkeit, Ajax ohne Kompatibilitätsprobleme einzubinden, bietet auch das bereits vorgestellte Framework jMaki.

[1] Simplica Support, http://www.simplica.com/posts/list/44.page

Trinidad RichFaces ICEfaces

Abbildung 4.1: Der eher spartanisch ausgestattete Trinidad-Kalender im Vergleich zu den wesentlich ansprechenderen Varianten von RichFaces und ICEfaces.

4.1.2 Komponentenbibliotheken kombinieren

Besteht der Bedarf nach Kombination von zwei verschiedenen JSF-Frameworks – die, wie zuvor erwähnt, mittlerweile meist Ajax in irgendeiner Form integriert haben – muss ebenso auf die Kompatibilität untereinander geachtet werden. Dies trifft zum einen die eventuell verschiedenen integrierten Ajax-Frameworks, zum anderen die jeweiligen, frameworkspezifischen Renderkits – wovon es pro Applikation nur eines geben sollte. Ein Renderkit ist der Teil des Frameworks, der schlussendlich für die Darstellung der einzelnen Komponenten im Browser zuständig ist. Reine Ajax-Frameworks wie AjaxAnywhere besitzen in der Regel kein Renderkit, da sie bekanntlich keine sichtbaren Komponenten darstellen. Für diese gilt die zweite Einschränkung natürlich nicht.

Ein weiterer Aspekt, der bei der Kombination von Komponentenframeworks Kopfzerbrechen bereiten kann, ist das zum Teil ziemlich unterschiedliche Design, so dass unter Umständen viel Zeit in die Vereinheitlichung des Designs gesteckt werden muss. In Abb. 4.1 sind zum direkten Vergleich Kalender-Komponenten dargestellt, um die teilweise stark differierenden Designqualitäten zu veranschaulichen.

4.2 JSF-Standards

Die Geschichte der JavaServer Faces beginnt bereits im Jahr 2001, als durch den *Java-Community-Process*[2] die erste Spezifikation (*Java-Specification-Request,* kurz *JSR*) des Standards verabschiedet wurde. Die JSF-Spezifikation 1.0, basierend auf dem JSR-127 wurde schließlich im März 2004 veröffentlicht. Version 1.1 folgte nur kurz darauf. Die Version 1.2, definiert durch den JSR-252, startete im Mai 2006. Diese Version ist Bestandteil der *Java*

[2] JCP, http://www.jcp.org

Enterprise Edition[3]. Im Juni 2007 wurde schließlich der JSR-314[4] zur Definition der kommenden Version 2.0 der JavaServer Faces vorgestellt.

4.2.1 Potentielle Problemquellen in JSF-Frameworks

Anpassung an den JSF-Standard

Wie im vorangehenden Absatz ersichtlich, schreitet die Entwicklung der JSF-Spezifikation stetig voran – es stellt sich die Frage, ob die nicht unmittelbar Beteiligten – sprich die Implementierungs- und Framework-Entwickler – ebenso schnell nachziehen. Im Falle der Version JSF Spezifikation 1.2 veröffentlichte Sun fast sofort nach der offiziellen Vorstellung eine neue Implementierung. MyFaces hingegen brachte Version 1.2 erst im Juli 2007 heraus. Das wirkt sich natürlich auch auf die JSF-Frameworks aus. Bis dato unterstützen lediglich sieben der untersuchten Frameworks nachweislich zumindest eine JSF-Version 1.2 – und das, obwohl die Änderungen zur Vorgängerversion nicht sehr gravierend waren. Wie wird das erst aussehen, wenn der große Sprung auf JSF 2.0 ansteht?

Möchte der Entwickler, was neue Technologien anbelangt, immer auf dem neuesten Stand sein, beinhaltet gerade diese Thematik eine Quelle potentiellen Ärgernisses. Auch wenn die vorgestellten Frameworks dem Entwickler viel abnehmen, darf nicht vergessen werden, dass mit dem Nutzungsgrad eines Frameworks auch die Abhängigkeit von diesem steigt. Wurde eine Applikation erst einmal mit einem bestimmten Framework realisiert, gibt es im Falle einer Weiterentwicklung des Standards nur zwei Möglichkeiten: entweder abzuwarten oder auf ein anderes Framework umzusteigen, welches bereits auf dem neuesten Stand ist.

Im schlimmsten Falle werden die verwendeten Komponenten in einigen Jahren nicht mehr weiterentwickelt – was bei Standardkomponenten sehr unwahrscheinlich ist – im Gegenteil: kann davon ausgegangen werden, dass jegliches Upgrade des Standards abwärtskompatibel vorgenommen wird – oder zumindest ein Pattern zum Migrieren auf die neue Version zur Verfügung gestellt wird. Stellt der Entwickler also den Anspruch auf Aktualität der verwendeten Technologien, sollte er alles daran setzen, möglichst unabhängig von jeglichen Frameworks zu entwickeln. Genau dieser Gedanke wird von dem schon zuvor vorgestellten Framework J4Fry umgesetzt.

Kompatibilität zu JSF-Implementierungen

Wer schon einmal eine Dokumentation eines JSF-Frameworks gelesen hat, ist fast unweigerlich über diese Fragestellung gestolpert: „Auf welchen Application-Servern läuft Framework XY?“. Die Standardimplementierung – egal

[3] Java EE, http://java.sun.com/javaee/

[4] Java Specification Requests, http://jcp.org/en/jsr/detail?id=314

ob MyFaces oder Sun RI – läuft üblicherweise auf jedem Server. Bei JSF-Frameworks ist dies leider nicht selbstverständlich.

Das gleiche gilt für Erweiterungen des Standards. Der JSF-Standard unterstützt Portlets[5] – was bei weitem nicht für jedes JSF-Framework gilt. Selbiges trifft auf Facelets[6] zu.

Fehleranfälligkeit von JSF-Frameworks

Je größer die Verbreitung einer Technologie, desto mehr wird sie getestet – und desto weniger fehleranfällig ist sie in der Regel. Das gleiche gilt für JSF-Frameworks. Die Standardimplementierungen sind am weitesten verbreitet, da sie mit der jeweiligen Implementierung des Standards ausgeliefert werden. Erweiterte Komponentenbibliotheken, die sich natürlich den Markt mit allen anderen Mitstreitern teilen müssen, können nicht dieselbe Verbreitung erreichen.

Komponenten austauschen

JSF-Komponenten sind bekanntlich wiederverwendbar – aber sind sie auch „wiederwegwerfbar“? Das Ziel von allen Komponenten sollte eigentlich sein, dass sie dem Standard folgen und somit ohne größeren Aufwand gegen andere standardkonforme Komponenten austauschbar sind. Die Austauschbarkeit wird vor allem im Zusammenhang mit der JSF-Spezifikation 2.0 interessant, da in dieser die Einführung von vielen erweiterten Komponenten wie Baum, Kalender usw. in der Standardimplementierung geplant ist.

Und damit kommt wieder das Ausgangsproblem ins Spiel – nämlich dass viele Frameworks ihrem eigenen Muster folgen und somit nicht ohne gröbere Veränderungen im Code austauschbar sind, da sie mehr oder weniger ihren „eigenen“ Standard implementieren[7].

4.2.2 Ausblick: JSF 2.0

Wie bereits erwähnt, gibt es seit Mitte 2007 den JSR-314 zu JSF 2.0. Die Umsetzung wird noch einige Zeit in Anspruch nehmen und einige tiefgreifende Änderungen sowie neue Features mit sich bringen. Im folgenden werden einige wesentliche zu erwartende Neuerungen[8] der Spezifikation vorgestellt.

- *Ajax-Integration* durch Anpassung des Lifecycles.

[5] Portlet 2.0, http://www.jcp.org/en/jsr/detail?id=286

[6] Facelets, https://facelets.dev.java.net

[7] Ein großer Teil dieses Abschnittes basiert auf Informationen der Gründer von J4Fry, die als professionelle JSF-Entwickler diese Probleme zu spüren bekommen und daraus resultierend ihr eigenes (möglichst standardkonformes) Framework aufbauen.

[8] JSR-314, http://jcp.org/en/jsr/detail?id=314

- *Zero-Configuration*: XML-Konfigurationsdateien wie faces-config.xml und web.xml sollen abgeschafft werden.
- *Templating* im Sinne von Facelets usw. soll JSP ersetzen bzw. unterstützen.
- *Skinning*: austauschbare Stylesheets einbinden.
- *Erweiterte Komponenten*: Trees, Kalender usw. sollen auch in der Standardimplementierung verfügbar werden.

Bei genauer Betrachtung obiger Liste drängt sich allerdings eine Frage auf: wozu werden dann noch Frameworks für Standard-JSF benötigt? Beantworten lässt sich dies natürlich erst, wenn Version 2.0 der JavaServer Faces endgültig vorliegt.

4.3 Performance

Die vor allem zu Anfangszeiten nicht eben glänzende Performance von JSF-Anwendungen ist auf das sehr komplexe Konzept des Frameworks zurückzuführen. Die Performance wurde zwar mittlerweile verbessert, jedoch unterscheiden sich die diversen Referenzimplementierungen zum Teil stark in verschiedenen Kriterien. Andere Webframeworks, wie beispielsweise Apache Struts[9], schneiden nach wie vor performancemäßig besser ab [3].

Es gibt jedoch einige Faktoren, die der Entwickler bei einer JSF Webapplikation optimieren kann – ob dies möglich ist, hängt aber zum Teil auch von den weiteren verwendeten Frameworks ab. Eine Übersicht zu dieser Problematik geben die folgenden Abschnitte.

4.3.1 Serverseitige vs. clientseitige Statusspeicherung

Der Status einer JSF-Anwendung und seine Verwaltung[10] stellt das Herz jeder JSF-Applikation dar und ist einer der aufwändigsten Prozesse des Frameworks. Was ist aber der Status genau?

Jede JSF-View (also das, was der Benutzer zu Gesicht bekommt) besteht aus sichtbaren und unsichtbaren Komponenten. Diesen werden im Normalfall Eigenschaften des Models zugewiesen, z. B. darzustellende Texte oder Sichtbarkeitsinformationen. Zwischen zwei Anfragen an den Server müssen die Komponenten inklusive ihrer Eigenschaften gespeichert werden – und dies wird als Status einer JSF-Anwendung bezeichnet.

Dieser Status kann im Normalfall bei allen Frameworks entweder client- oder serverseitig gespeichert werden. Der Unterschied zwischen den beiden

[9] Struts, http://struts.apache.org
[10] Der Status wird durch den JSF-Lifecycle verwaltet

Varianten liegt im Speicherverbrauch und in der benötigten Netzwerkbandbreite. Serverseitige Speicherung benötigt viel Speicher am Server, während clientseitige Speicherung des Status in erster Linie zu steigendem Bandbreitenbedarf führt. Eine Untersuchung zu diesem Thema findet sich in [1].

4.3.2 Facelets steigern die Performance

Facelets stellen eine Alternative zur Darstellung der View dar. Anstatt JSP-Syntax zu verwenden, basieren Facelets auf einem XML-Dialekt. Sie können die Performance einer JSF-Applikation um etwa zehn Prozent steigern [1]. Allerdings werden Facelets noch nicht von allen JSF-Frameworks unterstützt. Einen Überblick gibt es auf jsfmatrix.net[11], genauere Details sind natürlich in der jeweiligen Dokumentation zu finden.

4.3.3 Unterschiede zwischen JSF-Implementierungen

Mit dem steigenden Bekanntheitsgrad von JavaServer Faces nimmt nicht nur die Anzahl an verfügbaren Frameworks rasant zu – auch mehr Implementierungen des JSF-Standards werden entwickelt. Noch vor kurzem beschränkte sich das Angebot auf die Referenzimplementierung von Sun – kurz Sun RI – und Apache MyFaces. Mittlerweile steigt auch hier die Auswahl.

Für den Entwickler ist nun wichtig, darauf zu achten, dass gewählte Frameworks und die Implementierung auch kompatibel sind. Für Apache MyFaces und Sun RI ist die jeweilige Kompatibilität in den Frameworktabellen angeführt.

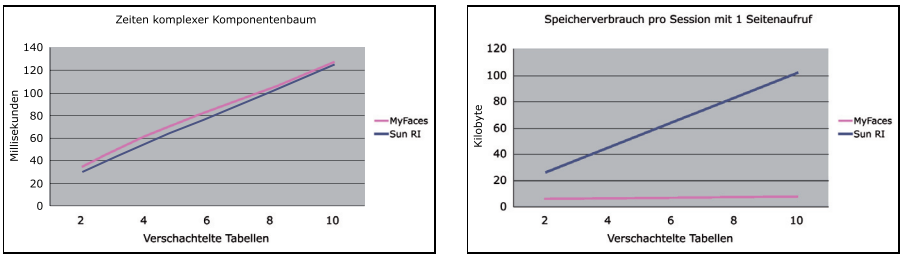
Sun RI vs. Apache MyFaces

Bei der Entscheidung, welche Implementierung verwendet werden soll, spielt auch die jeweilige Performance eine Rolle. Diese kann in Bezug auf Geschwindigkeit und Speicherverbrauch sehr unterschiedlich sein. Bezüglich Geschwindigkeit lässt sich beispielsweise zwischen MyFaces und Sun kaum ein Unterschied feststellen, während der Speicherverbrauch zugunsten MyFaces erheblich differiert, was auch in Abb. 4.2 dargestellt ist. Ob letzteres ein wichtiges Kriterium ist, muss jeder Entwickler für sich selbst entscheiden.

Auflistung bekannter JSF-Implementierungen

Wie bereits erwähnt, entstehen seit einiger Zeit auch weitere JSF-Implementierungen. Eine Auflistung der bekannten Implementierungen ist in Abb. 4.3 dargestellt.

[11] JSF-Übersicht, http://www.jsfmatrix.net/



(a) (b)

Abbildung 4.2: Sun RI vs. Apache MyFaces
Zeitliche Performance beim Abarbeiten eines komplexen Baumes (a), Speicherverbrauch des selbigen (b). Quelle: Java Magazin [1]

Implementierung	URL
Sun RI	http://java.sun.com/javaee/javaserverfaces/
Apache MyFaces	http://myfaces.apache.org/
Caucho Resin	http://www.caucho.com/press/2007-11-19.xtp

Abbildung 4.3: Auflistung von derzeit bekannten JSF-Implementationen – ohne Anspruch auf Vollständigkeit zu erheben!

4.3.4 Rendermethoden

Es wurde bereits erwähnt, dass das aufwändige Konzept der JavaServer Faces – die Statusverwaltung und Darstellung vor allem von komplexen Inhalten – die Performance erheblich bremst. Frameworks wollen nun, abgesehen von der Bereitstellung von ausgefeilten Komponenten, vor allem in diesem Bereich punkten. Die Zauberworte, die einem mittlerweile in fast jeder Framework-Präsentation ins Auge springen, heißen *Ajax* und *Partial Page Rendering*, kurz *PPR* – wobei natürlich letzteres ohne Ajax nicht möglich ist.

Was ist mit PPR meist gemeint?

PPR im Allgemeinen heißt, dass bei gewissen Benutzereingaben ein Ajax-Request an den Server geschickt wird und die daraus resultierende Antwort in vorher definierte Bereiche der schon angezeigten Seite eingearbeitet wird. Das Neuladen der Seite entfällt somit, und die Anwendung wirkt interaktiver und performanter.

Bei den meisten Ajax-JSF-Frameworks läuft der Vorgang folgendermaßen ab: der Ajax-Request wird abgesetzt, der JSF-Lifecycle ganz normal durchlaufen. Anschließend wird die neue Version des Komponentenbaumes mit der gespeicherten alten Version verglichen und die Differenzen werden an den

Client geschickt. Natürlich gibt es zur Implementierung dieses Prozesses verschiedenste Methoden, z. B. der *Direct-2-DOM-Renderer* von ICEfaces oder eine Realisierung mit JSF-Filtern und Prototype von Ajax4jsf. Das dahinter liegende Prinzip ist jedoch immer das Gleiche.

Was ist „echtes“ PPR?

Die meisten Frameworks, die PPR-Funktionalität beinhalten, haben also eines gemeinsam: sie durchlaufen den JSF-Lifecycle in voller Länge. Nun aber trägt die letzte Phase des Lifecycles bekanntlich den Namen *Render Response*. Rendern an sich ist also nicht die direkte Darstellung im Browser, sondern die Generierung des Seiteninhalts – was wiederum bedeutet, dass der Großteil der JSF-Ajax-Frameworks gar kein „echtes“ PPR implementiert!

Von allen untersuchten Frameworks implementiert lediglich eines nachweislich einen echten PPR-Ansatz – das junge Framework J4Fry. Bei diesem Ansatz wird ebenfalls ein normaler Ajax-Request abgesetzt und zu Beginn der JSF-Lifecycle ganz normal durchlaufen. Vor der Renderphase wird der Lifecycle jedoch mit einem *PhaseListener* gestoppt – das Model ist zu diesem Zeitpunkt ohnehin schon aktualisiert. Besagter PhaseListener ist auch für das Neurendern der in der Ajax-Komponente angegebenen darstellenden Komponenten zuständig.

Das J4Fry-Framework ist außerdem in der Lage, direkte Abhängigkeiten zu auslösenden Komponenten zu erkennen – so kann beispielsweise eine *tree*-Komponente (Baumdarstellung) so konfiguriert werden, dass bei einer Datenänderung nur die direkt voneinander abhängigen Baumknoten neu gerendert werden. Der Datenaustausch wird in beide Richtungen mit JSON vollzogen, die View wird mittels JavaScript aktualisiert. Da auf diese Weise Renderzeit gespart wird, ist diese Variante in der Regel schneller als traditionelles PPR[12].

[12] Rendermethoden, http://www.j4fry.org/jsfAjaxComparision.shtml

Kapitel 5

Schlussbemerkungen

„Ich suchte eine Oase und fand einen Urwald.“ Diese zugegebenermaßen pathetische Formulierung trifft vollkommen auf das Ergebnis meiner Recherchen zu. Zu Beginn ging ich davon aus, etwa zehn Frameworks genauestens zu evaluieren und anschließend in zwei bis drei Kategorien einzuteilen. Dem war allerdings nicht so – am Ende verfügte ich über eine stattliche Liste von 27 JSF-Ablegern[1], was sich natürlich auf den Detailliertheitsgrad der einzelnen Evaluierungen auswirkte. Zudem ist es unmöglich, in Bezug auf Aktualität der Informationen bei allen Frameworks völlig auf dem neuesten Stand zu bleiben, da sich in der JSF-Welt ständig etwas ändert.

Dass JSF im Moment sehr gefragt ist, ist unübersehbar. Ebenso eindeutig ist, dass Ajax dabei eine große Rolle spielt. Um letztere Aussage zu treffen, wird kein Fachartikel als Untermauerung benötigt – es genügt, sich auf den Webseiten der diversen Framework-Anbieter umzuschauen. Ständig erscheinen dort Nachrichten im Stil von „Komponente XY ist jetzt auch ajax-fähig“, zudem ist mir kein neu hinzugekommenes Framework bekannt, das nicht von Beginn an Ajax integriert. Auch umgekehrt funktioniert der Schluss: allein Ende 2007 haben zwei große Ajax-Framework-Anbieter – ZK und Bindows – die ersten Versionen ihrer JSF-Komponenten freigegeben, Backbase-Komponenten setzen auf der Backbase-Ajax-Engine auf, DWR implementierte eine JSF-Schnittstelle – um nur einige Beispiele zu nennen.

Zu Beginn der vorliegenden Arbeit habe ich erwähnt, dass die Möglichkeiten, welche die diversen JSF-Erweiterungen bieten sowie die Einfachheit der Umsetzung, bei mir große Begeisterung ausgelöst hatten. Nun, am Ende angelangt, muss ich gestehen, dass ich Frameworks, die sich allzu sehr von der Basis entfernen – und das nicht nur in Bezug auf JSF – nun mit gemischten Gefühlen betrachte. Einerseits fasziniert mich die Vielfältigkeit und die Einfachheit der Anwendung nach wie vor, andererseits teile ich die Bedenken in Bezug auf Fehleranfälligkeit und Unflexibilität, auch wenn mir selbst diesbezüglich noch die Praxiserfahrung fehlt.

[1] inklusive G4JSF und Ajax4jsf

Ich habe durch die Praxis, die ich im Sommer erworben habe, durchaus feststellen können, dass der Einstieg in JSF nicht gerade der einfachste ist. Dachte ich Ende des Sommers noch, ich sei schon eine recht gute JSF-Programmiererin, so bin ich durch die Recherchen zu dieser Arbeit ebenfalls schnell eines besseren belehrt worden. JSF ist viel mehr als nur Komponenten zusammenzusetzen – wenn der Entwickler sich wirklich damit beschäftigt. Die Erwähnung von *immediate*-Attributen, *PhaseListenern* usw. kommt nicht von ungefähr – diese Dinge haben in einem knapp gehaltenen Grundlagenkapitel eigentlich nichts verloren, sollen jedoch verdeutlichen, wie flexibel JSF ist, wenn es nur richtig gehandhabt wird. Ob diese Flexibilität mit diversen JSF-Frameworks auch noch so schön funktioniert, sei dahingestellt.

Anhang A

Erläuterungen zu den Frameworktabellen

A.1 Allgemeines zu den Tabellen

Die Frameworktabellen bieten eine Auswahl relevanter Kriterien, anhand derer Frameworks sinnvoll verglichen werden können. Im folgenden Abschnitt werden Kriterien bzw. Features näher erläutert, bei denen Erklärungsbedarf bestehen könnte.

A.1.1 Erklärung der angeführten Kriterien

Recherchedatum: Webtechnologie ist sehr schnelllebig, dieses Datum repräsentiert somit die letztmalige Recherche, da es unmöglich ist, in allem völlig auf dem neuesten Stand zu bleiben.

JSF-Version: Die laut Support kompatiblen Versionen der Referenzimplementierungen SUN und MyFaces.

Kompatibilität: Andere – laut Dokumentation oder Support – kompatible JSF-Frameworks.

JavaScript: Sind JavaScript Kenntnisse notwendig? Wenn nein – ist der Einsatz von JavaScript optional möglich?

JavaScript-Frameworks: JavaScript-Frameworks, die entweder bereits integriert oder kompatibel sind.

Dokumentation: Als Basis für die „Note“ dient der Aufwand, der benötigt wurde, um die gesuchten Informationen zu bekommen. Des weiteren wirkten sich fehlende bzw. nicht funktionierende Demo-Applikationen negativ aus.

Toggling: Bereiche ein- und aufklappen: Tabellen, Detailansichten usw.

Paging: Blättern in großen Datenmengen.

Sorting: Sortierfunktion in Listen, Tabellen usw.

Popup: Als Popup gelten sowohl „richtige“ Popups in einem eigenen Browserfenster als auch wegklickbare Bereiche.

Duale Liste: Zwei editierbare Bereiche, zwischen denen die enthaltenen Einträge mittels Buttons, Drag and Drop u. a. hin und her verschoben werden können.

Autocomplete: Automatische Vervollständigung bzw. Vorschläge in Eingabefeldern bei der Eingabe von Zeichen.

Skinning: Verschiedene Designs (Themes) stehen dem Benutzer der Anwendung zur Auswahl und sind beliebig umschaltbar.

Breadcrumbs: Klickbare Navigation, die den Pfad vom Wurzelelement zur aktuellen Seite anzeigt.

Dialoge: Jegliche Form von einfacher Kommunikation mit dem Benutzer: Alertboxen, kleiner Input-Dialog u. a.

Drag/Drop: Bereits implementierte bzw. einfach zu aktivierende Drag and Drop Funktionalität in Listen, Bäumen, Tabellen usw.

Kontextmenü: Menü, welches bei einem Rechtsklick mit der Maus erscheint.

Anhang B

Inhalt der CD

- PDF der Arbeit
- Textquellen aus dem Web, ausgenommen Frameworks
  - Ajax Definition: Jesse J. Garret
  - Vergleich der Ajax-Handhabung in wichtigen JSF-Frameworks: J4Fry
  - Lösungsmethoden für Ajax-Einbindung in JSF: Blueprints
  - Einstellung der Entwicklung von Netscape Navigator: Netscape
  - Beschreibung Gantt-Diagramm: Wikipedia
  - ICEFaces Sicherheit: Wikipedia
  - jsf-extensions Erklärung: jsf-extensions
  - jsf-extensions Ankündigung: jsf-extensions
  - Übersicht über viele JSF-Frameworks: JSFMatrix.net
  - JSF-Spezifikation 2.0: JCP
  - Simplica – Kompatibilität: Simplica
  - Tomcat Realm Definition: Apache
  - Webgalileo Faces – MyFaces: Webgalileo Faces Dokumentation
- Bilderquellen aus dem Web
  - Ajax Request: Google
  - Klassischer vs. Ajax-Request: Jesse J. Garret
  - JSF-Lifecycle: Sun

Die Links zu den Inhalten befinden sich sowohl in der jeweiligen Datei als auch an entsprechender Stelle in der vorliegenden Arbeit.

Literaturverzeichnis

[1] Marinschek, M., M. Geiler und G. Müllan: *Das Letzte herauskitzeln - Performante JSF-Anwendungen mit Apache MyFaces*. Java Magazin, 1:68–71, 2007.

[2] Müller, B.: *JavaServer Faces - Ein Arbeitsbuch für die Praxis*. Carl Hanser Verlag München Wien, 2006.

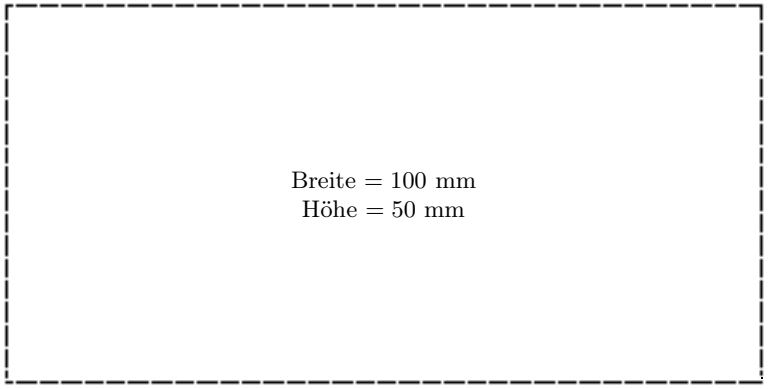
[3] Schlömmer, R. und M. Gessner: *Speed matters - JSF und Struts auf dem Leistungsprüfstand*. Java Magazin, 10:98–101, 2007.

[4] Seiler, D.: *ZK vs. Struts vs. JSF - ein Erfahrungsbericht*. Java Magazin, 10:85–97, 2007.

[5] Wenz, C.: *JavaScript & AJAX - Das umfassende Handbuch*. Galileo Press, Bonn, DE, 7. Aufl., 2007.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —