Fakultät Informationstechnik

Studiengang Softwaretechnik und Medieninformatik

Studienarbeit

Vergleichende Untersuchung von JavaServer Faces, AJAX und gängiger JSF-Erweiterungsframeworks

Simon Schneider

Sommersemester 2015

Erstprüfer: Prof. Dr. Joachim Goll

Zweitprüfer: Prof. Dr. Manfred Dausmann

Abkürzungsverzeichnis

ACE	ICEfaces Advanced Components
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSS	Cascading Style Sheet
CDI	Contexts and Dependency Injection
D2D	Direct-To-DOM
DOM	Document Object Model
EJB	Enterprise Java Beans
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICECORE	ICEfaces Core Components
JEE	Java Enterprise Edition
JSF	JavaServer Faces
JSP	JavaServer Pages
MVC	Model-View-Controller
RIA	Rich Internet Application
UI	User Interface
Unified-EL	Unified Expression Language
VDL	View Declaration Language
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensibel Markup Language

Inhaltsverzeichnis

1 JavaServer Faces 1
1.1 Einordnung 1
1.2 Aufgaben von JSF 2
1.3 Model 2-Architekturmuster 3
1.4 Verarbeitungszyklus von JSF 5
1.5 Aufbau von JSF-Anwendungen 9
1.5.1 Standard-JSF-Komponenten 9
1.5.2 View Declaration Language 12
1.5.3 Unified Expression Language 14
1.5.4 Managed Beans 15
2 AJAX 15
2.1 Einführung in AJAX 15
2.2 AJAX seit JSF 2.0 19
2.2.1 Das <f:ajax> Tag 20
2.2.2 JavaScript-API 22
2.2.3 Partieller JSF-Lebenszyklus 23
3 Erweiterungsframeworks 24
3.1 RichFaces 24
3.1.1 AJAX Action-Komponenten 25
3.1.2 AJAX Container-Komponenten 27
3.1.3 Partial Tree Processing und Partial View Rendering 28
3.1.4 Unterschiede zwischen JSF und RichFaces 28
3.2 ICEfaces 29
3.2.1 Vergleich zwischen Automatic AJAX und standardgemäßen JSF AJAX 29
3.2.2 Direct-To-DOM Rendering 33
3.2.3 Single Submit 34
3.2.4 ICEfaces-Komponenten 34
3.3 PrimeFaces 37
3.3.1 PrimeFaces-Komponenten 38
3.3.2 Partial Rendering und Partial Processing 44
3.3.3 Partial Submit 44
3.4 Vergleiche der Frameworks 44
3.4.1 Verfügbare Komponenten 44

3.4.2 Dokumentation 45
3.4.3 Kernfunktionalität 46
3.4.4 Performance 47
3.4.5 Lizenzen und Support 48
3.4.6 Trend 49
4 Zusammenfassung 50
5 Quellen 51
6 Abbildungsquellen 53

1 JavaServer Faces

1.1 Einordnung

JavaServer Faces (JSF) ist ein Framework zur Erstellung von serverseitigen Web-Anwendungen und ihren Benutzeroberflächen. JSF erleichtert die Entwicklung von Web-Anwendungen durch eine API und eine Tag-Library. Diese Tag-Library bietet eine große Anzahl an wiederverwendbaren Standard-UI-Komponenten an. Des Weiteren macht JSF den Datentransfer zwischen UI-Komponenten sehr einfach und ermöglicht es, selbst geschriebene Komponenten zu implementieren. Selbst geschriebene Komponenten sind dann sinnvoll, wenn ein Entwickler auf einen Anwendungsfall stößt, der nicht mit den Standard-Komponenten abgedeckt werden kann.

JSF basiert auf JavaServer Pages (JSP) und erweitert diese um das Model-View-Controller-Architekturmuster. Genauer gesagt verwendet JSF die Variante Model 2 des MVC-Musters. Diese Variante zeigt das folgende Bild:

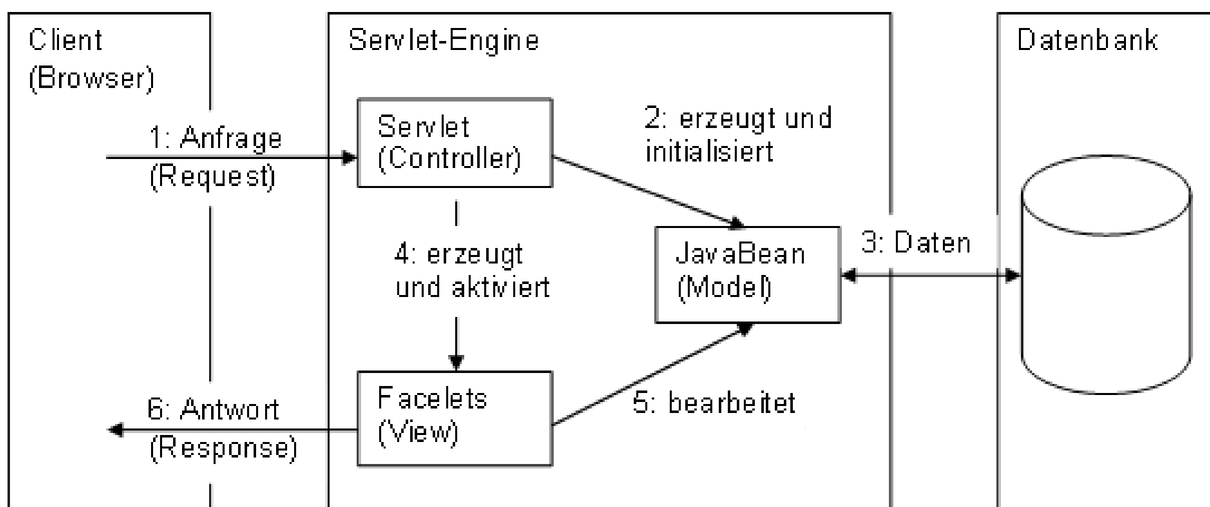

Abbildung 1: Model 2-Architektur im JSF-Framework

Web-Anwendungen werden wegen des MVC-Musters in drei Teile aufgeteilt. Diese drei Teile sind das Model, die View und der Controller. Diese drei Komponenten werden im Folgenden kurz erläutert:

- Die **View** beschreibt die grafische Repräsentation der darzustellenden Komponenten. Diese Komponenten werden durch JSF-Komponenten – welche Teil der Java Enterprise Edition (JEE) sind – beschrieben. Außerdem können diese Komponenten mithilfe von Cascading Style Sheets (CSS) oder durch Extensible Markup Language (XML) ergänzt werden. Die View informiert den Controller durch Ausführung von Aktionen und das Senden von Daten.
- Der **Controller** dient als „Requesthandler". Er hat die Aufgabe, das Model zu aktualisieren, Business-Logik auszuführen sowie die View zu aktualisieren. Als Controller kommt entweder ein Servlet vom Typ `FacesServlet` – eine Implementierung ist in der Java EE-API vorhanden – zum Einsatz oder wird in Form von Backing Beans (EJB[1], CDI[2]) selbst implementiert.
- Als **Model** werden JavaBeans (EJB) verwendet. Sie enthalten die Geschäftslogik und die Datenhaltung für die View.

Bei Verwendung des JSF-Frameworks muss ein Entwickler die benötigten Objekte nicht selbst erzeugen, sondern er gibt eine gewünschte Konfiguration vor, so dass der sogenannte Container die Objekte verwalten kann. Die Instanziierung und die Verwaltung der Model Objekte (JavaBeans) übernimmt dann der Container. Dieser Container ist fester Bestandteil des JSF-Frameworks. Die, vom Container verwalteten JavaBeans, nennt man auch Managed Beans. Eine JavaBean ist eine Java-Klasse, welche bestimmte Anforderungen erfüllen muss (siehe Kapitel 1.5.4 Managed Beans).

Beim JSF-Framework werden lose verknüpfte Komponenten eingesetzt. Dadurch wird ein hoher Grad an Wiederverwendbarkeit erreicht. Mittels dieser JSF-Komponenten lassen sich Seiten für Web-Anwendungen aufbauen. Diese JSF-Komponenten können dann vom Client-Browser gerendert und dargestellt werden.

Zusätzlich zu JSF gibt es mehrere Erweiterungsframeworks (Kapitel 3 Erweiterungs–frameworks), die den Funktionsumfang und auch das Angebot an Komponenten erweitern.

1.2 Aufgaben von JSF

Bei der Entwicklung von Web-Anwendungen befasst sich das JSF-Framework mit folgenden Aufgaben:

- **Bereitstellung von JSF-Komponenten**
  JSF stellt verschiedenste Komponenten zur Verfügung. Mithilfe dieser Kom–ponenten können Web-Oberflächen aufgebaut werden.

[1] EJB ist die Abkürzung für Enterprise JavaBean. EJBs sind im Grunde Plain Old Java Objects, welche mit Annotations versehen werden. EJBs können als Controller (Session-Bean), Entity-Bean (Model) oder als Message Driven Beans (asynchrone Kommunikation) eingesetzt werden.

[2] CDI ist die Abkürzung für Contexts and Dependency Injection

Wie bereits erwähnt, gibt es zahlreiche Erweiterungsframeworks. Zusätzlich können eigene Komponenten erstellt werden.

- **Speicherung des Zustandes einer Anwendung**
  Es können einzelne Komponenten als auch Anwendungsdaten gespeichert werden. Ein Beispiel hierfür ist der Zustand, ob ein Nutzer sich auf einer Seite noch anmelden muss, oder ob dies bereits erfolgt ist. Die Speicherung eines Zustandes kann serverseitig mit sogenannten Sitzungen (engl. sessions) oder auf Client-Seite mit Cookies erfolgen.

- **Datentransfer zwischen Client und Server**
  Das Http-Protokoll kann nur Daten in Form von Zeichenketten versenden. Aus diesem Grund übernimmt JSF die Konvertierung und Validierung der Anwendungsdaten, welche aus Eingabefeldern oder Web-Formularen ausgelesen werden. Die Konvertierung kann automatisch oder durch den Entwickler gesteuert ablaufen. Hierbei werden primitive- und erweiterte Datentypen sowie Datumsangaben unterstützt. Die Validierung prüft Daten auf syntaktische und semantische Eingabefehler.

- **Generierung von Ereignissen**
  Ereignisse werden von JSF-Komponenten ausgelöst, zum Beispiel durch Benutzeraktionen in der Web-Oberfläche. Die Ereignisbehandlung erfolgt in Methoden, welche zuvor an die Komponenten gebunden werden. Diese Ereignisse werden dann vom Server behandelt.

1.3 Model 2-Architekturmuster

JSF setzt die Variante Model 2 des MVC-Muster um. Dadurch gelingt JSF die Trennung zwischen Verhalten und Darstellung, genauer gesagt die Trennung zwischen Anwendungslogik und Benutzerschnittstelle. Es wird also das Prinzip Separation of Concerns umgesetzt. Der Vorteil des Architekturmusters besteht darin, dass zum Beispiel einzelne Komponenten ersetzt werden können, ohne andere Komponenten auch ersetzen zu müssen. Des Weiteren wird eine Vermischung von Codeteilen der Benutzeroberfläche und der Anwendungslogik vermieden. Somit kann an beiden Teilen parallel entwickelt werden.

Für eine Web-Anwendung ergeben sich vier operationell wichtige Schichten:

- Client-Schicht,
- Web-Schicht,
- Business-Schicht
- und Persistenz-Schicht.

Diese vier Schichten sind in Abbildung 2 zu sehen:

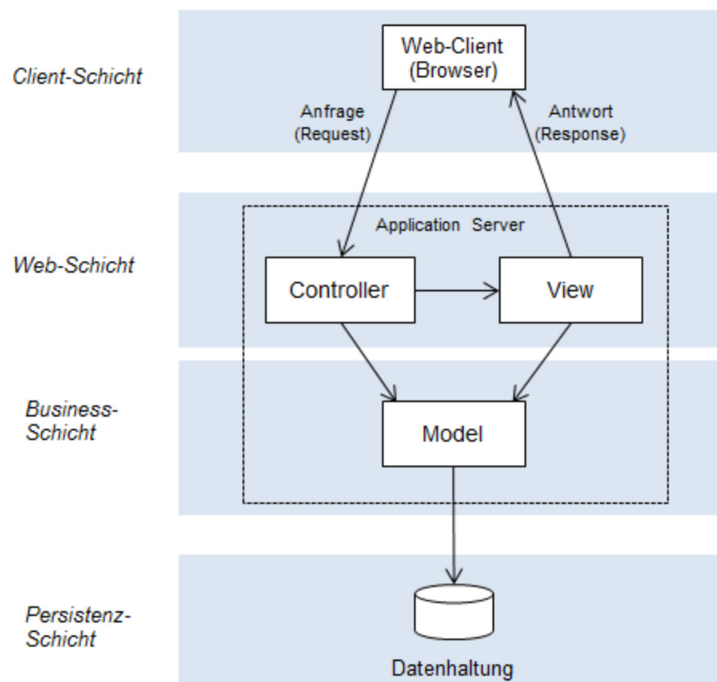

Abbildung 2: Model 2-Architektur bei JSF im Schichtenmodell

Das Entgegennehmen einer Anfrage eines Clients sowie die Steuerung von View und Model bestimmt der **Controller**. Zur Realisierung des Controllers wird eine Instanz der Klasse `FacesServlet` des JEE-Standards eingesetzt. Es wird für jede gestartete Web-Anwendung genau eine Instanz der Klasse `FacesServlet` erzeugt.

Dieses Servlet stellt ein Objekt vom Typ `FacesContext` zur Verfügung, welches Informationen über die aktuellen Anfragen hält.

Die Erstellung einer **View** als Ansicht einer Web-Anwendung wird mit der **View Declaration Language** realisiert. Der JSF-Standard ist hierfür Facelets. Facelets ist seit der JSF-Version 2.0 die Standard-VDL[3]. Damit werden die erforderlichen JSF-Komponenten deklariert. Dabei können ähnlich wie in HTML Attribute und weitere Eigenschaften wie Konverter und Validierer den JSF-Komponenten zugewiesen werden.

Die JavaBeans bilden in einer JSF-Anwendung das **Model** ab. Sie beinhalten die Daten und die Applikationslogik. Eine JavaBean kann dem JSF-Framework bekannt gemacht werden, sodass diese durch das JSF-Framework verwaltet werden kann. Werden JavaBeans durch das JSF-Framework verwaltet, nennt man sie ManagedBeans. Das Framework übernimmt dann die Instanziierung und die Lebensdauer dieser ManagedBeans.

Die Persistenz-Schicht – üblicherweise eine Datenbank – wird nicht durch JSF-Technologie abgedeckt.

[3] VDL ist die abkürzung für View Declaration Language

1.4 Verarbeitungszyklus von JSF

Um den Verarbeitungsprozess von JSF besser verstehen zu können, soll dieser näher betrachtet werden. Dieser Verarbeitungsprozess beginnt mit einem Http-Request eines Clients (Browser) und endet mit einer HTML-Seite als Antwort des Servers. Dieser Prozess wird in sechs Schritte unterteilt. Diese sechs Schritte zeigt das folgende Bild:

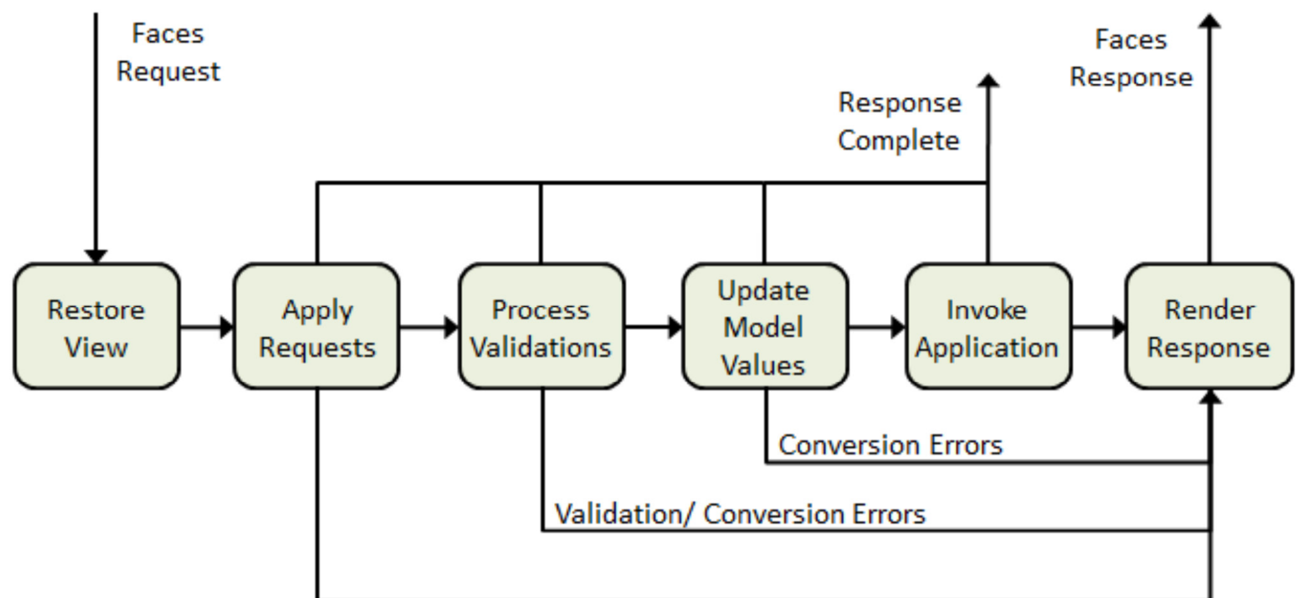

Abbildung 3: Verarbeitungszyklus von JSF

Eine Anfrage kann unterschiedlich viele dieser Schritte – je nach Anfrage und möglicher Verarbeitungsfehler – durchlaufen. Der JSF-Standard spezifiziert diese Schritte wie folgt:

- **Restore View:** Aufbau bzw. Aktualisierung des JSF-Komponentenbaums.
- **Apply Requests:** Übernahme der vom Anwender geänderten Werte aus der Http-Anfrage.
- **Process Validations:** Konvertierung und Validierung der Eingabewerte (Nicht leer, ist Zahl, etc.).
- **Update Model Values:** Aktualisierung – zum Beispiel durch das Aufrufen der Setter-Methoden – der Model-Objekte (Backing Beans) mit validierten Daten.
- **Invoke Applikation:** Aufruf der Businesslogik der Backing Beans.
- **Render Response:** Erzeugung der HTML-Antwort auf die Http-Anfrage und Übermittlung an den Browser.

Im Folgenden sollen diese sechs Schritte des JSF-Verarbeitungszyklus, genauer erklärt werden:

- **Restore View**
  Bei diesem Schritt wird vom JSF-Framework unterschieden zwischen einer erstmalig eintreffenden Anfrage und den darauf folgenden Anfragen an dieselbe View.

  Beim erstmaligen Eintreffen einer Anfrage wird zuerst der JSF-Komponenten–baum aufgebaut. Dies geschieht mithilfe einer Seitendeklarations–sprache (engl. View Deklaration Language, VDL). Im Falle der Seitendeklarationssprache Facelets wird der Komponentenbaum, mithilfe eines Parsers, aus einer XHTML-Datei aufgebaut. Die dabei ausgelesenen XHTML-Tags werden auf Objekte im JSF-Komponentenbaum abgebildet. Bei der ersten Anfrage wird somit der Initialzustand des Komponentenbaums aufgebaut.

  Bei folgenden Anfragen an dieselbe View werden zuvor gespeicherte Statusinformationen dazu verwendet, den letzten Zustand der View aus dem Initialzustand wiederherzustellen. Durch Änderungen der View – zum Beispiel durch eine Benutzereingabe – wird der zuvor erstellt Baum aktualisiert.

- **Apply Requests**
  In diesem Schritt werden die Benutzereingaben in den JSF-Komponentenbaum übernommen. Dabei holen sich die JSF-Komponenten die eingegebenen Werte aus der Http-Anfrage. Die hierbei ausgelesenen Werte werden noch nicht in die Objekte des Models, übernommen sondern werden zuerst noch in den JSF-Komponenten, des Komponentenbaums, gespeichert. Der Grund hierfür ist, dass die Eingabewerte noch nicht konvertiert und validiert wurden und es somit noch zu einem Fehlerfall führen könnte. Die Konvertierung und die Validierung erfolgt im nächsten Schritt.

- **Process Validations**
  Wie bereits erwähnt, werden in diesem Schritt die aus der Http-Anfrage entnommenen Werte konvertiert und validiert. Eine Konvertierung der Daten wird auf jeden Fall benötigt, da die Daten der Http-Anfrage als Zeichenkette vorliegen und in Java-Datentypen umgewandelt werden müssen. Eine Validierung der Benutzereingabe ist optional. Sie wird verwendet, um zu prüfen, ob eine Eingabe des Benutzers gültig ist oder nicht.

  Das JSF-Framework bietet einige Standard-Konverter und -Validierer an. Es ist jedoch möglich, benutzerdefinierte Konverter und Validierer selbst zu implementieren.

Nachdem ein Wert konvertiert und – falls erforderlich validiert – wurde, wird dieser dem Attribut `value` einer JSF-Komponente zugewiesen. Ändert sich nun der Wert des Attributs `value` der JSF-Komponente, so wird für diese Komponente ein Ereignis vom Typ `ValueChangeEvent` generiert. Dieses Ereignis wird für die darauffolgende Verarbeitung der Ereignisse an einen Event-Listener angemeldet.

- **Update Model Values**
  Nachdem die Daten im vorherigen Schritt konvertiert und ggf. validiert wurden, können nun die Daten in die ManagedBean-Objekte des Models geschrieben werden. Dies wird mithilfe der Setter-Methoden der ManagedBeans realisiert.

- **Invoke Application**
  In diesem Schritt erfolgt der Aufruf der Businesslogik. Dabei gibt es verschiedene Möglichkeiten, den Aufruf zu realisieren. Die erste Möglichkeit dabei ist es, das Attribut `action` einer zugehörigen JSF-Komponente mit einer Action-Methode einer ManagedBean zu verknüpfen, um ein Ereignis zu behandeln. Die Action-Methode führt die implementierte Anwendungslogik aus. Eine weitere Möglichkeit ist es, das Attribut `actionListener` mit einem ActionListener zu verknüpfen. Dieser bearbeitet dann ein vom Anwender ausgelöstes Ereignis.

- **Render Response**
  In diesem Schritt wird der JSF-Komponentenbaum gerendert. Dies bedeutet, dass er in eine für den Webbrowser lesbare HTML-Form gebracht wird. Diese HTML-Form bildet somit die Antwort auf die Http-Anfrage und wird an den Anwender zurückgeschickt. Je nachdem, wie viele Schritte im Verarbeitungs–zyklus durchlaufen wurden, kann die HTML-Antwort auch Fehlermeldungen enthalten, falls bei der Verarbeitung Fehler aufgetreten sind. Außerdem kommen in diesem Schritt auch die Konverter zum Einsatz, da die als Java-Typen vorliegenden Daten in Textzeichen zurückgewandelt werden müssen. Mit Abschluss des Renderns, zur Erzeugung einer darstellungsfähigen Ausgabe, endet der Verarbeitungszyklus einer Anfrage und die Antwort kann an den Client geschickt werden.

Wie bereits erwähnt, werden nicht immer alle dieser sechs Verarbeitungsschritte durchlaufen. Dies kann verschiedene Gründe haben, wobei nicht immer ein Fehlerfall sein aufgetreten sein muss. Es werden hier nicht alle dieser „Spezialfälle" im Detail besprochen, sondern im Folgenden nur kurz erwähnt:

- **Initiale Anfrage**
  Die initiale Anfrage ist die erste Anfrage an eine Webseite. In diesem Fall besteht noch kein JSF-Komponentenbaum. Dies bedeutet, dass dieser zuerst aufgebaut werden muss. Dabei werden nur die Phasen `Restore View` und `Render Response` durchlaufen, um den Initialzustand der View anzufordern. Alle übrigen Schritte entfallen dabei, da es noch keine gar Benutzereingaben gibt.

Demnach gibt es auch keine Konvertierung oder Validierung von Eingabewerten und auch keine Ausführung von Geschäftslogik.

- **Fehler bei der Konvertierung/Validierung**
  Tritt im Schritt `Process Validations` ein Fehler bei der Konvertierung oder der Validierung auf, so wird eine Fehlermeldung generiert. Die Verarbeitungs–schritte `Update Model Values` und `Invoke Application` werden übersprungen und es wird der Schritt `Render Response` ausgeführt. Es werden also keine Daten in die Model-Objekte geschrieben und es wird auch keine Businesslogik ausgeführt. Die Antwort, die der Client erhält, ist dieselbe Seite nur mit der zusätzlich generierten Fehlermeldung.

- **Einfluss auf den Verarbeitungszyklus**
  Um selbst Einfluss auf den Verarbeitungszyklus nehmen zu können, kann das Attribut `immediate` auf `true` gesetzt werden. Dadurch verändert sich der Verarbeitungszyklus je nach Typ der JSF-Komponente. Es wird dabei zwischen Eingabekomponenten wie zum Beispiel `<h:inputText>` und Steuerkom–ponenten wie zum Beispiel `<h:commandButton>` unterschieden.

  Setzt man dieses Attribut in einer Eingabekomponente auf `true`, so bewirkt dies, dass die Konvertierung und Validierung bereits während der Phase `Apply Requests` stattfindet. Als Beispiel könnte man hier eine Checkbox in einem Web-Formular nehmen. Werte von Eingabefeldern werden nur dann übernommen und in die Model-Objekte gespeichert, wenn diese Checkbox ausgewählt ist. Eine fehlende Betätigung der Checkbox würde den Verarbeitungszyklus nach der Phase `Apply Requests` beenden und zur letzten View zurückkehren, ohne die Werte aus den Eingabefeldern zu übernehmen.

  Wird das Attribut `immediate` bei einer Steuerkomponente auf `true` gesetzt, so bewirkt dies, dass die Phase `Invoke Application` vorgezogen wird. Dies bedeutet, dass die Geschäftslogik bereits in der Phase `Apply Requests` ausgeführt wird. Ein Beispiel wäre eine Schaltfläche zum Abbrechen, die auf einer Seite mit Pflichtfeldern genutzt wird. Solange diese Eingabefelder leer sind, wird die Validierung fehlschlagen und der Benutzer kann dann diese Seite nicht verlassen. Indem die Geschäftslogik vorgezogen und die Validierung übersprungen wird, lassen sich solche Seiten auch bei leeren Pflichtfeldern verlassen.

1.5 Aufbau von JSF-Anwendungen

In diesem Kapitel soll der Aufbau einer JSF-Anwendung beschrieben werden. Zunächst wird auf die Standard-JSF-Komponenten eingegangen, welche zur Erstellung einer View verwendet werden. Anschließend wird beschrieben, wie eine Webseite mit der Seitendeklarationssprache Facelets zu erstellen ist. Zum Schluss wird noch auf die Verbindung zwischen View und Model eingegangen. Diese Verbindung wird mithilfe der Unified-EL[4] realisiert.

1.5.1 Standard-JSF-Komponenten

Die Standard-JSF-Komponenten werden zur Erstellung von Benutzeroberflächen verwendet. Diese Komponenten werden in Form ihrer Tags in XHTML-Dokumenten deklariert.

Die sich dadurch ergebende Baumstruktur wird zur Laufzeit der Webanwendung im JSF-Komponentenbaum abgebildet. Die einzelnen Elemente des JSF-Kompo–nentenbaums sind Instanzen von Java-Klassen.

Das Verhalten von Komponenten wird in JSF-Komponentenklassen definiert. Dabei erben alle Komponenten indirekt oder direkt von der abstrakten Basisklasse `UIComponentBase`. Diese Vererbungshierarchie wird im folgenden Bild dargestellt:

[4] Unified-EL ist die Abkürzung für Unified Expression Language. Die Unified-EL realisiert die Verbindung zu Eigenschaften und Methoden einer Managed Bean.

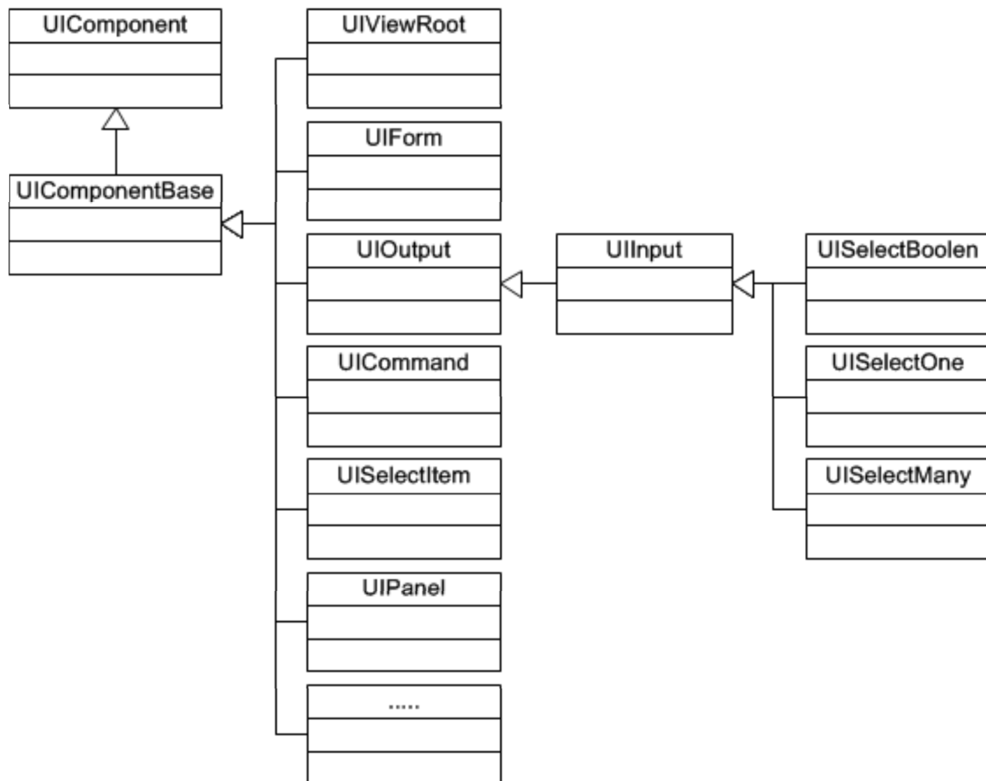

Abbildung 4: Vererbungshierarchie von JSF-Komponenten

Im Folgenden werden einige dieser Standard-JSF-Komponenten dargestellt und erklärt:

- **UIForm**
  Diese Komponente wird durch das Tag `<h:form>` deklariert und entspricht dem HTML-Tag `<form>`. Diese Komponente wird nicht grafisch dargestellt und umschließt lediglich weitere Tags, welche für Benutzereingaben und –aktionen gedacht sind und somit Daten an die Web-Anwendung leiten.

- **UIOutput**
  Diese Komponente dient zur Darstellung von Text. Sie kann mit folgenden Tags deklariert werden:

Tag-Name	Beschreibung
`<h:outputText>`	Unformatierter Text
`<h:outputLabel>`	Zur Beschreibung von GUI-Elementen
`<h:outputLink>`	Text für Hyperlinks
`<h:outputFormat>`	Formatierter Text

Tabelle 1: Tags der UIOutput-Komponente

- **UIInput**
  Mithilfe dieser Komponente können Eingaben des Benutzers realisiert werden. Diese Komponente kann ebenfalls mit verschiedenen Tags deklariert werden. Diese Tags werden in folgender Tabelle dargestellt:

Tag-Name	Beschreibung
`<h:inputText>`	Einzeiliges Texteingabefeld
`<h:inputTextarea>`	Mehrzeiliges Texteingabefeld
`<h:inputSecret>`	Einzeiliges Texteingabefeld, bei dem die Eingabe durch Platzhalter verschleiert wird. Dient zur Eingabe von Passwörtern.
`<h:inputHidden>`	Wird nicht als sichtbares Element gerendert

Tabelle 2: Tags der UIInput-Komponente

- **UICommand**
  Diese Komponente stellt ein Steuerelement dar. Dieses Steuerelement kann vom Benutzer aktiviert werden, wodurch ein Ereignis ausgelöst wird. Diese Kom–ponente muss sich im Tag `<h:form>` befinden bzw. von diesem Tag um–schlossen sein. Auch für diese Komponente gibt es verschiedene Tags. Diese werden in Tabelle 3 dargestellt:

Tag-Name	Beschreibung
`<h:commandButton>`	Erzeugt eine Schaltfläche (engl. Button)
`<h:commandLink>`	Erzeugt einen Hyperlink

Tabelle 3: Tags der UICommand-Komponente

- **UISelectBoolean**
  Mit dieser Komponente ist es möglich, einen Wahrheitswert darzustellen. Sie wird mit dem Tag `<h:selectBooleanCheckbox>` deklariert. Durch das Rendern ergibt sich eine Checkbox, die die Werte `wahr` oder `falsch` annehmen kann.

- **UISelectOne**
  Damit kann eine Auswahl von genau einem Element aus einer Gruppe von Elementen realisiert werden. Sie kann durch folgende Tags realisiert werden:

Tag-Name	Beschreibung
`<h:selectOneRadio>`	Erzeugt ein Optionsfeld (Radiobutton)
`<h:selectOneListbox>`	Erzeugt ein Listenfeld
`<h:selectOneMenu>`	Erzeugt Auswahlmenü (Combo-Box)

Tabelle 4: Tags der UISelectOne-Komponente

- **UISelectMany**
  Diese Komponente ähnelt der Komponente `UISelectOne`. Der Unterschied liegt darin, dass eine Auswahl mehrerer Elemente möglich ist. Sie kann durch folgende Tags realisiert werden:

Tag-Name	Beschreibung
`<h:selectManyCheckbox>`	Gruppe von Checkboxen
`<h:selectManyListbox>`	Erzeugt ein Listenfeld
`<h:selectManyMenu>`	Erzeugt Auswahlmenü

Tabelle 5: Tags der UISelectMany-Komponente

- **UIData und UIColumn**
  Damit lassen sich Tabellen mit Zeilen und Spalten zur Laufzeit erzeugen. Die dafür benötigten Tags sind `<h:dataTable>` und `<h:column>`. Dabei enthält die Komponente `UIData` typischerweise mehrere Komponenten `UIColumn`. Die Komponente `UIColumn` wiederum repräsentiert eine Spalte der Tabelle, in der sich mehrere Zeilen mit Text oder weitere JSF-Komponenten befinden können.

- **UIGraphic**
  Diese Komponente dient der Darstellung von Bildern und wird mit dem Tag `<h:graphicImage>` deklariert.

1.5.2 View Declaration Language

Die View Declaration Language (VDL) dient zur Deklaration einer View und beschreibt somit den Aufbau einer Webseite. Für JSF-Anwendungen ist Facelets die standardmäßige VDL.

Facelets wurde speziell für den Einsatz mit JSF entwickelt. Dadurch fügen sie sich perfekt in den JSF-Verarbeitungszyklus ein. Außerdem ist durch Facelets eine saubere Trennung zwischen der View, dem Model und dem Controller möglich.

Des Weiteren bietet Facelets die Möglichkeit mit Templates zu arbeiten, was den Grad der Wiederverwendbarkeit erhöht.

Eine View wird via Facelets durch XHTML deklariert. Hierfür bringt JSF eine Standard-Bibliothek mit. Zusätzlich können Tag-Bibliotheken von Drittanbietern und herkömmliche HTML-Tags verwendet werden.

Die JSF Standard-Tag-Bibliothek enthält zwei wichtige Tag-Gruppen. Sie müssen in der XHTML-Datei deklariert werden, um die JSF-Tags verwenden zu können. Die Deklaration der Tags sieht folgendermaßen aus:

```
<html xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"/>
```

Diese zwei Tag-Gruppen sind die folgenden Tag-Gruppen:

- **HTML-Custom-Tag-Library**
  Standardmäßig wird diese Tag-Bibliothek mit dem Namespace `h` in eine XHTML-Datei eingebunden. Diese Tags deklarieren Elemente der View und werden durch ein entsprechendes Objekt im JSF-Komponentenbaum repräsentiert. Diese Tags werden gerendert, damit eine HTML-Ausgabe erfolgen kann.

- **Core-Tag-Library**
  Diese Tags dienen zur Konfiguration der Darstellung. Damit können Platzhalter gesetzt und Validierer konfiguriert werden. Diese Tags werden beim Rendern nicht beachtet und mit dem Namespace `f` in eine XHTML-Datei eingebunden. Das folgende Beispiel zeigt die Verwendung eines Core-Tags mit dem Namespace f.

  Beispiel:

```
<h:inputText label="Password" id="password"
   value="#{userBean.password}" >
   <f:validateLength minimum="8" maximum="16" />
</h:inputText>
```

  In diesem Beispiel wird mithilfe des Tags `<f:validateLength>` die Eingabe des Benutzers auf eine bestimmte Länge überprüft. Ein gültiger String muss mindestens acht und darf maximal 16 Zeichen lang sein.

  Das folgende Bild zeigt die Eingliederung von Facelets in den JSF-Verarbeitungszyklus:

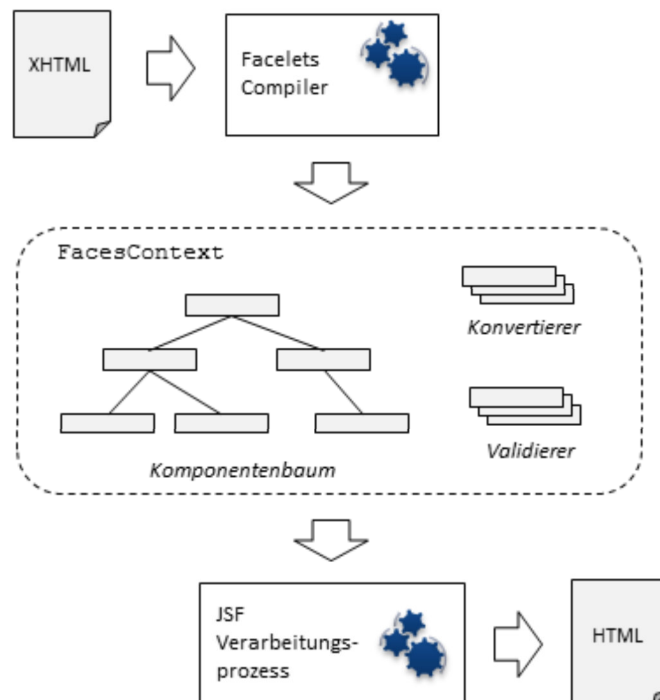

Abbildung 5: Eingliederung von Facelets in den JSF-Verarbeitungszyklus

1.5.3 Unified Expression Language

Einem Benutzer einer Webanwendung soll es möglich sein, Daten einzugeben und als Antwort Ergebnisse zu erhalten. Um die eingegebenen Daten im Model zu speichern und damit das Model Daten liefert, wird in JSF die Unified Expression Language (Unified-EL) verwendet. Die Verbindung zwischen View und Model findet mithilfe der Unified-EL statt.

Dabei gibt es zwei Arten von Bindungen zwischen JSF-Komponenten und Managed Beans:

- **Value Binding**
  Value Binding bedeutet das Binden von Werten einer JSF-Komponente an eine Eigenschaft der Managed Bean, beispielsweise beim Auslesen eines Texteingabefeldes.

  Beispiel zu Value Binding:

  ```
  <h:inputText value="#{loginBean.username}" />
  ```

  In diesem Beispiel wird ein String vom Texteingabefeld ausgelesen und in der Eigenschaft `username` der Managed Bean `loginBean` gespeichert.

- **Method Binding**
  Method Binding bedeutet das Binden an eine Methode einer Managed Bean, beispielsweise beim Drücken einer Schaltfläche. Der Methodenaufruf erfolgt dann beim Drücken dieser Schaltfläche.

  Beispiel zu Method Binding:

  ```
  <h:commandButton action="#{loginBean.login}" />
  ```

  In diesem Beispiel wird beim Drücken der Schaltfläche die Methode `login` der Managed Bean `loginBean` aufgerufen und ausgeführt.

Des Weiteren erlaubt die Unified-EL auch arithmetische Operatoren, Vergleichsoperatoren und den ternären ?:-Operator.

1.5.4 Managed Beans

Managed Beans bilden das Model einer Anwendung und werden vom JSF-Framework verwaltet. Sie repräsentieren die Daten und die Geschäftslogik einer Anwendung. Die Verbindung zwischen Eigenschaften und Methoden der Managed Bean wird mittels der Unified-EL realisiert.

Eine Managed Bean unterliegt dem JavaBeans-Standard und muss folgende Anforderungen erfüllen:

- Ein öffentlicher, nicht parametrisierter Konstruktor muss bereitgestellt werden.
- Der Zugriff auf Eigenschaften der Managed Bean muss mit getter- und setter-Methoden erfolgen.
- Muss serialisierbar sein.

Um dem JSF-Framework eine Managed Bean bekanntzumachen, wird die Annotation `@ManagedBean` einer Klasse vorangestellt. Mit dem Attribut `name` der Annotation wird sie unter diesem Namen registriert.

2 AJAX

2.1 Einführung in AJAX

Die Abkürzung AJAX steht für „**A**synchronous **J**avaScript **A**nd **X**ML“. AJAX ist jedoch keine neue Technologie, sondern vielmehr ein Sammelbegriff für eine Vielzahl von Technologien, die bereits seit mehreren Jahren eingesetzt werden. Die Basistechnologien, die dabei zum Einsatz kommen, sind XHTML, Cascading Style Sheets (CSS), das Document Object Model (DOM), JavaScript und das `XMLHttpRequest`-Object.

Der zentrale Baustein einer AJAX-Applikation ist die Skriptsprache JavaScript. Dazu greift eine mit AJAX entwickelte Webseite auf eine bereits vorhandene JavaScript-Bibliothek wie etwa jQuery zurück. Diese Bibliothek bietet Code für die einfache Verwendung des `XMLHttpRequest`-Objects an und behandelt auch Unterschiede zwischen einzelnen Browsern.

AJAX ist eine Technik, die dazu dient, dass Webseiten mit einem Server kommunizieren können, ohne dass dabei die entsprechende Webseite komplett neu geladen werden muss. Mit AJAX ist es somit möglich, immer nur genau diejenigen Daten an den Browser zu verschicken, die sich geändert haben.

Dies hat den Vorteil, dass der Benutzer beim Verwenden einer Webseite nicht durch komplettes Neuladen der Webseite unterbrochen wird.

Um die Vorteile von AJAX zu verdeutlichen, wird im Folgenden der Unterschied zwischen einer klassischen Webanwendung und einer Webanwendung mit AJAX erklärt.

Im klassischen Model einer Webanwendung wird bei jeder Änderung oder Aktion des Benutzers ein komplettes Neuladen der Webseite benötigt. Dies bedeutet, es muss eine Http-Verbindung zum Server aufgebaut werden, welcher dann die gesamten, aktualisierten Daten zurück an den Browser schickt. Die folgende Grafik zeigt dieses Verhalten:

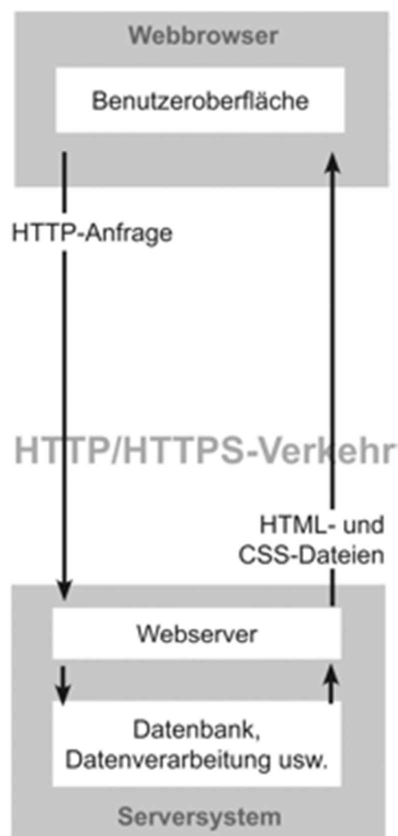

Abbildung 6: [B1] Klassische Webanwendung

Dieses Verhalten hat zur Folge, dass während der Server arbeitet, der Benutzer auf die Antwort warten muss. Erst nachdem der Server die Antwort geschickt hat, kann der Benutzer weiterarbeiten. Des Weiteren muss immer die ganze Seite neu geladen werden, was natürlich zu Performance-Einbußen führt.

Dies ist auch dann der Fall, wenn sich nur ein einziger Eingabewert geändert hat. Dies erhöht die Wartezeit beim Benutzer und den Netzwerkverkehr. Die folgende Abbildung verdeutlicht dieses Verhalten:

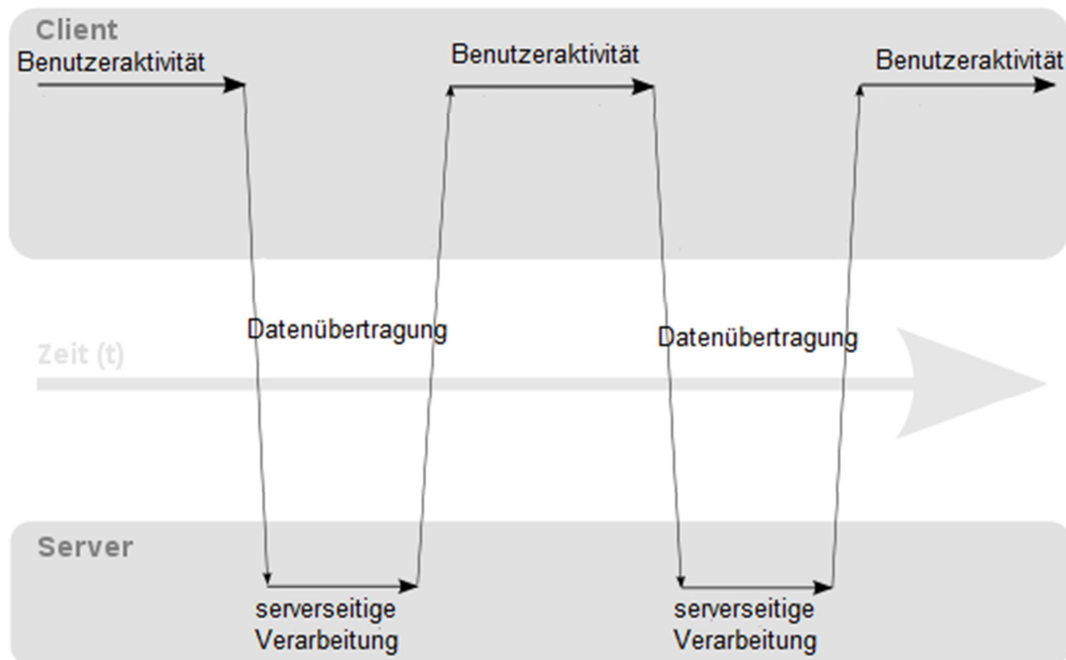

Abbildung 7: [B2] Datenübertragung bei klassischen Webanwendungen

Nun soll der Ablauf einer Webanwendung mit AJAX betrachtet werden. Wie bereits erwähnt, ist es mit AJAX möglich, nur kleine Teile einer Webanwendung – genauer Daten und Seiteninhalte der Webanwendung – an den Browser zu schicken. Dies wird durch die sogenannte AJAX-Engine ermöglicht, welche sich im Browser des Benutzers befindet. Diese Engine übernimmt nun die Kommunikation mit dem Server. Tätigt der Benutzer nun eine Änderung in Form einer Eingabe oder durch das Betätigen einer Schaltfläche, so wird zuerst ein JavaScript-Aufruf erzeugt. Die AJAX-Engine verarbeitet diesen Aufruf und stellt daraufhin die Verbindung zum Server in Form eines Http-Request her. Der große Vorteil ist, dass wesentlich weniger Daten übertragen werden müssen, da nur die Änderungen übertragen werden. Auch die Antwort des Servers kann dann mit JavaScript ausgewertet werden. Dies hat zur Folge, dass nur ein kleiner Teil der Webseite aktualisiert wird. Der Benutzer wird dabei nicht in seiner Arbeit auf der Webseite unterbrochen. Der Ablauf einer Webanwendung mit AJAX zeigt folgende Grafik:

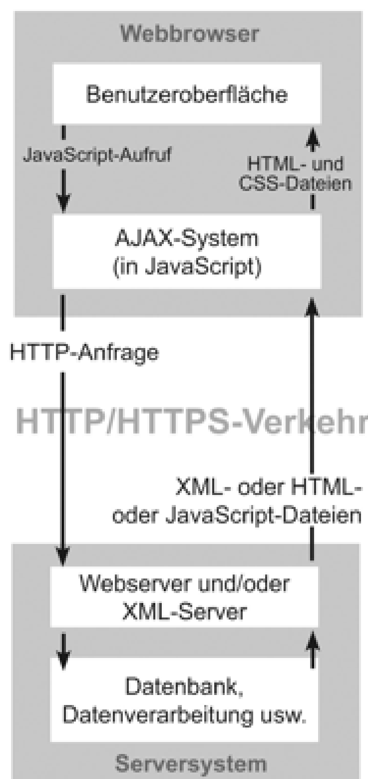

Abbildung 8: [B3] Webanwendung mit AJAX

Im Gegensatz zu einer klassischen Webanwendung erfolgt die Datenübertragung bei einer Webanwendung mit AJAX asynchron, da die Kommunikation mit dem Server durch die AJAX-Engine übernommen wird. Dies verdeutlicht die nächste Grafik:

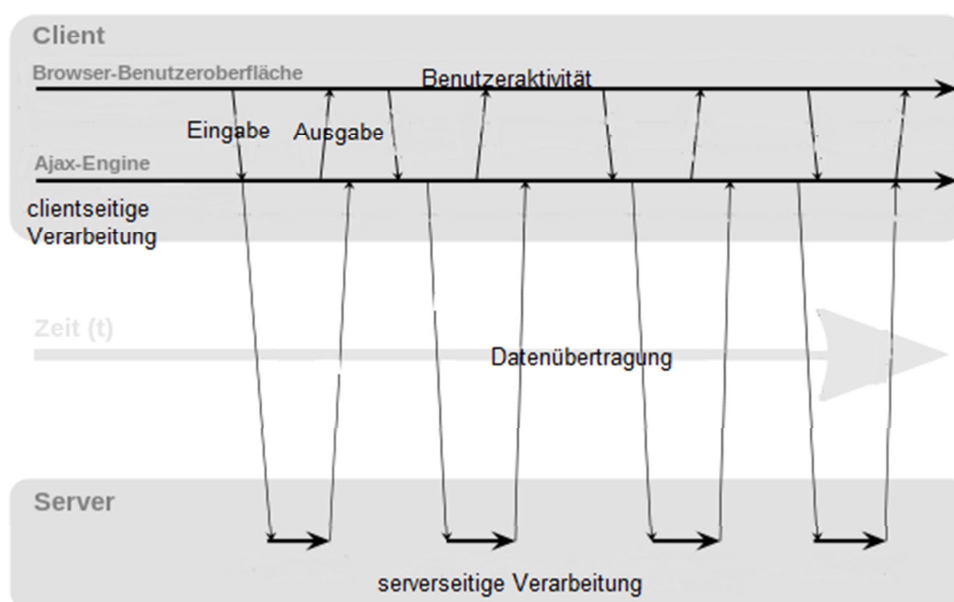

Abbildung 9: [B4] Datenübertragung einer Webanwendung mit AJAX

Die Vor- und Nachteile von AJAX werden im Folgenden kurz zusammengefasst:

- **Vorteile**
  - Die Webseite muss nicht komplett neu geladen werden
  - Reduzierung des Netzwerkverkehrs
  - Schnellere Reaktion auf Benutzereingaben
  - Kein Browser-Plug-In nötig

- **Nachteile**
  - JavaScript muss beim Benutzer aktiviert sein
  - Suchmaschinenoptimierung leidet (Suchmaschinen-Robots können kein JavaScript)
  - Probleme mit Browsernavigation (Vor- und Zurück-Button funktioniert nur über komplizierte Methoden)
  - Das Erstellen von Lesezeichen funktioniert nicht (es lassen sich nicht die einzelnen Zustände einer Seite abspeichern)

2.2 AJAX seit JSF 2.0

JSF definiert seit der Version 2.0 eine JavaScript-Bibliothek, die grundlegende AJAX-Operationen wie das Senden einer Anfrage und das Bearbeiten der Antwort abdeckt. Mithilfe dieser standardisierten Schnittstelle ist gewährleistet, dass alle Komponenten clientseitig dieselbe Funktionalität einsetzen und sich nicht in die Quere kommen. Diese JavaScript-API bildet auch die Grundlage für die Integration von AJAX in JSF-Anwendungen. JSF bietet grundsätzlich zwei verschiedene Ansätze, um Ansichten mit AJAX-Funktionalität auszustatten. Zum einen können Entwickler zum Absetzen einer AJAX-Anfrage direkt die Funktion der JavaScript-API aufrufen. Zum anderen gibt es mit dem Tag `<f:ajax>` eine deklarative Variante, um eine Komponente oder sogar einen ganzen Bereich einer Seitendeklaration mit AJAX-Funktionalität auszustatten. Die Spezifikation kümmert sich allerdings nicht nur um die Clientseite. Auch die Bearbeitung einer AJAX-Anfrage am Server ist umfassend spezifiziert. JSF 2.0 erweiterte den Lebenszyklus so, dass zum Bearbeiten einer AJAX-Anfrage nur die relevanten Teile des Komponentenbaums ausgeführt und gerendert werden. Diese Erweiterungen sind das **Partial-View-Processing** und das **Partial-View-Rendering**.

2.2.1 Das <f:ajax> Tag

Mit diesem Tag können eine ganze Reihe von JSF-Standardkomponenten mit AJAX-Verhalten ausgestattet werden. Um genau zu sein, sind es alle, die das Interface `ClientBehaviorHolder` implementieren.

Dadurch ist es ebenfalls möglich, eigene Komponenten zu erstellen, die dann auch mit AJAX-Verhalten ausgerüstet werden können.

Das Tag `<f:ajax>` kann auf zwei Arten eingesetzt werden. Entweder wird bei einer Komponente das Tag `<f:ajax>` als Kind-Tag eingefügt oder man wendet das Tag auf einen ganzen Bereich der Seite an. Das Tag `<f:ajax>` umschließt somit mehrere JSF-Komponenten, die mit AJAX-Verhalten ausgerüstet werden sollen. Nachfolgend wird an einem kurzen Beispiel erklärt, wie das Tag `<f:ajax>` in einer JSF-Anwendung zum Einsatz kommt. Dabei werden auch die verwendeten Attribute des Tags beschrieben.

Beispiel:

```
<h:form id="form">
     <f:ajax render="name">
          <h:inputText id="first" value="#{person.first}"/>
          <h:inputText id="last" value="#{person.last}"/>
          <h:commandButton value="Show">
               <f:ajax execute="first last" render="name"/>
          </h:commandButton>
     </f:ajax>
     <h:outputText id="name" value="#{person.name}"/>

</h:form>
```

In diesem Beispiel beinhaltet die Form zwei Eingabefelder für den Vor- und Nachnamen einer Person. Dabei ist es wichtig, dass auch mit AJAX das Tag `<h:form>` verwendet wird. Des Weiteren beinhaltet diese Form eine Schaltfläche sowie – wie weiter unten zu sehen – ein Textfeld. In diesem Textfeld soll der gesamte Namen ausgegeben werden. Ohne AJAX bewirkt ein Klick auf die Schaltfläche einen normalen Submit mit anschließendem Neuaufbau der gesamten Ansicht. Dabei ist der eingegebene Vor- und Nachname im Textfeld erst nach dem Übermitteln sichtbar. In diesem Beispiel wurden auch die zwei Arten, das Tag `<f:ajax>` einzusetzen, dargestellt.

Als Kind-Tag von `<h:commandButton>` bewirkt das Tag `<f:ajax>`, dass beim Drücken der Schaltfläche das Ausführen der beiden Eingabekomponenten und das Rendern des Textfeldes angestoßen werden. Das Textfeld wird somit ohne Neuaufbau der Ansicht aktualisiert.

Der Grund für das Ausführen der Eingabekomponenten liegt darin, dass das Kind-Tag im Attribut `execute` die IDs `first` und `last` der Eingabefelder zugewiesen bekommt. Das Gleiche gilt für das Rendern des Ausgabefeldes. Das Kind-Tag bekommt im Attribut `render` die ID des Ausgabefeldes zugewiesen.

Da das Tag `<f:ajax>` im Attribut `event` kein Ereignis definiert, werden nur jene Komponenten innerhalb des Tags mit AJAX-Verhalten ausgestattet, die ein Defaultevent haben. Konkret sind das die beiden Eingabekomponenten mit dem Event `valueChange`. Das AJAX-Verhalten der Schaltfläche für das Defaultevent bestimmt bereits das innere `<f:ajax>`-Tag und ändert sich deshalb nicht.

Mithilfe der zusätzlichen AJAX-Funktionalität – das umschließende `<f:ajax>` Tag – wird jetzt jedes Mal, wenn sich im Browser eines der beiden Eingabefelder ändert, das Textfeld neu gerendert. Die Tabelle 6 zeigt eine Übersicht aller Standardkom–ponenten und deren Defaultevent:

Komponente	Defaultevent
`<h:commandButton>`	action
`<h:commandLink>`	action
`<h:inputText>`	valueChange
`<h:inputTextarea>`	valueChange
`<h:inputSecret>`	valueChange
`<h:selectBooleanCheckbox>`	valueChange
`<h:selectOneRadio>`	valueChange
`<h:selectOneListbox>`	valueChange
`<h:selectOneMenu>`	valueChange
`<h:selectManyCheckbox>`	valueChange
`<h:selectListbox>`	valueChange
`<h:selectManyMenu>`	valueChange

Tabelle 6: AJAX-Default-Events

Es wäre auch möglich, im umschließenden `<f:ajax>`-Tag das Attribut `event` zu setzen. Für das obige Beispiel könnte `event=dbclick` gewählt werden. Dadurch würde nun bei jedem Doppelklick eine AJAX-Anfrage ausgelöst.

2.2.2 JavaScript-API

Die zweite Möglichkeit, AJAX unter JSF 2.0 zu verwenden, ist die JavaScript-Bibliothek. Diese Bibliothek befindet sich als Ressource mit dem Namen `jsf.js` in der Bibliothek `javax.faces`. Möchte man diese Bibliothek direkt verwenden, muss sie wie folgt eingebunden werden:

```
<h:outputScript name="jsf.js" library="javax.faces"
     target="head"/>
```

Nachdem die Ressource richtig in die Seite eingebunden wurde, können folgende Methoden der JavaScript-Bibliothek verwendet werden:

- `jsf.ajax.request(source, event, options)`
  Diese Methode sendet eine AJAX-Anfrage an den Server

- `jsf.ajax.response(request, context)`
  Diese Methode bearbeitet die Antwort des Servers auf die AJAX-Anfrage

- `jsf.ajax.addOnError(callback)`
  Diese Methode registriert eine Callback-Funktion zum Behandeln von Fehlern

- `jsf.ajax.addOnEvent(callback)`
  Diese Methode registriert eine Callback-Funktion zum Behandeln von Ereignissen

Nachdem in Kapitel 2.2.1 ein Beispiel zur Verwendung des `<f:ajax>` Tag gezeigt wurde, soll nun dieses Beispiel direkt mit der JavaScript-Bibliothek umgesetzt werden.

Beispiel:

```
<h:form id="form">
     <h:outputScript name="jsf.js" library="javax.faces"
target="head"/>
     <h:inputText id="first" value="#{person.first}"/>
     <h:inputText id="last" value="#{person.last}"/>
     <h:commandButton id="button" value="Show"
onclick="jsf.ajax.request(this, event, {execute: 'form:button
form:first form:last', render: 'form:name'}); return false;"/>
     <h:outputText id="name" value="#{person.name}"/>

</h:form>
```

In diesem Beispiel wird nun, anstatt das `<f:ajax>` Kind-Tag zu verwenden, direkt die JavaScript-Bibliothek in der Schaltfläche `<h:commandButton>` verwendet.

Dabei ist die Methode `jsf.ajax.request(source, event, options)` das Herzstück der Bibliothek. Diese Methode ist für das Senden der asynchronen AJAX-Anfragen an den Server zuständig. Mit den Parametern `source` und `event` werden das DOM-Element und das DOM-Ereignis übergeben, die für das Auslösen der AJAX-Anfrage verantwortlich sind. Der dritte Parameter `options` ist ein assoziatives Array, mit dem weitere Optionen als Schlüssel-Wert-Paare an die Funktion übergeben werden. Diese Optionen entsprechen weitestgehend den bereits bekannten Attributen von `<f:ajax>`. Die Funktion `jsf.ajax.request()` baut die AJAX-Anfrage zusammen und schickt sie asynchron an den Server. Sobald die Antwort auf die Anfrage vom Server zurückkommt, wird im Erfolgsfall die Funktion `jsf.ajax.response()` aufgerufen, um die Antwort zu verarbeiten und gegebenenfalls Teile der Ansicht neu aufzubauen.

Vergleicht man diese beiden Beispiele stellt sich die Frage, wozu die JavaScript-Bibliothek verwenden, wenn das `<f:ajax>`-Tag dasselbe tut und dazu noch einfacher anzuwenden ist? Die Antwort darauf ist, dass mit der JavaScript-Bibliothek auch eigene Komponenten mit AJAX-Funktionalität ausgerüstet werden können. Auch müssen diese Komponenten dann nicht das Interface `ClientBehavior-Holder` implementieren.

2.2.3 Partieller JSF-Lebenszyklus

Nachdem in den Kapiteln 2.2.1 und 2.2.2 erklärt wurde, wie eine AJAX-Anfrage zum Server gesendet wird, behandelt dieses Kapitel die serverseitige Bearbeitung der Anfragen.

JSF unterstützt seit der Version 2.0 das partielle Ausführen – **Partial-View-Processing** – und Rendern – **Partial-View-Rendering –**. Als Reaktion auf eine AJAX-Anfrage soll im ersten Schritt der Lebenszyklus nicht für den gesamten Komponentenbaum, sondern nur für die in der Anfrage spezifizierten Komponenten ausgeführt werden.

Im zweiten Schritt wird als Ergebnis der Anfrage ein weiterer Teil des Komponentenbaums, der nicht mit dem ersten Teil übereinstimmen muss, gerendert. Dazu wurde der Lebenszyklus in die zwei logischen Bereiche **Ausführen** und **Rendern** aufgeteilt. Diesen Lebenszyklus zeigt die folgende Grafik:

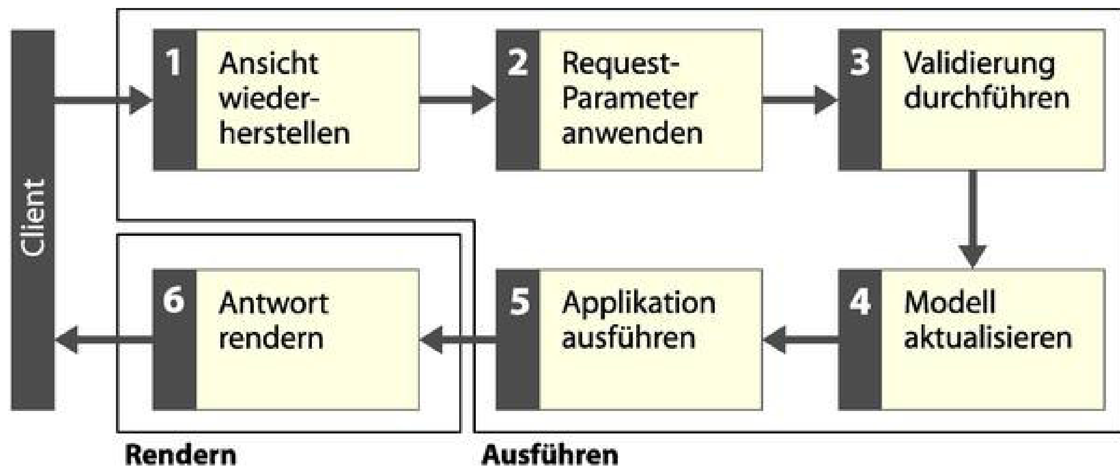

Abbildung 10: Partieller Lebenszyklus mit AJAX

Welche JSF-Komponenten in den beiden Bereichen zum Einsatz kommen, wird über die Parameter der AJAX-Anfrage bestimmt. Deren Werte entsprechen den Werten der `<f:ajax>`-Attribute `execute` und `render` beziehungsweise den gleichnamigen Parametern im assoziativen Array der Funktion `jsf.ajax.request()`.

3 Erweiterungsframeworks

In diesem Kapitel werden die drei bekanntesten Erweiterungsframeworks für JSF vorgestellt. Zusätzlich werden diese drei Frameworks mit JSF sowie untereinander verglichen.

3.1 RichFaces

Das RichFaces Framework ist eine Komponentenbibliothek für JavaServer Faces. Dieses Framework erweitert die Einsatzmöglichkeiten von AJAX – im Verbund mit JSF – um weitere Funktionen. RichFaces nutzt verschiedene Bereiche von JSF 2.0 einschließlich des Lebenszyklus, der Validierung, der Konvertierung und Verwaltung von statischen und dynamischen Ressourcen. Außerdem umfasst RichFaces Komponenten mit integrierter AJAX-Unterstützung, die in JSF-Anwendungen eingebaut werden können.

Das RichFaces Framework besteht aus zwei Tag-Bibliotheken. Diese zwei Bibliotheken sind:

- **a4j**
  Diese Tag-Bibliothek stellt AJAX-Core und Utility-Komponenten zur Verfügung.

- **rich**
  Diese Tag-Bibliothek bietet fertige UI-Komponenten. Dabei verfügen diese Komponenten bereits über eingebaute AJAX-Unterstützung. Standardmäßig benötigen diese Komponenten keine weitere Konfiguration, um Anfragen oder Updates zu senden.

Um diese Tag-Bibliotheken verwenden zu können, müssen diese in jeder XHTML-Seite wie folgt eingebunden werden:

```
<html xmlns:rich="http://richfaces.org/rich"
      xmlns:a4j="http://richfaces.org/a4j"  />
```

In den folgenden Unterkapiteln werden einige der wichtigsten Elemente des RichFaces Frameworks beschrieben.

3.1.1 AJAX Action-Komponenten

Diese Komponenten dienen dazu, um AJAX-Anfragen clientseitig an den Server zu senden. Im Folgenden werden ein paar der wichtigsten Action-Komponenten erklärt:

- **`<a4j:ajax>`**
  Mit `<a4j:ajax>` lässt sich AJAX-Verhalten zu Komponenten, die nicht über AJAX-Verhalten verfügen, hinzufügen. Die Voraussetzung dafür ist jedoch, dass die Komponenten das Interface `ClientBehaviorHolder` implementieren. Soll nun eine Komponente mit `<a4j:ajax>` ausgestattet werden, wird `<a4j:ajax>` einfach als Kind-Tag in diese Komponente eingefügt. Mithilfe des Attributs `event` ist es möglich, einen Trigger anzugeben. Wird dieses Attribut weggelassen, so wird das Defaultereignis der Vater-Komponente verwendet, um eine AJAX-Anfrage zu senden. Dies wird jetzt an einem Beispiel verdeutlicht.

  Beispiel:

```
<h:form>
  <h:inputText id="input" value="#{userBean.name}">
       <a4j:ajax render="outtext" />
  </h:inputText>
  <h:outputText id="outtext" value="#{userBean.name}" />
</h:form>
```

In diesem Beispiel wird das Defaultereignis `valueChanged` von `<h:inputText>` verwendet. Dies bedeutet, dass jedes Mal, wenn der Benutzer eine Eingabe tätigt, eine AJAX-Anfrage gesendet wird.

- **`<a4j:commandButton>`**
  Diese Komponente ist sehr ähnlich zum Oberflächenelement `<h:command-Button>` von JSF. Der Unterschied besteht jedoch darin, dass diese Komponente AJAX-Unterstützung anbietet. Des Weiteren benötigt diese Komponente nur das Attribut `value`, um zu funktionieren. Standardmäßig nutzt `<a4j:commandButton>` das Ereignis `click` anstatt des Ereignisses `submit`.

  Beispiel:

```
<h:form>
     <h:outputText value="Name:" />
      <h:inputText value="#{userBean.name}" />
      <a4j:commandButton value="Button" render="out"
           execute="@form" />
</h:form>

<a4j:outputPanel id="out">
   <h:outputText value="#{userBean.name}"
   rendered="#{userBean.name}"/>
</a4j:outputPanel>
```

  In diesem Beispiel beinhaltet die Form ein Eingabefeld für den Namen einer Person. Des Weiteren beinhaltet diese Form eine Schaltfläche, welche mithilfe des Tags `<a4j:commandButton>` dargestellt wird. Am Ende des Beispiels ist ein Output-Container mit dem Attribut `id="out"` zu sehen. Dieser Output-Container bewirkt, dass alle seine Kind-Komponenten automatisch aktualisiert werden. Gibt der Benutzer nun seinen Namen in das Textfeld ein und betätigt die Schaltfläche, bewirkt dies das Ausführen der Eingabekomponente und das Rendern des Textfeldes. Das Textfeld wird somit ohne Neuaufbau der Ansicht aktualisiert und der eingegebene Name wird angezeigt.

- **`<a4j:commandLink>`**
  Diese Komponente funktioniert ähnlich wie der `<a4j:commandButton>`. Anstatt einer Schaltfläche wird hier jedoch ein Hyperlink angezeigt.

- **<a4j:poll>**
  Mit dieser Komponente ist es möglich, periodische Anfragen an den Server zu senden und somit eine Seite periodisch zu updaten. Dazu wird das Attribut `interval` benutzt, um den Zeitzyklus in Millisekunden festzulegen indem die Anfragen gesendet werden sollen.

  Beispiel:

```
<h:form>
    <a4j:poll id="poll" interval="500" reRender="counter" />
</h:form>

<h:form>

   <a4j:commandButton value="Start Counter"
          action="#{UserBean.startCounter}" reRender="poll" />
   <a4j:commandButton value="Stop Counter"
        action="#{UserBean.stopCounter}" reRender="poll" />

   <h:outputText id="counter" value="#{UserBean.counter}" />

</h:form>
```

  In diesem Beispiel wird mit der Schaltfläche „Start Counter“ ein Zähler gestartet, der einen Wert hochzählt. Analog dazu kann dieser Zähler mit „Stop Counter“ angehalten werden. Die Seite wird durch `<a4j:poll>` alle 500ms aktualisiert und der Wert des Zählers wird in `<h:outputText>` angezeigt.

3.1.2 AJAX Container-Komponenten

Das Interface `AjaxContainer` markiert die Teile des JSF-Komponentenbaums, die bei jeder AJAX-Anfrage beim Client aktualisiert und gerendert werden sollen. Komponenten, die dieses Interface implementieren, sind:

- **<a4j:outputPanel>**
  Mithilfe dieser Komponente ist es möglich, mehrere Komponenten, die aktualisiert werden sollen, zu gruppieren und sie als Ganzes zu aktualisieren, ohne dabei die einzelnen Kind-Komponenten angeben zu müssen. Dies zeigt das folgende Beispiel.

  Beispiel:

```
<h:form>
     <h:outputText value="Name:" />
     <h:inputText value="#{userBean.name}" />
     <a4j:commandButton value="Button" render="out"
      execute="@form" />
```

```
</h:form>

<a4j:outputPanel id="out">
   <h:outputText value="#{userBean.name}"
        rendered="#{userBean.name}"/>
</a4j:outputPanel>
```

Dies ist dasselbe Beispiel wie in Kapitel 3.1.1 mit der Schaltfläche `<a4j:commandButton>`. Hierbei geht es jedoch um das `<a4j:outputPanel>`. Nachdem die Schaltfläche gedrückt wurde, werden alle Kind-Komponenten innerhalb der Komponente `<a4j:outputPanel>` automatisch aktualisiert, ohne dabei jede einzelne angeben zu müssen. In diesem Fall ist die Komponente `<h:outputText>` gruppiert. Es könnten sich natürlich noch mehrere Kind-Komponenten in der Gruppe befinden. Auch diese würden automatisch aktualisiert. Die Komponente `<a4j:outputPanel>` dient also als Wrapper-Komponente.

- **`<a4j:mediaOutput>`**
  Diese Komponente ermöglicht es, Bilder, Videos, Sounds und andere binären Ressourcen anzuzeigen. Diese Ressourcen können auch zur Laufzeit festgelegt werden.

3.1.3 Partial Tree Processing und Partial View Rendering

Auch RichFaces unterstützt das partielle Ausführen – **Partial Tree Processing** – und das partielle Rendern – **Partial View Rendering** –. Mit dem Attribut `execute` einer Komponente – zum Beispiel bei `<a4j:commandButton>` – wird angegeben welche Teile des JSF-Komponentenbaums während einer AJAX-Anfrage zu bearbeiten sind. Dies wird Partial Tree Processing genannt. Mithilfe des Attributs `render` einer Komponente – wieder als Beispiel von `<a4j:commandButton>` – wird angegeben, welche Teile aktualisiert werden sollen. Dies wird Partial View Rendering genannt.

3.1.4 Unterschiede zwischen JSF und RichFaces

JSF wertet alle Parameter während des Renders einer Seite aus. Während der Anfrage einer JSF-Seite werden alle – `execute` und `render` – Werte vom Client zum Server gesendet. Im Gegensatz dazu wertet RichFaces diese Optionen serverseitig während der Anfrage aus.

Das bedeutet, dass JSF auf aktuelle Änderungen während einer Anfrage nicht reagieren kann. RichFaces dagegen kann dies und wird die aktuellen Werte übernehmen. Dies funktioniert dadurch, dass RichFaces die Werte, die vom Client geändert wurden, serverseitig neu auswerten kann. Zusätzlich erhöht sich durch diesen Mechanismus die Datenintegrität.

3.2 ICEfaces

Das zweite Erweiterungsframework für JSF, das hier behandelt werden soll, ist ICEfaces. ICEfaces ist ein Open-Source Rich Internet Application (RIA) Entwicklungs-Framework für Java EE. ICEfaces funktioniert plattformübergreifend im Bereich von Desktop über Smartphones und Apple auf Android.

Die ICEfaces Komponentenbibliotheken unterstützen die neueste HTML 5-Version sowie Responsive-/ Adaptive-Design-Techniken. Damit ist es möglich, dass einzelne Seiten optimal auf verschiedenen Geräten dargestellt werden können.

ICEfaces bietet jedoch mehr als nur eine Reihe von Komponenten. Das Framework bietet außerdem neue Funktionen, die das Entwickeln mit Standard-Java EE erweitern und vereinfachen. Diese werden in den folgenden Unterkapiteln behandelt.

Um ICEfaces-Komponenten in einer JSF-Anwendung verwenden zu können, muss dies in jeder XHTML-Seite mit folgender Definition bekannt gemacht werden.

```
<html xmlns:ace="http://www.icefaces.org/icefaces/components"
      xlmns:icecore="http://www.icefaces.org/icefaces/core" />
```

3.2.1 Vergleich zwischen Automatic AJAX und standardgemäßen JSF AJAX

Automatic AJAX ist ein Mechanismus des ICEfaces-Frameworks, der minimale Seiten-Aktualisierungen von einem JSF-Lebenszyklus zum nächsten Lebenszyklus berechnet. Dadurch entfällt die Notwendigkeit des Tags `<f:ajax>`.

Seit JSF 2.0 wird AJAX-Funktionalität mit dem Tag `<f:ajax>` unterstützt. Durch dieses Tag wird eine AJAX-Anfrage gesendet. Diese wird im JSF-Lebenszyklus bearbeitet und schließlich wird die View neu gerendert und in die Seite eingefügt. Dabei entscheidet der Entwickler, durch welches Ereignis die Anfrage abgeschickt werden soll. Außerdem legt er fest, welche Komponenten daran beteiligt sein sollen, also welche Komponenten ausgeführt und welche Komponenten gerendert werden sollen. Dies wird nochmal an einem kleinen Beispiel verdeutlicht.

Beispiel:

```
<h:form>
     <h:inputText id="input" value="#{bean.value}">
          <f:ajax execute="@this" event="blur"
           render="output"/>
     </h:inputText>
     <h:outputText id="output" value="#{bean.value}"/>
</h:form>
```

In diesem Beispiel beinhaltet das Tag `<h:form>` zwei Kind-Komponenten. Als Erstes eine Eingabekomponente, welche mithilfe des Tags `<h:inputText>` dargestellt wird und als Kind-Tag das Tag `<f:ajax>` hat. Als Zweites ist die Komponente `<h:outputText>` zu sehen, welche den eingegebenen Wert dar–stellen soll. Den Ablauf der auftretenden Ereignisse zeigt das folgende Bild:

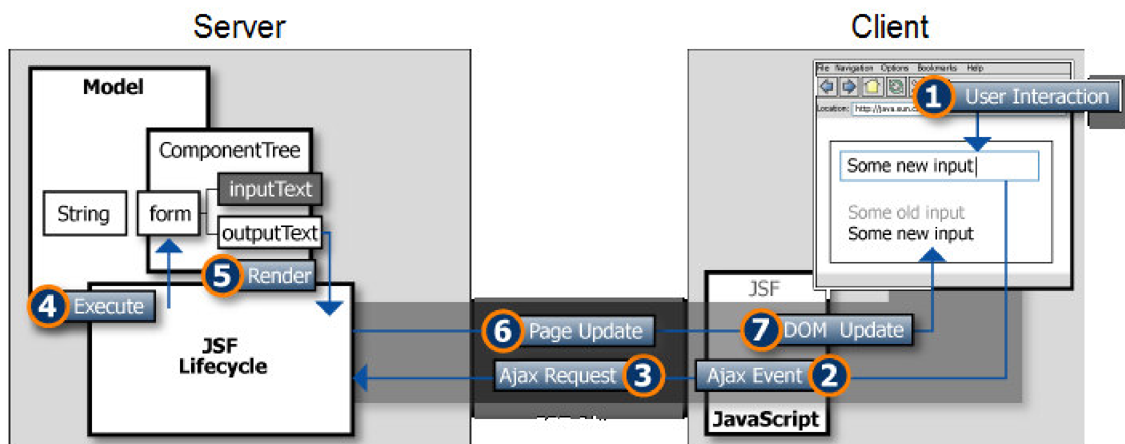

Abbildung 11: [B5] Klassicher JSF-AJAX Ablauf

Erklärung des Ablaufs:

1. Der Benutzer tätigt eine Eingabe.
2. Das Ereignis `blur` verursacht die Ausführung des Tags `<f:ajax>`.
3. Die AJAX-Anfrage wird zum Server gesendet.
4. Der JSF-Lebenszyklus führt nur die Komponente `<h:inputText>` aus und updatet das Model.
5. Die JSF Render Phase wird nur für `<h:outputText>` ausgeführt.
6. Die Antwort mit dem Seitenupdate für `<h:outputText>` wird erzeugt.
7. Das DOM Update wird für `<h:outputText>` angewendet.

Für diesen einfachen Fall genügt das Tag `<f:ajax>`. Wegen der steigenden Komplexität der Seiten steigt jedoch auch die Fehleranfälligkeit durch falsche Benutzung des Tags `<f:ajax>`.

Deswegen wird bei ICEfaces Automatic AJAX verwendet. Mithilfe dieser Funktion lassen sich minimale Seiten Updates erzeugen, ohne dass der Entwickler überprüfen muss, wie, wann oder warum Seiten-Updates auftreten.

Der Grund hierfür ist, dass das ICEfaces-Framework Direct-To-DOM (D2D) Rendering verwendet. D2D schreibt die Ausgabe der JSF-Rendering-Phase in ein serverseitiges Document-Object-Model. Dieses DOM ist dann genau das, was das Client-DOM benötigt.

Mithilfe einer zwischengespeicherten Version des DOM ist es dem Framework möglich, genau diejenigen Updates zu ermitteln, welche benötigt werden, um Änderungen der Ansicht nach Beendigung des JSF-Lebenszyklus zu beeinflussen.

Während D2D sich um die Seiten-Updates kümmert, muss dieser Prozess immer noch durch ein UI-Ereignis gestartet werden. Dazu bietet ICEfaces die Funktion `Single Submit` an. Mithilfe dieser Funktion führen die Komponenten automatisch einen Submit aus, sobald der Benutzer mit ihnen interagiert. Die Funktion `Single Submit` verhält sich ähnlich wie dieses Beispiel:

```
<f:ajax execute="@this" render="@all"/>
```

Aber der Unterschied besteht darin, dass nicht die gesamte Seite aktualisiert wird, da der D2D-Mechanismus das Seiten-Update abfängt und die minimale Menge der benötigten Updates berechnet. Die `Single Submit`-Funktion gibt es als Attribut eingebaut in den ICEfaces Advanced Components (ACE Components) oder als umschließendes Tag `<icecore:singleSubmit>`. Hierzu ebenfalls ein Beispiel.

Beispiel:

```
<h:form>
      <icecore:singleSubmit>
            <h:inputText id="input" value="#{bean.value}"/>
            <h:outputText id="output" value="#{bean.value}"/>
      </icecore:singleSubmit>

</h:form>
```

In diesem Beispiel übernimmt das umschließende Tag `<icecore:single-Submit>`, dass automatisch ein Submit ausgeführt wird, sobald der Benutzer eine Eingabe tätigt.

Den Ablauf der auftretenden Ereignisse zeigt das folgende Bild:

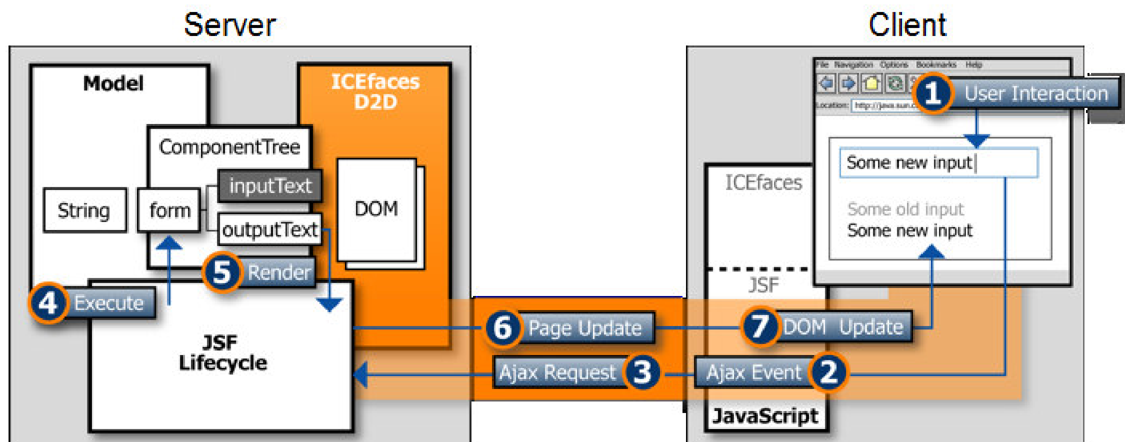

Abbildung 12: [B6] Ablauf mit ICEfaces

Erklärung des Ablaufs:

1. Der Benutzer tätigt eine Eingabe.
2. Das Ereignis `blur` verursacht den Aufruf der Funktion `Single Submit` der Komponente `<h:inputText>`.
3. Durch das Ausführen der Funktion `Single Submit` wird eine AJAX-Anfrage erzeugt und zum Server gesendet.
4. Der JSF-Lebenszyklus führt nur die `<h:inputText>`-Komponente aus und updatet das Model.
5. Die JSF-Rendering-Phase durchläuft den gesamten Komponentenbaum und erzeugt ein serverseitiges DOM.
6. Die DOM-Unterschiede werden berechnet und das Seiten-Update für die Komponente `<h:outputText>` wird als Antwort erzeugt.
7. Das DOM-Update wird auf die Komponente `<h:outputText>` angewendet.

Automatic AJAX optimiert zwar die Entwicklungszeit, aber geht zu Lasten der Laufzeit. Falls es nötig ist, die gesamte Seite neu zu rendern, muss auch das gesamte DOM verglichen werden. Es ist möglich, mit dem Tag `<f:ajax>` dieses Verhalten zu optimieren, indem man mit diesem Tag für eine bestimmte Komponente das Verhalten des Attributs `render="@all"` überschreibt. Dadurch wird es dem Entwickler ermöglicht, eine Untergruppe anzugeben die übertragen werden soll.

Das ICEfaces Framework erkennt dies als **Partial Rendering** und führt dann den D2D-Mechanismus nur für diese Untergruppe des Komponentenbaums aus.

3.2.2 Direct-To-DOM Rendering

Direct-To-DOM (D2D) Rendering ist ein Mechanismus, der einen JSF-Kompo–nentenbaum in ein DOM nach dem W3C Standard rendert. Das gerenderte DOM wird zwischengespeichert und wird dann dazu genutzt, um den Unterschied zwischen zwei aufeinanderfolgenden Seitenaufrufen festzustellen. Dabei wird die kleinste Menge an Änderungen von der einen Seite zur anderen festgestellt, um diese dann zu updaten. D2D ist der Kern der Automatic AJAX-Funktion des ICEfaces Frameworks.

Um diese Technologie zu verwenden, muss der JSF-Anwendung die Datei `ICEfaces2.0.jar` hinzugefügt werden. Dabei wird das Standard **RenderKit** mit dem **ICEfaces RenderKit** überschrieben.

Das **ICEfaces RenderKit** stellt den **DomResponseWriter** zur Verfügung. Dieser leitet die Ausgabe der gerenderten Komponenten in ein serverseitiges DOM um.

Dieses – vom D2D-Mechanismus – neu erzeugte DOM stellt eine Abbildung des clientseitigen DOMs dar. Mithilfe des neu erstellen DOM und des zwischen–gespeicherten DOM kann der D2D-Mechanismus die Unterschiede feststellen und die benötigten Updates berechnen.

Der Standard JSF-AJAX-Mechanismus wird dann dazu verwendet, diese Unterschiede an das Client DOM zu übertragen und einzufügen. Damit ist der JSF-Lebenszyklus mit D2D-Rendering abgeschlossen. Dieser Prozess wird in der folgenden Grafik dargestellt:

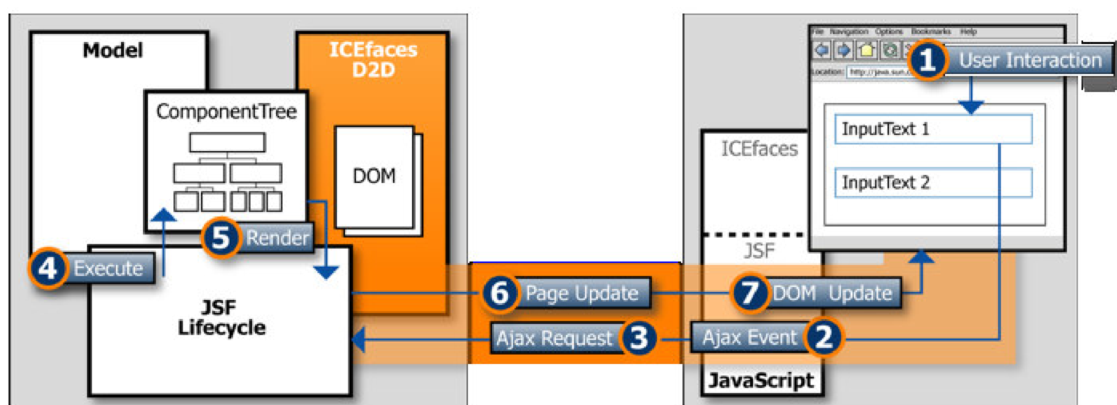

Abbildung 13: [B7] D2D Mechanismus

Erklärung des Ablaufs:

1. Der Benutzer interagiert mit einigen Browser-Steuerelementen auf einer Seite.
2. Durch das Ausführen der Funktion `Single Submit` wird eine AJAX-Anfrage erzeugt.
3. Die AJAX-Anfrage wird zur Anwendung gesendet.
4. Der JSF-Lebenszyklus wird für eine Komponente des Komponentenbaums ausgeführt und das Model wird aktualisiert.
5. Die Rendering-Phase wird für den gesamten Komponentenbaum durchlaufen. Der D2D-Mechanismus berechnet dann die erforderlichen Updates.
6. Seiten-Updates werden als AJAX-Antwort zurückgesendet.
7. Der Standard JSF-Mechanismus aktualisiert die clientseitige DOM mit den Seiten-Updates.

3.2.3 Single Submit

Die Funktion `Single Submit` bietet die Möglichkeit, eine Komponente im JSF-Lebenszyklus auszuführen. Das bedeutet, dass nur die angegebene Komponente dekodiert, validiert, die zugehörige JavaBean aktualisiert und ihr Ereignis ausgelöst wird. Jedoch wird eine vollständige Rendering-Phase durchlaufen. Anschließend wird der D2D-Mechanismus verwendet, um herauszufinden, was aktualisiert werden muss.

Wie bereits erwähnt gibt, es zwei Möglichkeiten, die Funktion `Single Submit` zu verwenden. Entweder als Attribut `singleSubmit` in den ACE-Komponenten oder als Tag `<icecore:singleSubmit>`.

3.2.4 ICEfaces-Komponenten

In diesem Kapitel wird kurz auf die Hauptmerkmale der Komponenten ICEfaces Advanced Components (ACE Components) und ICEfaces Core Components (ICECORE) eingegangen:

- **ACE-Komponenten**
  Die ACE-Komponenten nutzen eine Mischung aus serverseitigen und client–basierten Rendering-Techniken, um eine mächtige und ansprechende Web–anwendung mit reduziertem Netzwerkverkehr und weniger serverseitiger Verarbeitung bereitzustellen. Die Hauptmerkmale der ACE Komponenten sind folgende:

  - ACE-Komponenten nutzen leistungsstarke JavaScript-Komponenten von 3rd-Party Bibliotheken wie jQuery, ohne dass der Entwickler JavaScript direkt verwenden muss.
  - ACE-Komponenten unterstützen umfangreiche clientseitige Funktionalität, um Webanwendungen ansprechbarer und skalierbarer zu machen.

  Im Folgenden wird noch ein Beispiel für eine ACE-Komponente gegeben. Es handelt sich dabei um eine der meist genutzten Komponenten, der `DataTable`-Komponente. In ICEfaces wird eine `DataTable`-Komponente mit dem Tag `<ace:dataTable>` realisiert.

  Beispiel:

```
<h:form>
     <ace:dataTable id="carTable"
           value="#{dataTableBean.carsData}" var="car"
           paginator="true"
           paginatorPosition="bottom"
           rows="10">
       <ace:column id="id" headerText="ID"
          sortBy="#{car.id}" filterBy="#{car.id}"
          filterMatchMode="contains">
            <h:outputText id="idCell" value="#{car.id}"/>
       </ace:column>
       <ace:column id="name" headerText="Name"
          sortBy="#{car.name}" filterBy="#{car.name}"
          filterMatchMode="contains">
            <h:outputText id="nameCell"
               value="#{car.name}"/>
       </ace:column>
       <ace:column id="chassis" headerText="Chassis"
          sortBy="#{car.chassis}"
          filterBy="#{car.chassis}"
          filterMatchMode="contains">
            <h:outputText id="chassisCell"
               value="#{car.chassis}"/>
       </ace:column>
```

```
        <ace:column id="weight" headerText="Weight"
           sortBy="#{car.weight}" filterBy="#{car.weight}"
           filterMatchMode="contains">
              <h:outputText id="weightCell"
                value="#{car.weight}lbs."/>
        </ace:column>
        <ace:column id="accel" headerText="Accel"
           sortBy="#{car.acceleration}"
           filterBy="#{car.acceleration}"
           filterMatchMode="contains">
              <h:outputText id="accelerationCell"
                value="#{car.acceleration}"/>
        </ace:column>
        <ace:column id="mpg" headerText="MPG"
           sortBy="#{car.mpg}" filterBy="#{car.mpg}"
           filterMatchMode="contains">
              <h:outputText id="mpgCell"
                value="#{car.mpg}"/>
        </ace:column>
        <ace:column id="cost" headerText="Cost"
           sortBy="#{car.cost}" filterBy="#{car.cost}"
           filterMatchMode="contains">
              <f:facet name="header">Cost</f:facet>
              <h:outputText id="costCell"
                value="#{car.cost}">
               <f:convertNumber type="currency"
                  currencySymbol="$" groupingUsed="true"
                  minFractionDigits="2"
                  maxFractionDigits="2"/>
              </h:outputText>
         </ace:column>
    </ace:dataTable>
</h:form>
```

In diesem Beispiel wird eine `DataTable`-Komponente definiert. Im Attribut `value="#{dataTableBean.carsData}"` wird eine Liste mit Fahrzeugen und deren Eigenschaften referenziert. Mithilfe des Attributs `paginator="true"` wird ein Seitenumbruch erzeugt und durch das Attribut `row="10"` werden pro Seite maximal 10 Reihen angezeigt. Durch das Tag `<ace:column>` wird eine Spalte zur `DataTable`-Komponente hinzugefügt. Insgesamt wurden hier sieben Spalten erzeugt, welche den Eigenschaften eines Fahrzeugs entsprechen. Die `DataTable`-Komponente kann die Daten nicht nur tabellarisch darstellen, sondern mithilfe der Tags `sortBy` und `filterBy` kann spaltenweise der Inhalt sortiert oder gefiltert werden.

Die folgende Grafik zeigt die gerenderte Form im Browser dieser `DataTable`-Komponente.

ID	Name	Chassis	Weight	Accel	MPG	Cost
1	Enduro	Van	15383lbs.	10	17.86	$6,617.17
2	Tamale	Bus	7331lbs.	15	16.65	$31,464.24
3	Doublecharge	Pickup	5333lbs.	15	17.84	$10,922.73
4	Swordfish	Bus	10956lbs.	5	5.17	$6,019.83
5	Iguana	Pickup	1696lbs.	10	9.43	$19,736.16
6	Dart	Motorcycle	9261lbs.	15	12.85	$37,947.84
7	Pisces	Luxury	7846lbs.	10	15.13	$19,235.20
8	Flash	Mid-Size	11499lbs.	10	12.74	$29,942.38
9	Tomcat	Mid-Size	10766lbs.	15	7.04	$14,342.74
10	Passion	Subcompact	2082lbs.	10	13.38	$8,015.01

1 2 3

Abbildung 14: ICEfaces DataTable

Wie bereits erwähnt ist es mit der `DataTable`-Komponente möglich, Inhalte zu filtern. Dazu wurde in das Eingabefeld – unterhalb der Überschrift Name – der String „En“ eingegeben. Das gefilterte Resultat zeigt die nächste Abbildung.

ID	Name	Chassis	Weight	Accel	MPG	Cost
	En					
1	Enduro	Van	15383lbs.	10	17.86	$6,617.17
15	Endino	Subcompact	4997lbs.	5	6.02	$31,221.48
18	Encyclo	Motorcycle	5725lbs.	5	14.17	$34,430.44

Abbildung 15: gefilterter ICEfaces DataTable

- **ICECORE-Komponenten**
  Die ICEfaces Core Components (ICECORE) sind eine Reihe von Komponenten, die nicht sichtbar sind. Sie bieten jedoch eingebaute Lösungen für allgemeine Probleme in vielen JSF-Anwendungen wie zum Beispiel die Komponente `<icecore:singleSubmit>` in Kapitel 3.2.1, welche automatisch einen Submit ausführt.

3.3 PrimeFaces

PrimeFaces ist die wahrscheinlich populärste JSF-Komponentenbibliothek. Das Framework bietet ebenfalls eingebaute AJAX-Unterstützung mithilfe der auf JavaScript basierten Bibliothek jQuery.

Außerdem bietet PrimeFaces ebenfalls eine Reihe von Komponenten, die über den JSF-Standard hinausgehen und zusätzlich wird auch die AJAX-Unterstützung erweitert.

Um PrimeFaces Komponenten nutzen zu können, muss in jeder XHTML-Datei folgendes deklariert werden:

```
<html xmlns:p="http://primefaces.org/ui" />
```

3.3.1 PrimeFaces-Komponenten

PrimeFaces bietet für eine ganze Reihe von Komponenten aus dem JSF-Standard Erweiterungen an wie zum Beispiel `<p:inputText>`, `<p:selectOneRadio>` oder `<p:outputLabel>`. Im Grunde verfügen diese Komponenten über dieselbe Grund–funktionalität wie die gleichnamigen Standardkomponenten. Einige davon sind jedoch mit Zusatzfunktionalität ausgerüstet. Auch die Konverter und Validatoren aus der JSF-Tag-Bibliothek können weiterhin verwendet werden.

Des Weiteren gibt es eine große Zahl von PrimeFaces eigenen Komponenten. Diese alle aufzuzählen würde jedoch den Rahmen dieser Arbeit sprengen. Aus diesem Grund werden hier nur zwei PrimeFaces-eigene-Komponenten in den folgenden Beispielen erklärt:

- **`<p:ajax>`**
  Das Tag `<p:ajax>` wird ähnlich wie das Tag `<f:ajax>` verwendet. Dieses Tag kann nicht wie `<f:ajax>` als umschließendes Tag über mehrere Komponenten, sondern nur innerhalb einer einzelnen Komponente als Kind-Tag verwendet werden. Ein weiterer Unterschied ist, dass an Stelle des Attributs `execute` das Attribut `process` und an Stelle des Attributs `render` das Attribut `update` benutzt wird. Benutzt werden kann dieses Tag für alle JSF-Komponenten sowie für alle PrimeFaces-Komponenten. Zusätzlich werden per Definition in JSF bei jeder AJAX-Anfrage die Daten des gesamten Formulars an den Server geschickt. Falls dies nicht gewünscht ist, bietet das Tag `<p:ajax>` die Möglichkeit, über das Attribut `partialSubmit` einzugreifen. Wird dieses Attribut auf **true** gesetzt, werden nur die Daten der für die AJAX-Anfrage relevanten Komponenten an den Server gesendet. Die Möglichkeit, das Attribut `partialSubmit` verwenden zu können, ist ein weiterer Unterschied zu Standard-JSF. Im Folgenden ein kurzes Beispiel.

Beispiel:

```
<h:form>
   <p:inputText id="name" value="#{bean.name}" >
        <p:ajax event="keyup" update="outName" />
   </p:inputText>
   <h:outputText id="outname" value="#{bean.name}"/>
</h:form>
```

In diesem Beispiel kann der Benutzer eine Eingabe tätigen. Da die Komponente `<p:inputText>` die Komponente `<p:ajax>` als Kind-Tag beinhaltet und die Komponente `<p:ajax>` auf das Ereignis `keyup` reagiert, wird jedesmal, wenn der Benutzer eine Taste drückt und diese wieder loslässt, eine AJAX-Anfrage an den Server gesendet. Außerdem wird durch das Attribut `update` die Kom–ponente `<h:outputText>` neu gerendert.

- **`<p:poll>`**
  Diese Komponente erlaubt das periodische Senden von AJAX-Anfragen. Das Intervall zwischen zwei Anfragen wird dabei im Attribut `interval` als Sekundenwert angegeben. Die restlichen Attribute zum Steuern der AJAX-Anfrage wie etwa `process` oder `update` sind identisch zur Komponente `<p:ajax>`. Auch zu dieser Komponente ein kurzes Beispiel.

  Beispiel:

```
<h:form>
   <p:poll interval="5" listener="#{counterBean.counter}"
               update="outCount" />
   <h:outputText id="outCount" value="#{counterBean.count}" />
</h:form>
```

  In diesem Beispiel wird alle fünf Sekunden eine AJAX-Anfrage an den Server gesendet. Während der Abarbeitung dieser Anfrage am Server wird zuerst die im Attribut `listener` referenzierte Methode `counter` aufgerufen. Diese Methode inkrementiert das Attribut `count` der JavaBean `CounterBean` um eins. Anschließend rendert JSF die Komponente `<h:outputText>` neu und aktualisiert die Ausgabe am Client. Des Weiteren gibt es Komponenten, welche die Attribute von `<p:ajax>` bereits integriert haben. Beispiele dafür sind die Komponente `<p:commandButton>` oder die Komponente `<p:commandLink>`. Diese Komponenten senden standardmäßig AJAX-Anfragen, wenn nicht explizit das Attribut `ajax` auf **false** gesetzt wird. Auch hierzu ein abschließendes Beispiel.

Beispiel:

```
<h:form>
  <p:inputText id="name" value="#{bean.name}" />
  <p:commandButton value="Submit" process="name"
    update="outText" />
  <h:outputText id="outText" value="#{bean.name}"/>
</h:form>
```

Bei einem Klick auf die Schaltfläche wird ohne spezielle Vorkehrungen automatisch eine AJAX-Anfrage ausgelöst. Durch die Angaben in den Attributen `process` und `update` wird serverseitig die Komponente mit dem Attribut `id="name"` ausgeführt und die Komponente mit dem Attribut `id="outText"` wird neu gerendert.

- **<p:accordionPanel>**
  Mit dem Tag `<p:accordionPanel>` wird die Komponente `AccordionPanel` realisiert. Diese Komponente stellt mehrere auf- und zuklappbare Tabs untereinander dar. Dabei gruppiert ein Tab beliebigen Inhalt und wird durch das Tag `<p:tab>` realisiert. Die einzelnen Tabs können durch Anklicken auf ihre Titelzeile auf- und zugeklappt werden. Standardmäßig ist maximal ein Tab aktiv. Wird ein inaktiver Tab ausgewählt, so wird dieser aufgeklappt und der zuvor aktive Tab wird wieder zugeklappt. Um sich ein AccordionPanel besser vorstellen zu können, wird eine Realisierung eines AccordionPanels im folgenden Beispiel gezeigt.

Beispiel:

```
<h:form>
  <p:accordionPanel>
       <p:tab title="Tab 1"> Inhalt Tab 1 </p:tab>
       <p:tab title="Tab 2"> Inhalt Tab 2 </p:tab>
       <p:tab title="Tab 3"> Inhalt Tab 3  </p:tab>
  </p:accordionPanel>
</h:form>
```

In diesem Beispiel wird eine Komponente vom Typ `AccordionPanel` erstellt mit drei Tabs als Kind-Komponenten. Dabei wird im Attribut `title` die Überschrift des jeweiligen Tabs angegeben. In diesem Beispiel ist der Inhalt der Tabs reiner Text, es können jedoch anstatt Text auch Bilder, Hyperlinks oder andere Komponenten eingefügt werden. Standardmäßig werden alle Tabs der Kom–ponente `<p:accordionPanel>` gerendert. Bei sehr komplexen Tabs kann dies jedoch durchaus zu langen Ladezeiten beim Seitenaufbau führen. Deswegen bietet PrimeFaces die Möglichkeit, Tabs dynamisch über AJAX nachzuladen.

Dazu wird im Tag `<p:accordionPanel>` das Attribut `dynamic` auf **true** gesetzt. Dies hat den Vorteil, dass Tabs erst beim Aktivieren nachgeladen werden. Bei der im Beispiel gezeigten Variante kann nur ein Tab zu einem Zeitpunkt aktiv sein, jedoch erlaubt die Komponente `<p:accordionPanel>` das Aufklappen mehrerer Tabs, wenn das Attribut `multiple` auf **true** gesetzt wird. Wie dieses Beispiel im Browser aussieht, zeigt folgendes Bild:

Abbildung 16: [B8] AccordionPanel

- **`<p:calendar>`**
  Mit dem Tag `<p:calendar>` bietet PrimeFaces eine komfortable Lösung, ein Datum auszuwählen. Die `Calendar`-Komponente hat zwei verschiedene Darstellungsmodi, welche über das Attribut `mode` gesetzt werden können. Der Modus `mode="inline"` bietet dem Benutzer eine Komponente an, welche rein als Auswahlfeld für ein Datum ohne Texteingabemöglichkeit fungiert.

  Im Modus `mode="popup"` hingegen wird die Komponente als Eingabefeld angezeigt. Dabei wird die Calendar-Komponente nur bei Bedarf angezeigt. Außer dem Attribut `mode` verfügt die Komponente `<p:calendar>` über einige weitere Attribute, um das Verhalten und die Darstellung anzupassen. Über das Attribut `showOn` lässt sich das Öffnen des Auswahlfeldes im Popup-Modus steuern. Dazu wird mithilfe des Attributs `showOn="button"` die Datumsauswahl über eine Schaltfläche neben dem Eingabefeld geöffnet und mit dem Attribut `showOn="focus"` wird die Datumsauswahl geöffnet, sobald das Eingabefeld den Fokus hat. Mit dem Attribut `pattern` lässt sich das Format des Datums festlegen. Es folgt ein Beispiel für die `Calendar`-Komponente.

  Beispiel:

```
<h:form>
   <p:outputLabel for="calendar" value="Datum: " />
   <p:calendar id="calendar"
               value="#{calendarBean.date}"
               mode="popup" showOn="focus"
               pattern="dd.MM.yyyy"/>
</h:form>
```

In diesem Beispiel wird eine einfache `Calendar`-Komponente im Popup-Modus erzeugt. Das Attribut `showOn=“focus”` bewirkt, dass sobald die Komponente `<p:outputLabel>` den Fokus hat, die `Calendar`-Komponente geöffnet wird. Zusätzlich wurde das Format des Datums mithilfe des Attributs `pattern=“dd.MM.yyyy”` gesetzt. Die gerenderte Form im Browser zeigt die folgende Grafik:

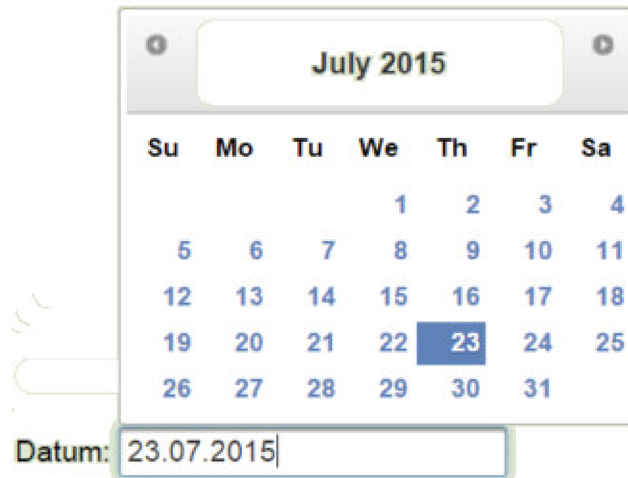

Abbildung 17: Calendar

- **`<p:dataTable>`**
  Mithilfe des Tags `<p:dataTable>` wird eine leistungsfähige Komponente zur tabellarischen Darstellung dynamischer Daten erzeugt. Im Gegensatz zur Standard-JSF-Komponente `<h:dataTable>`, bei der Features wie Sortieren oder Filtern fehlen, unterstützt `<p:dataTable>` diese Features. Die grundlegende Funktionsweise der Komponente `<p:dataTable>` entspricht der Komponente `<h:dataTable>`. Auch zu dieser Komponente wird ein Beispiel gegeben.

  Beispiel:

```
<h:form id="“formLandedAirplanes”>
      <p:dataTable
            value=“#{airportFacadeBean.notStartedAirplanes}”
            id=“tableLandedAirplanes” var=“aps”
            style=“border:3px solid black” >
      <f:facet name="header">Available Airplanes on
                                    Airport
      </f:facet>
      <p:column>
            <f:facet name=“header”>Airplane</f:facet>
            #{aps.name}
```

```
        </p:column>
        <p:column>
            <f:facet name="header">Type</f:facet>
            #{aps.type}
        </p:column>
        <p:column>
            <f:facet name="header">Airline</f:facet>
            #{aps.airline.name}
        </p:column>
        <p:column>
            <f:facet name="header">Guide</f:facet>
            #{aps.guide.name}
        </p:column>
        <p:column>
            <f:facet name="header">targetStartTime</f:facet>
            #{aps.targetStartTime}
        </p:column>
        <p:column>
            <f:facet name="header">actualLandingTime</f:facet>
            #{aps.actualLandingTime}
        </p:column>
        </p:dataTable>
    </h:form>
```

`<p:dataTable>` verfügt über die Attribute `value` und `var`. In diesem Beispiel referenziert `value` eine Liste von Instanzen der Klasse `Flugzeug`. Eine Instanz der Klasse `Flugzeug` besitzt verschiedene Eigenschaft wie einen Namen, einen Typ, die zugehörige Fluggesellschaft, einen Lotsen sowie eine Start- und Landezeit. Jede Eigenschaft eines Flugzeugs wird jeweils als eigene Spalte mit dem Tag `<p:column>` definiert. Mithilfe des Attributs `var="aps"` werden die einzelnen Instanzen der Liste in die Komponente `<p:dataTable>` eingetragen. Die Überschriften der Spalten werden mit dem Tag `<f:facet>` gesetzt.

Auch hierzu einen Darstellung einer gerenderten `DataTable`-Komponente:

Started Airplanes

Airplane	Type	Airline	Guide	targetLandingTime	actualStartTime
ap1	type	Lufthansa	Simon	2015-07-23 18:00:00.0	2015-07-23 17:50:47.159
ap2	type	Lufthansa	Timo		2015-07-23 17:47:46.507
ap3	type	Lufthansa	Simon		2015-07-23 17:50:03.431

Abbildung 18: PrimeFaces DataTable

3.3.2 Partial Rendering und Partial Processing

PrimeFaces bietet – basierend auf Standard-JSF 2.0 – die Möglichkeit des **Partial Rendering** und **View Processing**. Damit wird dem JSF-Lebenszyklus mitgeteilt, was ausgeführt und was gerendert werden soll.

Zusätzlich zu den Komponenten, die bereits integriertes AJAX-Verhalten besitzen, bietet PrimeFaces auch einen Mechanismus an, um JSF-Komponenten mit AJAX zu aktualisieren.

Damit diese Komponenten das Partial Rendering unterstützen, werden sie mit den Attributen `update`, `process`, `onStart` und `onComplete` ausgestattet. Beim Partial Rendering werden dann nur die Komponenten gerendert, die in diesen Attributen angegeben sind.

Beim Partial Processing werden ebenfalls nur die Komponenten ausgeführt, die angegeben werden. Dies bedeutet, es wird nur für die angegebenen Komponenten der JSF-Lebenszyklus durchlaufen.

3.3.3 Partial Submit

Wie bereits erwähnt, gibt es in PrimeFaces die Funktionalität des partiellen Ausführens. Damit werden der Netzwerkverkehr und die Datenverarbeitung verringert. Um diese Funktionalität nutzen zu können, muss sie aktiviert werden. Dies geschieht, in dem das Attribut `partialSubmit` – zum Beispiel der Komponente `<p:commandButton>` – auf **true** gesetzt wird. Wenn dieses Attribut aktiviert ist, werden nur die Daten der Komponenten, welche für den partial Submit relevant sind, zum Server übertragen. Standardmäßig ist partial Submit deaktiviert.

3.4 Vergleiche der Frameworks

Nachdem die Erweiterungsframeworks RichFaces, ICEfaces und PrimeFaces in den vorangegangen Kapiteln vorgestellt wurden, sollen diese zum Abschluss verglichen werden.

3.4.1 Verfügbare Komponenten

Im Folgenden wird die Anzahl an verfügbare Komponenten der jeweiligen Erweiterungsframeworks aufgezeigt:

- **RichFaces**
  RichFaces bietet ungefähr 40 Core-Komponenten. Jedoch liefert RichFaces ein Component Development Kit, mit dem sich leicht eigene Komponenten mit in–tegrierter AJAX-Unterstützung erstellen lassen.

- **ICEfaces**
  ICEfaces enthält ungefähr 70 Core-Komponenten und zusätzlich ungefähr 40 ACE-Komponenten. Diese ACE-Komponenten nutzen eine Mischung aus serverseitigen und clientbasierten Rendering-Techniken. Außerdem werden durch diese ACE-Komponenten der Netzwerkverkehr und die serverseitige Verarbeitung reduziert.

- **PrimeFaces**
  PrimeFaces bietet ungefähr 120 Komponenten. Dabei sind neben den Standardkomponenten auch viele Extras dabei wie zum Beispiel ein HtmlEditor oder ein Excel-Exporter. Falls diese Anzahl an Komponenten nicht ausreicht, gibt es noch die sogenannten PrimeFaces Extensions. Diese bauen auf PrimeFaces auf und können ebenfalls frei genutzt werden.

3.4.2 Dokumentation

In diesem Unterkapitel wird kurz auf den Umfang der Dokumentationen der jeweiligen Erweiterungsframeworks eingegangen:

- **RichFaces**
  RichFaces besitzt einen Online Developer-Guide. Dieser umfasst derzeit 91 Seiten, der von Release zu Release erweitert wird. Leider wird die Technologie in diesem Guide nur sehr oberflächlich erklärt. Ebenfalls gibt es sehr wenige Tutorials zur Erstellung einer Anwendung mit RichFaces.
  Zusätzlich zum Developer-Guide gibt es einen Component-Guide, in welchem die Komponenten mit Beispiel beschrieben werden. Der Component-Guide umfasst ungefähr 260 Seiten.

- **ICEfaces**
  Die ICEfaces Dokumentation ist sehr umfangreich. Sie enthält eine große Menge an Tutorials, Beispielen, Bildern und zusätzlich sogar Video-Tutorials. Insgesamt umfasst der Developer-Guide fast 400 Seiten. Dabei werden sowohl das Framework, die Technologie als auch die Komponenten erklärt.

- **PrimeFaces**
  PrimeFaces bietet mit fast 600 Seiten die umfangreichste Dokumentation. Auch in diesem Developer-Guide werden das Framework, die Technologie sowie die Komponenten mit Beispielen erklärt.

3.4.3 Kernfunktionalität

In diesem Unterkapitel wird noch auf interessante Aspekte der Erweiterungs–frameworks eingegangen:

- **RichFaces**
  Einer der interessantesten Ergänzungen von RichFaces ist der **advanced queuing mechanism**. JSF 2.0 bietet bereits einen solchen Mechanismus an, um clientseitige Ereignisse zu sequenzieren. Jedoch fehlt es diesem Mechanismus an Einstellungsmöglichkeiten. RichFaces bietet die Optionen `requestDelay` und `ignoreDupResponse`. Mithilfe von `requestDelay` kann eine Zeit in Millisekunden angegeben werden, um eine Anfrage an den Server zu verzögern. Falls in dieser Zeit eine ähnliche Anfrage erfolgt, wird die ursprüngliche Anfrage durch die neue Anfrage ersetzt. Mithilfe des Attributs `ignoreDupResponse` werden Antworten des Servers, die auf eine ähnliche Anfrage erfolgen, ignoriert. Somit werden keine unnötigen Updates beim Client vorgenommen. Eine weitere nützliche Funktionalität von RichFaces ist die clientseitige Validierung.

- **ICEfaces**
  Die interessanteste Ergänzung in ICEfaces ist der Direct-To-DOM Mechanismus. Dabei speichert ICEfaces eine Kopie der DOM, die den Zustand des Browsers in einer Baumstruktur repräsentiert, auf dem Server. Wenn der Benutzer mit der Seite interagiert, wird ein AJAX-Aufruf an den Server durchgeführt.
  Dabei wird ein neues DOM erzeugt. Dieses DOM wird mit dem zwischen–gespeicherten DOM abgeglichen. Die Unterschiede werden dann als DOM-Update an den Client zurückgeschickt. Danach werden die Änderungen im Client aktualisiert.

- **PrimeFaces**
  Die größte Stärke von PrimeFaces ist die große Anzahl an Komponenten und die vielen Add-Ons, die das Entwickeln von Webanwendungen mit JSF vereinfachen. Außerdem bietet PrimeFaces die Möglichkeit, den Status einer Anfrage mithilfe des Tags `<p:ajaxStatus>` darzustellen.

3.4.4 Performance

Um die Leistung der Erweiterungsframeworks zu testen, wurde die – in Verbindung mit JSF – am häufigsten verwendete Komponente verwendet. Dies ist eine `DataTable`-Komponente mit fünf Spalten und 100 Reihen. Der Test wurde auf einem Laptop mit einem Intel Core i5, 8GB RAM und Windows 7 Betriebssystem durchgeführt. Dabei wurde das Tool **Apache AB Stress Tool** verwendet. Die nun folgende Tabelle zeigt eine Auswertung der Ergebnisse von 5000 Anfragen:

	PrimeFaces	ICEfaces	RichFaces
Anfragen pro Sekunde (Durchschnitt) [# / sek]	33,46	11,12	28,59
Zeit pro Anfrage (Durchschnitt) [ms]	29,884	89,928	34,974
Übertragungsrate [KByte / sek]	1767,06	829,41	1329,59

Tabelle 7: Performance-Test

Die beste Performance hat nach diesem Test PrimeFaces, gefolgt von RichFaces und zum Schluss ICEfaces. Der Grund dafür könnte sein, dass ICEfaces durch den D2D-Mechanismus bei großen Dokumenten länger für die Verarbeitung braucht als die anderen zwei Frameworks.

3.4.5 Lizenzen und Support

Dieses Unterkapitel geht auf die verschiedenen Lizenzen sowie den Support der Erweiterungsframeworks ein.

	PrimeFaces	ICEfaces	RichFaces
Lizenz	Apache 2.0	Mozilla Public License , Apache 2.0 & Commercial	GNU Lesser General Public License
Community-Support	Foren, Tutorials, Blogs, Dokumentation, Wiki	Foren, Tutorials, Blogs, Dokumentation, Wiki	Foren, Tutorials, Blogs, Dokumentation, Wiki
Zusatzkomponenten / Zusatzsoftware	PrimeFaces Extensions (Open Source Komponenten-Bibliothek aufbauend auf PrimeFaces)	ICEfaces EE (kostenpflichtige Zusatzsoftware und Support)	Component Development Kit (Open Source Tool für eigene Komponenten)
Mobile-Komponenten	Ja	Ja	Ja

Tabelle 8: Lizenzen und Support

Commercial-Support:

- **PrimeFaces**
  PrimeFaces bietet die Möglichkeit, eine ELITE-License zu erwerben. Mit dieser Lizenz kann ein Entwickler – im Gegensatz zur Public-Community – auf die neuesten Features und MetroUI-Elemente zugreifen. Diese Lizenz kostet jährlich 249$ pro Entwickler. In diesem Paket ist jedoch kein Support durch PrimeTek inbegriffen. Vollen Support erhält man nur als PRO-User. Dazu gibt es jedoch leider keine Angaben zu den Gebühren. Der Entwickler muss sich deswegen direkt an PrimeTek wenden.

- **ICEfaces**
  ICEfaces verfügt über eine Enterprise Edition. Mit dieser Edition erhält der Entwickler weitere Software, Features, Online Training und vieles mehr. Jedoch ist die ICEfaces EE kostenpflichtig. Die Kosten für eine Anwendung mit Support in der kleinsten Preisklasse belaufen sich auf 1500$. Soll mehr als eine Anwendung erstellt werden, benötigt man eine sogenannte Corporate-Subscription. Mit der kostengünstigsten Variante lassen sich maximal fünf Anwendungen erstellen. Die Kosten mit Support belaufen sich dort auf 13.500$.

- **RichFaces**
  Auch bei der Verwendung von RichFaces ist es möglich Commercial-Support zu erhalten, jedoch muss man sich dafür direkt an Red Hat wenden, um an In–formationen für Kosten und Gebühren zu kommen.

3.4.6 Trend

Zum Abschluss dieser Arbeit wird noch auf die Beliebtheit der Erweiterungs–frameworks eingegangen. Die folgende Grafik wurde mit dem Tool **Google Trends** erstellt. Sie zeigt das Interesse im zeitlichen Verlauf von Januar 2004 bis Juni 2015.

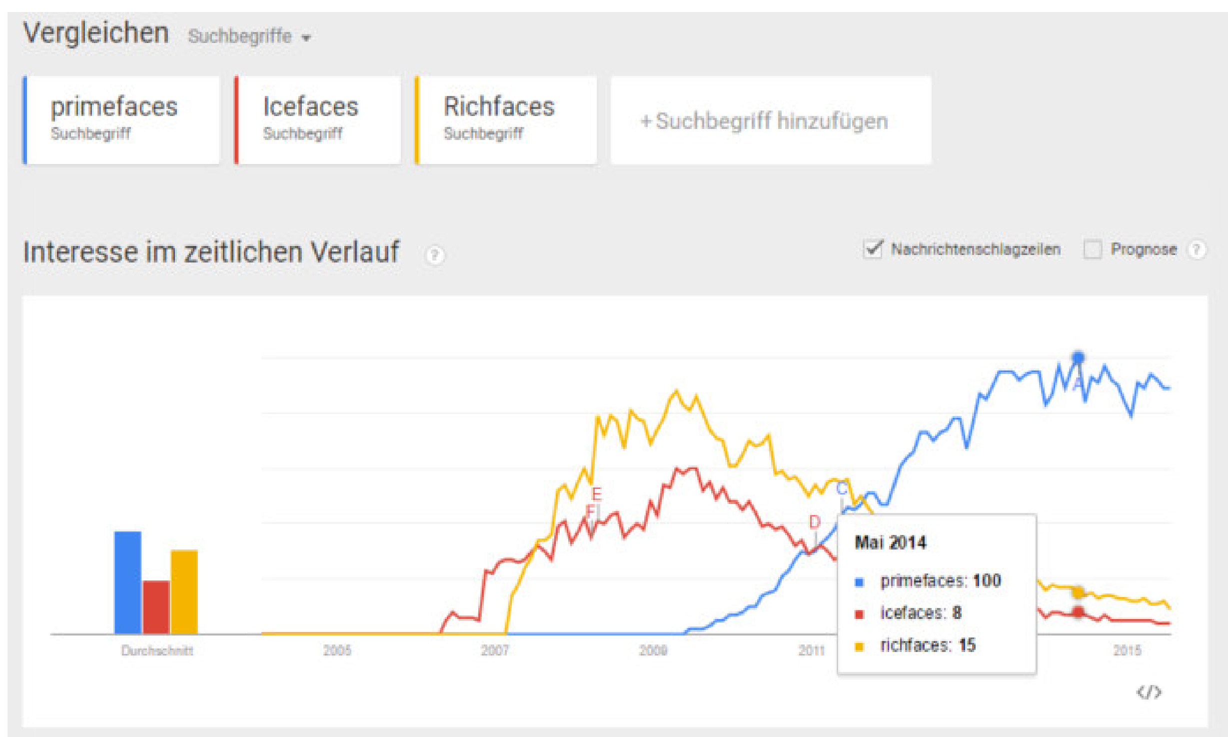

Abbildung 19: [B9] Trends für die Beliebtheit der Erweiterungsframeworks

Anhand dieser Grafik ist sehr deutlich zu erkennen, dass PrimeFaces, seit dem Jahr 2010 immer beliebter wurde und im Mai 2014 den höchsten Stand erreichte. Im Gegensatz dazu ist das Interesse an ICEfaces und RichFaces immer mehr gesunken, sodass insbesondere dem Framework ICEfaces praktisch kein Interesse mehr entgegengebracht wird.

4 Zusammenfassung

Das Ziel dieser Studienarbeit war es, zuerst einen Überblick über die Grundlagen der Webentwicklung mit dem JSF-Framework zugeben. Es wurden die Aufgaben dieses Frameworks sowie die Architektur, der JSF-Verarbeitungszyklus und der Aufbau einer JSF-Anwendung betrachtet.

Nachdem die Erklärung der Grundlagen abgeschlossen war, wurde mit AJAX eine Technik betrachtet, die es im Rahmen von JSF ermöglicht, mithilfe von JavaScript und weiteren Technologien mit einem Server zu kommunizieren und Daten zu versenden, ohne dass die Seite neu geladen werden muss. Im Zusammenhang mit AJAX wurde auch der partielle JSF-Verarbeitungszyklus erklärt.

Als Letztes wurden drei verschiedene Erweiterungsframeworks für JSF betrachtet. Hierbei war es das Ziel, die Besonderheiten der jeweiligen Frameworks zu erklären und deren Unterschied zum JSF-Standard herauszuarbeiten. Zum Schluss wurden diese drei Erweiterungsframeworks gegenseitig verglichen.

Aus diesem Vergleich konnte kein klarer Gewinner ermittelt werden, welches das beste Erweiterungsframework ist. Alle haben ihre Vor- und Nachteile. Die Entscheidung, welches Framework zum Einsatz kommt, muss der Entwickler anhand der Anforderungen an die Webanwendung selbst entscheiden. Der Trend für die Beliebtheit spricht jedoch klar für das Framework PrimeFaces.

5 Quellen

[jsf] Einführung in die Entwicklung verteilter Systeme mit Java (Kapitel 6 JavaServer Faces), Hochschule Esslingen, Frank-Müller-Hofmann, Martin Hiller, Gerhard Werner

Einführung in JavaServer Faces,
http://jsfatwork.irian.at/book_de/standard_components.html#!idx:/introduction.html:1,
Stand: 05.05.2015

Konzepte von JavaServer Faces,
http://jsfatwork.irian.at/book_de/standard_components.html#!idx:/jsf.html:2,
Stand: 08.05.2015

Standard JSF-Komponenten,
http://jsfatwork.irian.at/book_de/standard_components.html#!idx:/standard_components.html:3,
Stand: 08.05.2015

[ajax] AJAX und JSF,
http://jsfatwork.irian.at/book_de/standard_components.html#!idx:/ajax.html:7,
Stand: 14.05.2015

AJAX Einführung – Übersicht & Einleitung,
http://www.webmasterpro.de/coding/article/ajax-einfuehrung-uebersicht.html ,
Stand: 14.05.2015

AJAX – Aufbau und Ablauf,
http://www.webmasterpro.de/coding/article/ajax-ajax-aufbau-und-ablauf.html ,
Stand: 15.05.2015

[richfaces] Developer Guide,
http://docs.jboss.org/richfaces/latest_4_5_X/Developer_Guide/en-US/pdf/Developer_Guide.pdf,
Stand: 24.05.2015

Component Reference,
http://docs.jboss.org/richfaces/latest_4_5_X/Component_Reference/en-US/pdf/Component_Reference.pdf,
Stand: 27.05.2015

RichFaces Showcase,
http://showcase.richfaces.org/,
Stand: 27.05.2015

[icefaces] ICEfaces 2 Documentation,
http://res.icesoft.org/docs/v2_devguide/ICEfaces-2-Docs.pdf,
Stand: 04.06.2015

ICEfaces Overview,
http://www.icesoft.org/java/projects/ICEfaces/overview.jsf,
Stand: 05.06.2015

ICEfaces Showcase,
http://icefaces-showcase.icesoft.org/showcase.jsf?grp=aceMenu&exp=aceSuiteOverview,
Stand: 07.06.2015

[primefaces] PrimeFaces User Guide,
http://www.primefaces.org/docs/guide/primefaces_user_guide_5_2.pdf,
Stand: 08.06.2015

PrimeFaces – JSF und mehr,
http://jsfatwork.irian.at/semistatic/primefaces.html,
Stand: 10.06.2015

PrimeFaces Showcase,
http://www.primefaces.org/showcase/index.xhtml,
Stand: 10.06.2015

6 Abbildungsquellen

[B1] Klassische Webanwendung:
http://de.onpage.org/wiki/images/d/dc/AJAX_Bild.png

[B2] Datenübertragung bei klassischen Webanwendungen:
http://www.itblogging.de/wp-content/uploads/2010/08/klassische-webarchitektur.png

[B3] Webanwendung mit AJAX:
http://de.onpage.org/wiki/images/d/dc/AJAX_Bild.png

[B4] Datenübertragung einer Webanwendung mit AJAX
http://media2mult.uni-osnabrueck.de/pmwiki/fields/–wp11/m2m.d/Ajax.Asynchronitaet/media/image/prozess_ajax.jpg

[B5] Klassicher JSF-AJAX Ablauf:
http://res.icesoft.org/docs/v2_devguide/ICEfaces-2-Docs.pdf Seite 56

[B6] Ablauf mit ICEfaces:
http://res.icesoft.org/docs/v2_devguide/ICEfaces-2-Docs.pdf Seite 58

[B7] D2D Mechanismus:
http://res.icesoft.org/docs/v2_devguide/ICEfaces-2-Docs.pdf Seite 62

[B8] AccordionPanel:
http://jsfatwork.irian.at/semistatic/primefaces.html

[B9] Trends für die Beliebtheitder Erweiterungsframeworks:
https://www.google.de/trends/explore#q=primefaces%2C%20Icefaces%2C%20Richfaces&cmpt=q&tz=Etc%2FGMT-2