Bachelorarbeit

Behzad Daghigh

Konzeption und Realisierung einer
personalisierbaren J2EE WEB-Applikation
für ein Emissionsüberwachungssystem

Fakultät Technik und Informatik
Department Informatik

Faculity of Engineering and Computer Science
Department of Computer Science

Behzad Daghigh

Konzeption und Realisierung einer personalisierbaren J2EE WEB-Applikation für ein Emissionsüberwachungssystem

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas Schmidt
Zweitgutachter : Prof. Dr. rer. nat. Michael Neitzke

Abgegeben am 27. September 2007

Behzad Daghigh

Thema der Bachelorarbeit

Konzeption und Realisierung einer personalisierbaren
J2EE WEB-Applikation für ein Emissionsüberwachungssystem

Stichworte

AJAX, JSF, JSP, Servlet- Container, Tomcat, J2EE, SVG, XML,
Javascript, CSS, XHR

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Visualisierung von
Emissionswerten in einer verteilten Umgebung. Es wird eine
geeignete Methode ermittelt, Emissionswerte in einem Web- Browser
anzuzeigen, ohne, daß sich die Seite neu aufbauen muß.
Der User soll außerdem die Auswahl an Darstellungsformen
auswählen können.

Behzad Daghigh

Title of paper

Conception and implementation of a personal J2EE
web application for a emission monitoring system

Keywords

AJAX, JSF, JSP, Servlet- Container, Tomcat, J2EE, SVG, XML,
Javascript, CSS, XHR

Abstract

The report deals with the potentials of visualization of emission monitoring
values in a distributed enviroment. The goal is to examine option for
monitoring values in a web client without to refresh the whole page.
It is essential to have an efficient transfer between client/ server in order to
minimize delay.The client can also select the monitoring values to inspect
in order have a customized view.

Inhaltsverzeichnis

Abbildungsverzeichnis

Glossar

ActionScript ... Eine proprietäre Skriptsprache von Adobe auf Basis von ECMAScript

CIFS ..... Common Internet File System

DOM ..... Document Object Model

GUI ..... Graphical User Interface

HTTPS ..... HyperText Transfer Protocol Secure

J2EE ..... Java 2 Platform, Enterprise Edition

J2SE ..... Java 2 Platform Standard Edition

JAAS ..... Java Authentication and Authorization Service

JAXP ..... Java API for XML Processing

JDBC ..... Java Database Connectivity

JNDI ..... Java Naming and Directory Interfaces

JSF ..... JavaServer Faces

JSON ..... Javascript Object Notation

JSP ..... JavaServer Pages

MVC ..... Model- View- Controller

RIA ..... Rich Internet Application

Servlet ..... (Dynamische) Java Klassen in einem WebContainer

Struts	Struts, eine von der Apache Foundation entwickeltes MVC Webframework
SVG	Scalable Vector Graphics
UI	User Interface
URL	Uniform Resource Locator
WML	Wireless Markup Language
XHR	XMLHttpRequest
XHTML	extensible Hypertext Markup Language

1 Einführung in die Thematik

1.1 Einleitung

Schadstoffemissionen genehmigungspflichtiger Anlagen müssen gemäß Bundesimmissionsschutzgesetz ständig überwacht und erfasst werden. Die Fa. MAIHAK AG bietet modulare Emissionsdatenerfassungssysteme an. Ein zentraler Emissions-PC erfasst Daten von Gasanalysatoren und Prozessrechnern und führt damit die verordnungskonforme Verrechnung, Speicherung und Ausgabe durch. Dem Anwender stehen zur Visualisierung verschiedene Möglichkeiten der aktuellen und historischen Emissionsdaten zur Verfügung. Diese können derzeit direkt am Emissionsrechner oder an einem über ein Netzwerk verbundenen Arbeitsplatz-Rechner aufgerufen werden. Die Verteilung der Daten geschieht über die Windowsfreigabe von Verzeichnissen.

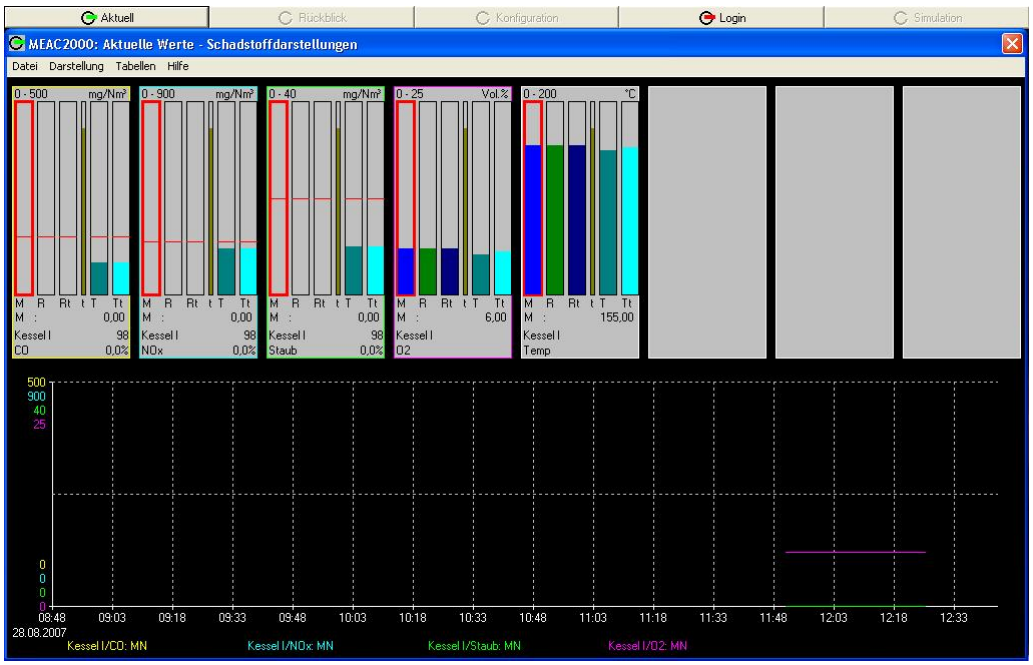

Abbildung 1.1: Auszug Prozessrechner

Zur Visualisierung der Emissionsdaten stehen Balken (oben) oder Verlaufskurven (unten) zur Verfügung. Mit einem Klick auf die graue Fläche oben kann man neue Schadstoffkomponenten zur Anzeige hinzufügen. In der unteren Verlaufskurve wird durch einen Klick auf das Diagramm eine neue Komponente hinzugefügt. Zwischen Balkendiagramm und Polygonzug besteht der wesentliche Unterschied darin, daß Balkendiagramme nur die aktuellen Werte anzeigen, während bei Verlaufskurven eine zeitliche Zuordnung zu den Emissionswerten besteht.

1.2 Motivation

Die zentrale Architektur des Emissionsrechners bringt Nachteile mit sich. Es wird immer ein Client- Programm auf dem selben Rechner installiert sein müssen, um Zugriff auf die Daten im Hauptrechner zu erlangen. Die Vergangenheit hat gezeigt, daß diese Lösung beim Kunden fehlerträchtig ist. Der Emissionsrechner basiert auf einem Windows-System, die Verteilung wurde über CIFS (Common Internet File System) realisiert. Zusätzlich sind Zugriffsrechte für die Dateifreigaben auf die Anwender zu verteilen. Jeder Anwender muß zentral auf dem Emissionsrechner angelegt sein. Dies birgt jedoch einen hohen Aufwand; eine Fehlkonfiguration der Zugriffsrechte bringt zusätzlich Sicherheitsrisiken. Bei einem Softwareupgrade muß jedes Client- System aktualisiert werden. Dies erfordert einen hohen Wartungsausaufwand.
Aus diesen Gründen setzt man Client/ Server Systeme, die auf auf Standard- Webtechnologien basieren, ein, denn die meisten Betriebssysteme enthalten einen Webbrowser. Somit entfällt jegliche Software Installation auf dem Client System. Wenn bei der Entwicklung die Kompatibilität zu allen Browsern beachtet wird, erreicht man zusätzlich Plattformunabhängigkeit. Im Gegensatz zu herkömmlichen Client/ Server System muß bei Programmänderungen nur zentral der Webserver modifiziert werden, was sich positiv im Wartungsaufwand niederschlägt. Durch die Kommunikation über Port 80 zwischen Client und Server muß man keine Firewall Probleme befürchten, da in den meisten Firmen dieser Port nicht restriktiv ist.

1.3 Ziel

Eine webbasierte Java- Lösung soll die in Abbildung 1.1 gezeigte Oberfläche im Browser ähnlich darstellen. Die Konfiguration der Anzeige bezüglich der Schadstoffkomponenten übernimmt der Anwender. Nach der Konfiguration soll ein Balkendiagramm seine Anzeige selbständig aktualisieren, daher muß sich das Diagramm dynamisch ändern können. Ein Problem ist hier das HTTP Protokoll, welches zustandslos arbeitet. Das bedeutet,

daß beim Aufruf des Webservers die Verbindung auf- und abgebaut wird. Dies macht die Steuerung der Anwendung schwierig, weil keine Informationen über den Anwender gespeichert werden, bzw. der Webserver nicht “weiß”, daß schon andere Anfragen gestellt wurden.
Eine andere Problematik bei herkömmlichen Webapplikationen ist der Neuaufbau von HTML- Seiten bei einem Anfrage/ Antwort Zyklus, welches die Interaktivität der Anwendung stört. Da dies beim Prozessrechner, der ein lokales Programm ausführt, nicht der Fall ist, muß ein geeignetes Konzept gefunden werden, um diese Problematik zu unterbinden.
Ein weiteres Thema, das zu betrachten ist, betrifft die visuellen Komponenten auf dem Emissionsrechner. Webseiten bestehen hauptsächlich aus HTML. HTML ist eine textbasierte Auszeichnungssprache zur Darstellung von Text, Tabellen und Bildern im Webbrowser. Ursprünglich nur für die einfache Übertragung auf Text und Bildern ausgelegt, eignet es sich nicht für die Darstellung von grafischen Komponenten, wie Verlaufskurven oder Balkendiagramme, weil keine derartige Funktionalität in HTML existiert. Eine Untersuchung alternativer Möglichkeiten zur Visualisierung ist hier notwendig und die Plattform, auf der die Applikation laufen soll.

2 Anforderungen an die Webapplikation

In diesem Kapitel werden sowohl die funktionalen als auch die nicht- funktionalen Anforderungen an die Webapplikation formuliert.

2.1 Funktionale Anforderungen

Die funktionalen Anforderungen treffen Aussagen über die zu erfüllenden Eigenschaften der Software.

1. Der Anwender soll sich über ein Browser- Dialog bzw. Modales Formular anmelden/ authentifizieren können.
2. Der Anwender soll sich über eine Menüzeile im Browser abmelden können.
3. In der Menüzeile soll ein Dropdown- Menü mit einem modalen Konfigurationsdialog enthalten sein.
4. Die im Konfigurationsdialog angezeigten Elemente sind: Anlage, Schadstoffkomponente und Messvariante.
5. Es existieren drei Messvarianten MW (Momentanwert), RW (Rasterwert) und TW (Tageswert)
6. Zur Ansicht der Werte stehen Verlaufsansicht, Balken- und -Tabellen zur Verfügung.
7. Graphische Elemente sind frei platzierbar, d.h. DragAndDrop fähig.
8. Nach der Auswahl der grafischen Ansicht sollen Emissionsdaten in der Hauptansicht sofort angezeigt werden.

9. Die Hauptansicht soll unter einem frei wählbarem Namen abgespeichert werden können, um diese Ansicht beim beim nächsten Anmelden wiederherstellen zu können.

Der Anwender soll sich über ein Browser- Dialog bzw. Modales Formular anmelden/ authentifizieren können.
Eine dem Anwender bekannte URL führt auf die Willkommensseite der Webapplikation. Dort wird er über ein modales Formular aufgefordert, seinen Usernamen und Passwort einzugeben. Das System prüft auf richtige Syntax, die in Absprache mit dem Administrator festgelegt wird, denn es ist mit JSF möglich für ein eigenes Format eigene Konverter zu schreiben. Gelingt die Anmeldung, wird auf die Hauptseite weitergeleitet. Wenn dem System die Anmeldung nicht bekannt ist, wird eine entsprechende Fehlermeldung ausgegeben und die Login- Seite verbleibt.

Der Anwender soll sich über ein Menüpunkt im Browser abmelden können.
Jede webbasierte Applikation sollte dem Anwender ermöglichen, sich ordnungsgemäß abzumelden. Intuitiv schließen Anwender den Browser, aber dadurch bleiben im System wertvolle Ressourcen reserviert, die die Speicherlast erhöhen. Über den Menüpunkt “abmelden” in der Menüzeile wird die Session beendet und zur Startseite weitergeleitet.

In der Menüzeile soll ein Dropdown- Menü mit einem modalen Konfigurationsdialog enthalten sein.
Vielmehr soll eine grafische Komponente im Browser angezeigt werden, die in HTML als Element nicht existiert. Wie im nächsten Kapitel zu sehen ist, wird diese Funktionalität anhand von speziellen UI- Komponenten bereitgestellt.

Die im Konfigurationsdialog angezeigten Elemente sind: Anlage, Schadstoffkomponente und welchen Messvariante.
Dieser modale Konfigurationsdialog erscheint, wenn auf den Menüpunkt “Konfiguration” geklickt wird. In diesem Formular sind die Anlagen als ListBox, die Schadstoffkomponenten als Dropdown- Menü und die Messvarianten als Checkbox zu implementieren. In diesem Formular kann der Anwender über einen Button die ausgewählte Komponente zur Ansicht hinzufügen.

Es existieren drei Messvarianten MW (Momentanwert), RW (Rasterwert) und TW (Tageswert).
Diese drei Messvarianten geben darüber Auskunft, welche Variante die Schadstoffkom-

ponente annehmen kann.

Zur Ansicht der Werte stehen Polygonansicht, Balken- und -Tabellen zur Verfügung.
Diese grafischen Elemente sollen als Ansicht für die Daten auswählbar sein. Diese Elemente werden nicht nativ durch HTML unterstützt, daher werden für diese Lösung andere Verfahrensweisen vorgestellt.

Graphische Elemente sind frei platzierbar, d.h. DragAndDrop fähig.
Um bei einer Vielfalt von einzelnen grafischen Elementen den Überblick und die Anordnung zu behalten, kann der Anwender mit einem Mausklick die grafischen Elemente verschieben. DragAndDrop Funktionalität im Browser wird durch Javascript ermöglicht, d.h. dies geschieht auf dem Client, daß bedeutet, es sind keine weiteren Vorkehrungen auf dem Server zu treffen.

Nach der Auswahl der grafischen Komponente sollen Emissionsdaten sofort angezeigt werden.
Nachdem der Anwender seine Schadstoffkomponente ausgesucht und bestätigt hat, soll sofort in der Hauptansicht mit Aktualisierung der Werte im Browser beginnen. Ein Ziel ist, daß die Aktualisierung ohne Wiederaufbau der Seite bzw. keine Störung der Interkativität auftreten darf.

Modaler Auswahldialog zum Speichern der Hauptansicht unter einem Namen, um diese Ansicht beim nächsten Anmelden wiederherstellen zu können.
Beim Menüpunkt “Speichern” sollen die auf der Hauptseite befindlichen grafischen Ansichten, also die Konfiguration der Hauptseite, auf dem Webserver unter einem selbstgewählten Namen abgelegt werden. Als Vorteil für den Anwender, um nicht immer nach der Anmeldung die Ansicht neu konfigurieren zu müssen.

2.2 Nicht- funktionale Anforderungen

Nicht- funktionale Anforderungen betreffen die Umstände, unter der die geforderte Funktionalität zu erbringen ist. Ein wichtiger Punkt, der zu beachten ist, betrifft die Sicherheit der Applikation:

- Ein unautorisierter Zugriff auf die Applikation muß verhindert werden.

- Die in Anmeldung übertragenen Formulardaten sollen verschlüsselt werden.

Ein weitere Eigenschaft, essentiell für diese Webapplikation, ist die *Performance*. Genaue Aussagen in einer verteilten Umgebung über die Performance zu treffen, ist schwierig, da hier mehrere Faktoren voneinander abhängen. In diesem Kontext nimmt man als Beispiel die Aktualisierung einer oder mehrerer Schadstoffkomponenten. Von der Anfrage bis zur Antwort spielen die Anzahl der zu aktualisierenden Komponenten im Browser, die Menge der User, die gleichzeitig sich auf dem Webserver angemeldet sind, die Auslastung des Webservers, eine Rolle.

- Die Anwendung soll “schnell” sein, daß Aktualisierungen von Komponenten die Interaktivität nicht gefährden.

2.3 Plattformanforderungen

Eine wichtige Basis betrifft die Plattform, auf der die Applikation laufen soll. Folgende Vorraussetzungen soll sie erfüllen:

- Die Plattform soll eine modernen Architekur besitzen (Wartungsfreundlichkeit, Wiederverwendbarkeit und Erweiterbarkeit)
- Unabhängige Enwicklung von Präsentation/ Anwendungslogik/ Steuerung im Web
- eine vorhandene Basisfunktionalität von Komponenten soll implementiert sein, wie z.B. Fehlermanagement, Internationalisierung, Sicherheitsmodelle, UI- Komponenten

3 Überblick Webtechnologien

In diesem Kapitel wird kurz auf aktuelle Technologien Bezug genommen, die relevant für das Thema sind.

3.1 J2EE Plattform

Java 2 Platform, Enterprise Edition (J2EE) definiert einen Standard zur Entwicklung von Enterprise Anwendungen. Diese Plattform erleichtert durch standardisierte und modulare Komponenten die Entwicklung von verteilten Anwendungen.
Sie unterscheidet sich von der Java 2 Platform Standard Edition (J2SE) dadurch, daß sie besonders auf die Entwicklung sicherer, robuster und interoperabler Enterprise-Anwendungen ausgelegt ist. Unter Enterprise-Anwendungen versteht man Anwendungen, die eine besonders hohe Performanz, Ausfallsicherheit und Transaktionssicherheit bieten. Weitere Aspekte sind Skalierbarkeit, Lastverteilung und eine grundlegende Modularität der eingesetzten Programmteile. J2EE-Applikationen laufen in Rahmen von J2EE-Applikationsservern, deren Funktionalitäten und Aufgaben weitgehend durch die J2EE-Spezifikation definiert sind. Dadurch ist eine J2EE-Applikation in der Praxis abhängig vom konkret eingesetzten Applikationsserver.
So werden Enterprise-Anwendungen zumeist von kommerziellen Firmen eingesetzt, die hohe Anforderungen haben. Die Entwicklung der Technologie J2EE ist im Laufe der letzten Jahre vorangeschritten, und deren Verbreitung am Markt hat stark zugenommen, so daß J2EE-basierte Systeme auch in kleineren Firmen und von Privatpersonen gut einsetzbar sind. Ein Beweis hierfür ist die Verfügbarkeit mehrer Open-Source- Implementierungen wie zum Beispiel in Form des JBOSS-Applikationsservers.[1]
J2EE wird durch die J2EE- Spezifikation definiert. Sie liegt aktuell in der Version 5 EE [2] vor und baut auf der J2SE in der Version 5 auf. Sie definiert eine Sammlung von APIs und Technologien, die jede J2EE-Implementierung bereitstellen muß (vgl Abbildung 3.1).
Um einige modulare Programmteile zu nennen sind die JDBC API für Datenbankzugriffe, XML- API, Web- Sevices und Sicherheitsmodelle für den Schutz von Daten in Internet-Applikationen. EJBs als Middleware- Komponente (mit Enterprise JavaBeans können wichtige Konzepte für Unternehmensanwendungen, z. B. Transaktions-, Namens- oder

Sicherheitsdienste umgesetzt werden, die für die Geschäftslogik einer Anwendung nötig sind), Java Servlet API für dynamische Webanwendungen, JSP als dynamische Skriptsprache in HTML und die XML Technologie. Besonderes Augenmerk gilt “Servlets and Java Server Pages (JSP)” in der dynamischen Webseitenentwicklung.

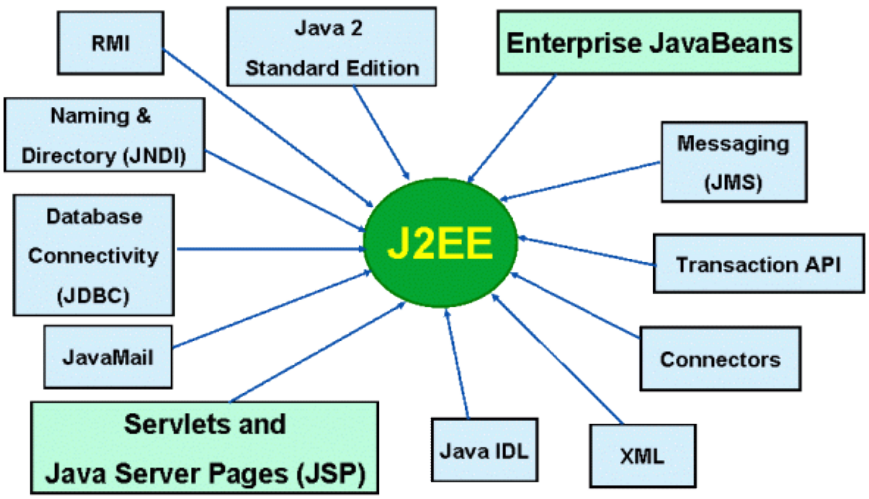

Abbildung 3.1: J2EE- Komponenten [3]

Die folgende Abbildung spiegelt die J2EE Architektur wieder:

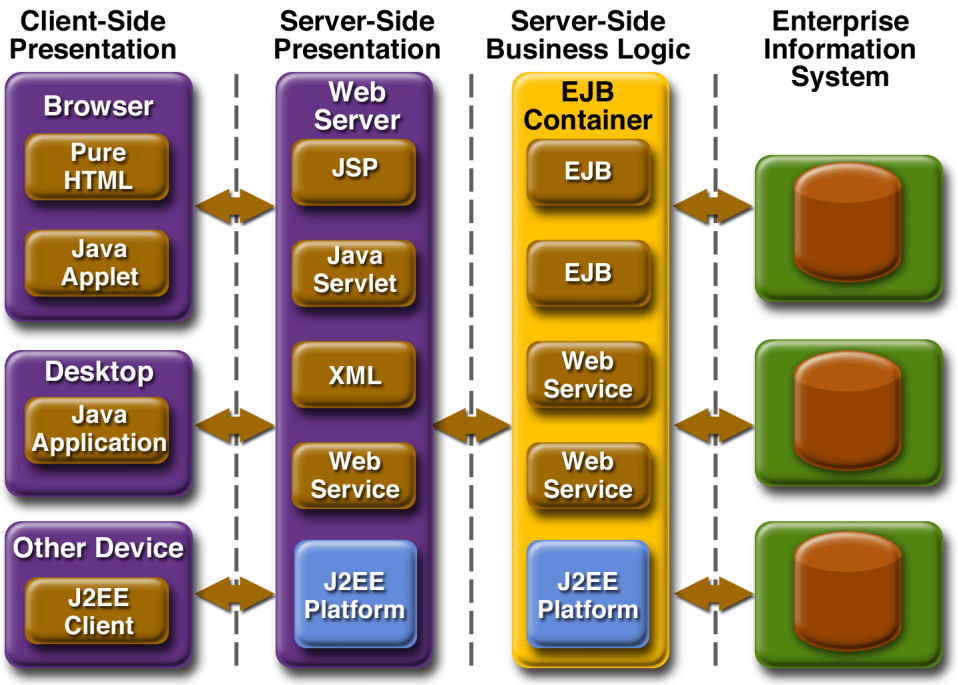

Abbildung 3.2: J2EE- Architektur [2]

Geschäftslogiken und -objekte , als eigene Komponenten, werden in EJBs gekapselt. Dynamische HTML Seiten, als Serverkomponente, werden mittels JSPs oder Servlets entwickelt. Diese Technologien werden im nächsten Abschnitt erläutert, da sie für die Bewältigung der Aufgabe ein elementarer Bestandteil sind. Ein anderer Bestandteil ist die Repräsentation auf dem Client. Sie erlaubt für die Verfolgung der Ziele mittels Browser zwei Technologien: Java- Applets oder HTML. Java- Applets sind eigenständige Programme, die vom Client heruntergeladen und lokal ausgeführt werden. Applets sind eingebettet in HTML. Diese Lösung hat zwar den Nachrteil, daß man eingeschränkt auf das Client System Zugriff hat, aber durch Swing ist es möglich, moderne und komplexe Grafikobjekte auf dem Browser zu gestalten, was allein mit HTML Sprachkonstrukte nicht machbar ist. Allerdings setzen Applets voraus, daß eine JRE auf dem Client installiert sein *muß*. Je nach Komplexität des Programms kann die Ausführungsgeschwindigkeit variieren. Da ein lokales Programm ausgeführt wird, tauchen mehrere Probleme auf. Angenommen das Applet müsse eine Verbindung zu einem verteilten System z.B. Datenbank aufbauen. Das Sandbox Konzept erlaubt nur Verbindung zu dem Server, von wo das Applet geladen wurde. Dies macht die Entwicklung von verteilten Anwendungen umständlich. Ein weiterer Nachteil wäre die aus Entwicklungssicht nicht vorhandene Abstraktion, welche die Realisierung der Anwendung erschweren würde. Bei Webbrowsern gewinnt man ein ho-

hes Maß an Abstraktion, weil Verbindungen transparent sind. Hier gilt festzuhalten, daß als einzige Option in J2EE gemäß Abbildung 3.1 nur die clientseitige Entwicklung über HTML und Javascript bleibt.

3.1.1 Dynamische Webentwicklung in J2EE

3.1.1.1 Servlets

Die Java- Servlet API [4] bietet Schnitstellen, um die Ausgabe von Java- Klassen als dynamisch generierte Webseiten an Clients zu liefern. Vorteil ist die gesamte Verfügbarkeit der Java Klassenbibiothek, welche erlaubt auf entfernte Datenbanken, XML Daten oder EJB zuzugreifen. Die Servlet- API stellt Methoden zur Verfügung, um auf Umgebungsvariablen und Formulardaten zuzugreifen. Ein einfaches Beispiel aus einem Servlet- Auszug, der die Funktionsweise beschreibt und die QuadratZahl von 1 bis 15 in HTML ausgibt:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class QuadratServlet extends HttpServlet {

 public void doGet (HttpServletRequest req,
  HttpServletResponse resp) throws ServletException,IOException {
  resp.setContentType ("text/html");
  PrintWriter out = resp.getWriter();
  out.println ("<html>");
  out.println ("<head>");
  out.println ("<title>Quadratzahlen</title>");
  out.println ("</head>");
  out.println ("<body>");
  out.println ("<h1>Quadratzahlen von 1 bis 15</h1>");
  for (int i = 1; i <= 15; i++) {
   out.println (i + " = " + i * i + "<br />");
  }
  out.println ("</body>");
  out.println ("</html>");
 }
}
```

Abbildung 3.3: Servlet- Auszug

Die Methode doGet() implementiert die Antwort, die das Servlet auf eine HTTP-GET-Anfrage geben soll. Das HttpServletRequest-Objekt enthält die Einzelheiten dieser Anfrage, wie z.B. Header, Methodenart und Parameter. Die HttpServletResponse entspricht der Antwort , die über den Webserver an den Client übermittelt werden soll.
Servlets wurden von Sun spezifiziert. Die Version der aktuellen Spezifikation ist 2.5 [4]. Implementiert werden Servlet-Engines jedoch von verschiedenen Herstellern. Eine Servlet-Engine ist eine Server Applikation, die Servlets ausführt, die Client-Anfragen an ein angefragtes Servlet weiterleitet und die Servlet-Antwort an den Client zurücksendet.

3.1.1.2 JSP

Da die Ausgabe einzelner HTML-Zeilen über println()-Befehle recht umständlich und unübersichtlich ist, wurde als alternative Lösung die Java Server Pages (JSP) vorgestellt.[5] Sie entsprechen dem Ansatz von PHP oder Microsofts ASP: An bestimmten Stellen innerhalb einer normalen HTML-Datei können Java-Anweisungen durch <% ... %> (Skript-

lets) eingeschlossen werden. JSP Seiten werden vom Webserver beim ersten Aufruf in ein Servlet umgewandelt.[6]
Ein Beispiel, das das aktuelle Datum ausgibt:

```
<HTML>
 <HEAD>
  <TITLE>Hello World</TITLE>
 </HEAD>
 <BODY>
  <H1>Hello World</H1>
    Today is: <%= new java.util.Date().toString() %>
 </BODY>
</HTML>
```

Abbildung 3.4: JSP Auszug

Grundsätzlich lassen sich JSP Seiten als HTML mit zusätzlichen JSP-spezifischen Tags und Java-Code beschreiben. Eine JSP Seite kann grob in die folgenden Elemente aufgeteilt werden:

- statischer Inhalt wie HTML
- JSP-Direktiven
- JSP-Tag-Bibliotheken (Tag Libraries)

Der statische Inhalt sind all jene Elemente, die vom Webserver in die HTTP-Response (Antwort) ohne Veränderung übernommen werden (z. B. HTML-Tags). So wäre ein normales HTML-Dokument gänzlich ohne JSP-Elemente, wie eingebetteten Java-Code oder JSP-Aktionen, eine gültige JSP. Mit JSP- Direktiven ist es möglich JavaBeans anzusprechen. Bereits die Version JSP 1.0 enthielt die Möglichkeit, Beans aus JSPs heraus ohne expliziten Java-Code zu nutzen. Die von der Spezifikation mitgelieferten Tags machten es möglich, alle Formulardaten auszulesen und einer Bean zuzuordnen. Alleine damit kam man aber nicht zum Ziel, so daß auf jeden Fall ergänzender Java-Code in Skriptlets nötig war. Einen deutlichen Fortschritt hat SUN mit der Spezifikation 1.1 für JSPs getan. Hier wurden die Custom- Tags eingeführt (Tag-Libraries). Mit diesen war es nun möglich, häufig wiederkehrende Aufgaben in eigene Custom-Tag-Bibliotheken auszulagern. Schleifen

und Verzweigungen sind zwei der wohl meist genutzten Beispiele. Schon bald gab es Bibliotheken anderer Hersteller mit häufiger genutzten Funktionalitäten. Auf Skriplet Code konnte man trotzdem nicht verzichten.

3.1.2 Verteilte Anwendungsarchitekturen im Web

In diesem Unterabschnitt werden die am häufigsten eingesetzten Webarchitekturen diskutiert , denn die Auswahl einer geeigneten Architektur ist maßgeblich für eine einfach zu wartende und erweiterbare Anwendung.

3.1.2.1 Servlets und JSP Model 1 Architektur

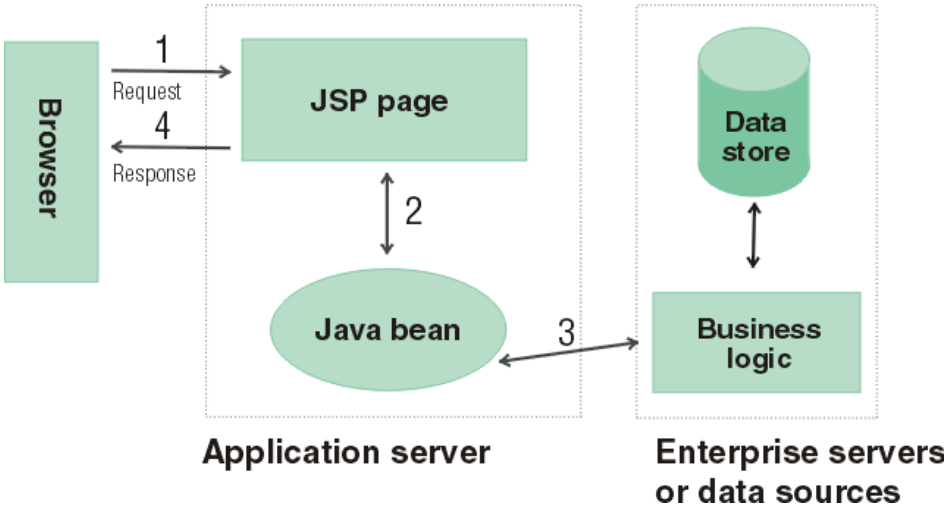

Abbildung 3.5: JSP Model 1 [7]

Die Abbildung zeigt den Vorgang eines kompletten Zyklus. Charakteristisch für die Model 1- Architektur ist, daß für nahezu jede Seite ein JSP/ Servlet verwendet wird und Links oder Weiterleitungen in der Anwendung ebenfalls durch sie angegeben sind. Idealerweise würde die Logik außerhalb von JSP/ Servlet angestoßen. In der Praxis wird jedoch aus Komfort Anwendungsbestandteile direkt innerhalb von JSPs (Skriptlets) oder in Servlets verlagert. Selbst bei einer sauberen Trennung von Darstellung- Code und Logik- Code besteht immer das Problem, daß in Servlets HTML -Markup erzeugt werden muß. Dies ist sehr unflexibel. Selbst bei reiner JSP- Seiten Programmierung wird es schnell unübersichtlich, da sich Java- Code mit HTML- Markup vermischt. Schnell wird hier klar, daß eine Trennung zwischen Code, der nur mit der Oberfläche zu tun hat, und Code, der die Logik enthält, sinnvoll ist.[8] Ein Anwendungsmuster, das sich etabliert hat, nennt sich MVC, im Webbereich in abgewandelter Form, JSP Model 2 .

Servlets haben dennoch ihre Berechtigung, wie im weiteren Verlauf dieses Kapitels deutlich wird, denn sie fungieren als FrontController für JSP Model 2 Webanwendungen.

3.1.2.2 Ursprüngliches MVC

Die Festlegung auf eine seitenzentrierte Sichtweise aus den Model 1- Architekturen stößt bei größeren und modernen Anwendungen schnell auf ihre Grenzen. Die Entkopplung zwischen Anwendungslogik und Präsentation wird seit den achtziger Jahren in GUI- Anwendungen anhand MVC eingesetzt. MVC löst das Problem, Daten verschiedenartig zu präsentieren. Abbildung [9] demonstriert das MVC- Pattern.

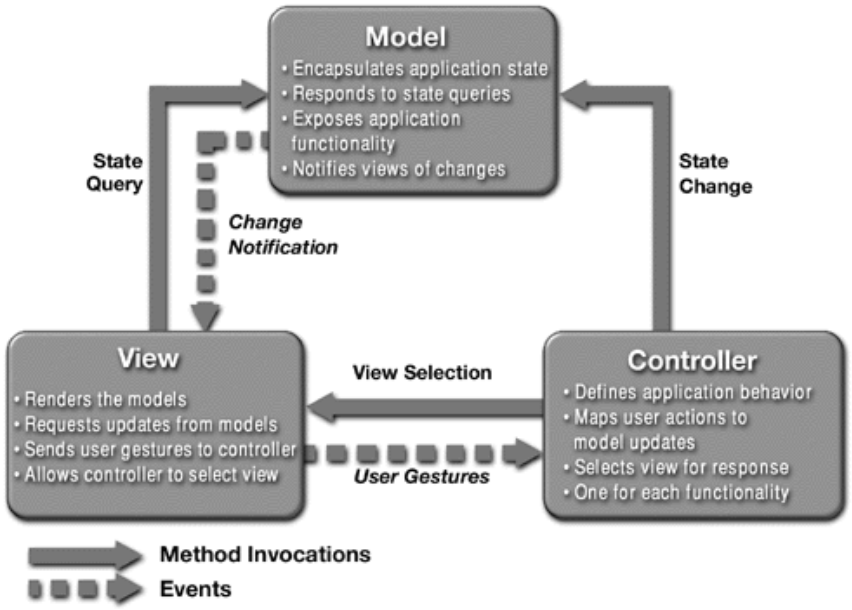

Abbildung 3.6: MVC- Pattern [9]

Das *Model* enthält den Zustand der Daten. Es kennt weder den Controller noch die View. Es kann aber über das Observer- Pattern registrierte View Objekte bei Modelländerungen benachrichtigen.
Die *View* ist für die Darstellung des Models zuständig. Für das Model kann es mehrere Views geben. Dies hat den Vorteil, für verschiedene Clients (z.B SWING, HTML, PDF) andere Ausgaben zu generieren. Bei Zustandsänderung des Models kann die View sich aktualisieren.
Der *Controller* nimmt die Eingaben des Anwenders entgegen. Er steuert oder leitet die Anfragen ggb. an geeignete Funktionalitäten des Models weiter.

3.1.2.3 JSP Model 2

Das ursprüngliche MVC- Pattern in Webanwendungen ist so nicht möglich. Die Ursache liegt in der verteilten Kommunikation, bzw. dem HTTP- Protokoll. Das Model kann die View in Webanwendungen nicht informieren. (vgl. Kapitel 1.3 Ziel) Der Server kann nicht von sich aus aktiv werden. Aufgrund dieses Nachteils wurde ein angepaßte Variante des MVC geschaffen. Die Aufteilung sieht folgendermaßen aus: Ein zentrales FrontController- Servlet fungiert als *Controller*. Der Controller nimmt die Eingaben des Anwenders entgegen (z.B. aus einem Formular), wandelt sie in die entsprechenden Datentypen für das Model um, und leitet diese Anfrage an einem bestimmten Handler- Servlet weiter. Dieses Servlet ruft für das Model(JavaBean) aus der Anfrage die richtige Methode(Business Logic) auf. Nach der Rückkehr der Methode wandelt das Handler- Servlet die Daten für die (View) in ein geeignetes Format um. Der Controller entscheidet außerdem, welche View für die Antwort ausgewählt wird.[10]
Das Model entspricht weitgehend dem originalen MVC. Webseiten, also die View, können sich nicht als Observer beim Model registrieren. Die Daten werden weiterhin in JavaBeans gekapselt. Der Controller kümmert sich um die Ablaufsteuerung und die Initialisierung der Beans.
Die View kann nur den aktuellen Status des Models wiedergeben (vgl. Abbildung [7]).

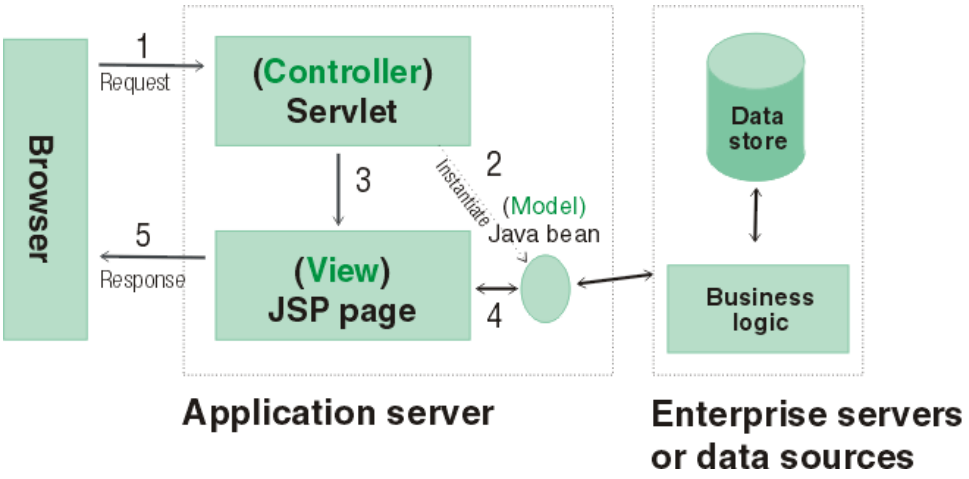

Abbildung 3.7: JSP Model 2 [7]

3.1.2.4 Ergebnis

Der Vorteil bei der JSP Model 2- Architektur ist, daß die Ablaufsteuerung zentralisiert ist und die JSP Seiten sich nicht mehr gegenseitig referenzieren, wie im JSP Model 1. Zwar ist das JSP Model 2 komplex aufgebaut, gewinnt aber an Wiederverwendbarkeit

und Erweiterbarkeit.[8] Eine Ausprägung auf dieser Basis ist JSF.

3.2 AJAX

AJAX steht für Asynchronious Javascript and XML. Der Ursprung von AJAX ist Microsoft. Zu Zeiten des Internet Explorer 5 hat das Internet-Explorer-Team eine Technologie in die Browser integriert, um im Hintergrund HTTP-Anfragen abzusetzen und die Rückgabe auszuwerten. Die Web-Schnittstelle von Outlook benötigte HTTP-Anfragen im Hintergrund, um beispielsweise ohne permanentes Neuladen zu prüfen, ob es schon neue Mails gibt. Dieses Feature wurde als ein ActiveX-Control namens XMLHttpRequest [11] (XHR) implementiert. Mittlerweile unterstützen alle Browser dieses Feature.[12]
XMLHttpRequest kann lediglich eine HTTP-Anfrage an einen Webserver schicken und die Rückgabe auswerten. Die Schwierigkeit besteht darin, die Server-Rückgabe auf der Seite darzustellen. Dazu wird JavaScript, in Verbindung mit Document Object Model (DOM) [13] , verwendet. DOM ist seit 1998 ein Standard des W3C und wurde seitdem mehrfach aktualisiert (aktuell DOM Level 2). DOM ist ein plattform- und sprachunabhängige Schnittstelle, welches Anwendungen dynamischen Zugriff auf Inhalt und Struktur von Dokumenten erlaubt.
Ein XHTML-Dokument wird zu einem hierarchischem Baum abgebildet, mit dem man mittels JavaScript darauf zugreifen kann. Jedes XHTML-Element und auch jeder Text, der nicht von Tags umgeben ist, wird zu einem Knoten (engl. node) in dem Baum. Elemente zwischen einem öffnenden und dem dazugehörigen schließenden Tag sind Kindknoten (engl. child nodes).
Der Ablauf wird bei ein XHR Anfrage skizziert:

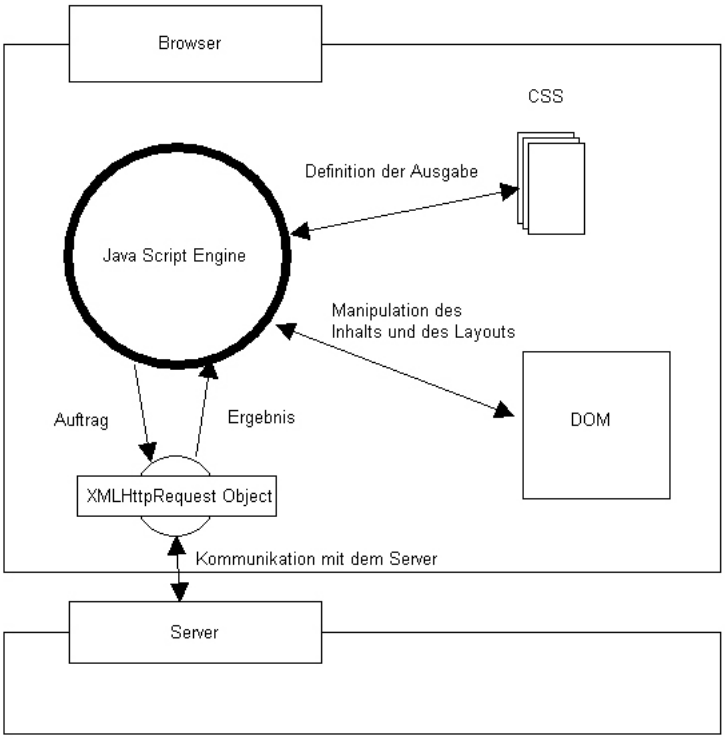

Abbildung 3.8: AJAX- Zyklus [12]

1. Der Browser schickt via Javascript ein Anfrage zum Server (XHR)
2. Der Server bearbeitet die Anfrage im Hintergrund
3. Die partielle oder komplette Seite wird als Antwort mittels XML zurückgesendet
4. Der Browser aktualisiert mittels Javascript seinen DOM Baum

3.3 Andere proprietäre Technologien

Außerhalb von J2EE existieren noch weitere RIAs (Rich Internet Application), die kurz erwähnt werden, weil sie Komponenten enthalten, die speziell für Visualisierung entwickelt worden sind. Eine proprietäre Lösung von Adobe heißt Flex.
Wichtige Komponenten sind die Programmiersprache MXML, die deklarativ aufgebaut ist, und ActionScript, die befehlsorientiert arbeitet. Der Flex- Compiler nimmt diese MXML und ActionScript und generiert eine Flashdatei daraus.Der LifeCycle Dataservice ist die Serverkomponente.[14] Über diesen Dienst kann man andere (z.B. Java) Serveranwendungen ansprechen. Die 3. Version wird als Open Source in der 2. Jahreshälfte 2007

verfügbar sein.
Da Adobe Flex eine propriertäre Lösung ist, wird eine Entwicklung/ Konzept auf dieser Basis nicht entstehen. Es muß allerdings zugegeben werden, daß der Flashplayer als Browser- Plugin sehr weit verbreitet ist, aber die genannten Vorteile von J2EE als standardisierte Architektur überwiegen. Besonders durch den Kostenvorteil vieler Opensource Lösungen bietet es einen guten Einstieg.

3.4 Bewertung

Die dynamische Webseitenentwicklung von J2EE ist eine geeignete Plattform für dynamische Webseiten, da sie dem Entwickler viel Arbeit durch vorhandene, modulare und standardisierte Komponenten abnimmt. Bei Nutzung von Standardkomponenten bietet J2EE Vielfalt und beschleunigt die Entwicklung (vgl Abbildung 3.1). Servlet- Programmierung bietet den Vorteil der Plattformunabhängigkeit bei der Auswahl der Servlet- Engine. Die Servlet- Engines sind austauschbar, wenn sie die benötigte Servlet- Spezifikation erfüllen. Die Servlet- API umgeht die Problematik des zustandslosen HTTP- Protokolls und Servlets, in dem es eigene Methoden bereitstellt, die Sessions implementieren. Doch der größte Nachteil überwiegt aus architektonischer Sicht; die Vermischung von Anwendungs- und Präsentationslogik, die normalerweise so weit wie möglich vermieden werden sollte. Somit ist es nur schwer möglich das Design einer Internetanwendung einem Webdesigner und die Implementierung der Anwendungskomponenten einem Entwickler zu überlassen, da die Darstellung der Internetseiten direkt innerhalb des Servlets erstellt werden müssen (vgl. Abbildung 3.1.1.1). Hier schafft JSP ebenfalls keine Abhilfe, JSPs sind schwerer zu „lesen“ als Servlets, da HTML-Quelltext und Java-Code noch mehr vermischt sind als bei Servlets (vgl. Abbildung 3.1.1.2). Die Problematik bei Servlets existiert in JSP weiterhin , nur wird sie in HTML verlagert. Diese Nachteile machen die Wartung der Anwendung schwierig, daher sind sie nur für kleine Projekte geeignet. Für größere Projekte eignen sich hier Web- Frameworks, die versuchen diese Nachteile zu eliminieren bzw. auf einer höheren Ebene zu abstrahieren. Einer dieser Frameworks ist aufgrund der Nachteile von Servlets und JSPs entstanden und nennt sich JSF (JavaServer Faces). Genauer wird im nächsten Kapitel darauf eingegangen bzw. die Vorteile dieses Standards erläutert.

Um das Ziel “Aktualisierung der Emissionsdaten ohne permanenten Seitenneuaufbau ” aus der Aufgabenstellung zu lösen, ist eine AJAX Engine notwendig. Die zentrale Frage lautet: Soll auf eine geeignete Bibliothek zurückgegriffen oder XHR von “Hand” im-

plementiert werden? Die Optimierung per “Hand” gestaltet sich in Javascript durch seine lose Typisierung als schwierig und fehleranfällig. Bedingt durch verschiedene Browserimplementierungen sind redundante Entwicklung von Methoden unausweichlich und aufwändig. Deshalb ist es wünschenswert, ein AJAX -Framework zu benutzen, der die genannten Probleme verbirgt und sich in die Webarchitektur nahtlos einfügt. Es existieren viele freie Lösungen (z.B Dojo, Prototype, ScriptAculo), aber die Komplexität und Fehleranfälligkeit hinter Javascript bleibt dennoch bestehen. Durch JSF haben Hersteller verstärkt begonnen, AJAX Funktionalität in ihren eigenen Implementierung transparent zu unterstützen.

4 JSF

4.1 Was ist JSF?

JSF ist ein standardisiertes, serverseitiges User Interface (UI) Framework [15]. Es basiert auf der JSP Model 2 Architektur. Es beruht auf den Erfahrungen, die mit *Struts* gemacht wurden. Seit Java EE 5 ist es ein Bestandteil der J2EE Spezifikation.
Durch JavaServer Faces (JSF) wird es möglich, die unterschiedlichen Aufgaben wie Präsentation und Logik voneinander zu trennen. Sowohl die Darstellung (View), als auch die Navigation zwischen den Seiten und die Fehlerbehandlung (Controller), werden getrennt voneinander behandelt. Dabei werden so genannte User-Interface Komponenten bereitgestellt. Für die Ausgabe der Komponenten werden wiederum RenderKits benötigt. RenderKits sind Implementierungen, die JSP Tags auf JSF Seiten in HTML ausgeben. Gemäß der Model 2 Architektur ist das RenderKit austauschbar, damit andere Ausgaben außer HTML erzeugt werden können, z.B Wireless Markup Language (WML) . JSF JavaServer Faces stellt eine Auswahl von Standardkomponenten zur Verfügung, wie Ein- und Ausgabekomponenten, und ein Standard-Render-Kit für eine HTML-Ausgabe in einem Browser. Darüber hinaus sieht das Framework Erweiterungen vor. So ist es möglich, eigene UI Komponenten zu entwickeln und auch eigene Render-Kits zu implementieren, die eine andere Ausgabe als HTML erzeugen.
JSF generiert mittels JSP oder Servlets HTML- Seiten und schickt sie zum Client. Browserspezifische Probleme wie Darstellung beim Internet Explorer oder Mozilla Firefox werden durch JSF abgenommen.
Manche Webapplikationen basieren auf mehrere Formularseiten, die auszuwerten sind. Wie schon erwähnt, bietet das zustandslose HTTP- Protokoll keine Möglichkeit, die Anwenderdaten zu persistieren; JSF unterstützt diese und andere Eigenschaften.[10]
Eine Auflistung der Merkmale:

- Verwaltung der Zustände über mehrer Seiten über einen Komponentenbaum
- Browserspezifische Generierung von HTML
- Serverseitige Formularüberprüfung durch Standard- Validatoren

- Einheitliche Ereignisbehandlung durch swingähnliche events
- Fehlerbehandlung durch messages
- Vorhandene UI- Komponenten, die erweiterbar sind.
- Internationalisierung durch Ressource Bundles

4.1.1 JSF Aufbau

Die Komponenten, die JSF nutzt, bestehen aus mehreren Bestandteilen. Sie umfaßt das klassische FrontController- Servlet, die speziellen JSF- Komponenten Validatoren, Konverter, JavaBeans(Backing Beans), Event- Listener, UI- Komponenten und Renderer. Die Navigation mit ihrer beschreibenden Definition ist von der Anwendungslogik entkoppelt. Der *Controller* setzt sich somit aus mehreren Komponenten zusammen. Zentral ist hier das FacesServlet. Dieses ist für die Verwaltung des JSF- Lifecycles eines Requests verantwortlich; der genaue Ablauf wird im nächsten Abschnitt erklärt. Der aktuelle Zustand der Request- Verarbeitung und Response- Generierung wird pro Anfrage in einem FacesContext- Objekt verwaltet. Die Validatoren und Konverter, die die Plausibilität bei Standardeingaben (z.B. Zahlen, Strings, Datum) übernehmen, sind dem Controller ebenfalls bekannt
Die Event- Listener registrieren sich am Controller. Listener und Actions sind spezifisch an UICommand- Komponenten (Links, Buttons) gebunden und müssen für die jeweilige Aktion geschrieben werden. Diese reagieren auf Status- Änderungen (ValueChangedEvent) oder auf Ereignisse in Form von Klicks (ActionEvent). Der NavigationsHandler, welcher zentral über eine Datei (facesConfig.xml) konfiguriert wird, wird ebenfalls vom Controller ausgewertet. Nachdem eine Action- Methode einen Rückgabewert zurückliefert, wird die nächste View ausgewählt.
Wie auch bei Struts wird die *View* über JSPs programmiert. Das Besondere aber an JSF sind die UI- Komponenten. Diese bestehen aus JSP- Tags. Die Tags dienen zur Einbindung von visuellen oder nicht- visuellen Komponenten in eine JSF- Seite. Die UI- Komponenten müssen bestimmte Schnittstellen implementieren, um z.B. Zustände, Renderer verwalten zu können.
Das *Model* setzt sich in JSF aus JavaBeans zusammen und zum anderen aus der Geschäftslogik der Anwendung zusammen. Beans können an UI-Komponenten gebunden (Method- and ValueBinding), und mit geänderten Werten eines Formulars befüllt werden. Ein umstrittener Punkt in der Entwicklungsszene bei den Beans ist die Gestaltung der Anbindung an die Geschäftslogik. Ein direkter Aufruf der Logik in einer Bean ist architektonisch gesehen kein gutes Design, aus dem Grund wird im weiteren Verlauf noch ein

bekanntes Pattern vorgestellt.

4.1.2 JSF Lifecycle

Eine Besonderheit bei JavaServer Faces ist die genaue Definition eines Lebenszyklus, den alle Anfragen durchlaufen. Dieses umfasst insgesamt sechs Teilschritte, bspw. Schritte zur Wiederherstellung des Komponentenzustands, Validierung, Modellaktualisierung und das Rendern (HTML- Code) der View (vgl. Abbildung JSF- Lifecycle [16]).

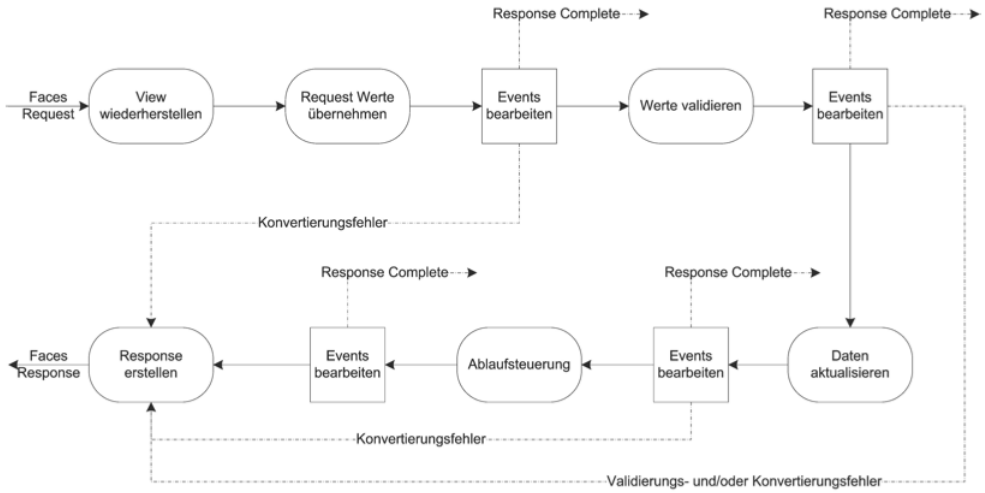

Abbildung 4.1: JSF- Lifecycle

Die einzelnen Phasen werden im folgenden erläutert:

View wiederherstellen:
Diese Phase generiert den Komponentenbaum für die Seite, falls sie nicht existiert. Im Komponentenbaum werden die UI- Komponenten einer Seite verwaltet.
Request Werte übernehmen:
In dieser Phase wird jedem Knoten des Komponentenbaums der neue Werte aus dem Request zugewiesen und ggb. auf Konvertierungsfehler überprüft. Falls Fehler aufgetreten sind, wird dies in FacesMessage- Queue als Nachricht abgelegt und in die letzte Phase gesprungen.
Werte validieren:

In dieser Phase werden alle Validierungen aus dem Komponentenbaum verarbeitet. Der lokal gespeicherten Werte werden auf Gültigkeit überprüft. Die Validierungsregeln können vordefiniert sein, oder vom Entwickler selbst definiert werden. Bei einem Fehler wird eine FacesMessage erzeugt und zur letzten Phase gesprungen, wo die Fehlermeldung auf der aktuellen Seite angezeigt wird.
Daten aktualisieren:
In dieser Phase werden die Werte der Komponenten in Java Objekte kopiert, d.h. die Properties der JavaBeans werden aktualisiert. Dabei können allerdings Konvertierungsfehler entstehen, da Request Parameter immer Strings sind und Objektvariablen jeden beliebigen (gültigen) Java Typ wiederspiegeln können. Tritt ein Konvertierungsfehler auf, wird direkt in die Render Response Phase gesprungen.
Ablaufsteuerung:
In dieser Phase werden alle Events (die Geschäftslogik) abgearbeitet. Die Navigation auf die nächste View wird hier entschieden.
Response erstellen:
In dieser Phase wird der Antwortkomponentenbaum erstellt und gerendert. Die Daten des Komponentenbaums werden mit Hilfe der Servlet- API in einem HTTP-Session Objekt auf dem Server gespeichert oder auf dem Client mittels Cookies oder versteckten Formularfeldern.

4.2 Grafische UI- Probleme

Im Gegensatz zur Tabellenansicht, welche als JSF- UI enthalten ist, existieren keine grafischen UI in JSF für Verlaufskurven oder Balkenansicht. Es existieren zwar freie Implementierungen von JSP-Custom Tags in JSF, aber diese JSF Komponenten sind nicht ajaxfähig und werden auf dem Webserver generiert. Der Webserver konstruiert anhand von Beans die Balkenansichten oder Verlaufskurven. Es werden Bilder (jpgs) zum Client geschickt. Dies ist nicht nur eine Belastung auf dem Server, sondern es müßte periodisch geschehen. Dies ist sehr ineffizient. Der Nachteil dieser Lösung ist die schlechte Skalierbarkeit. Die Hauptarbeit wird auf dem Server verlagert. Es wird nach einer Lösung gesucht, welches ein Standard ist und am besten die Visualisierung komplett auf den Client verlagert, um die genannten Anforderungen der “grafischen Ansichten” und der Performance zu erfüllen.

4.3 Vergleich zu Struts

Die JSF- Vorgehensweise weicht von Struts deutlich ab. Struts ist in erster Linie ein Framework für Webanwendungen, welches die Formularverarbeitung und die Delegation der Business- Logic erleichtert. Mit Hilfe von Struts wird dem Entwickler ein Hilfsmittel zur Verfügung gestellt, welches in erster Linie die Ablaufsteuerung unterstützt. Struts entscheidet anhand einer Benutzeranfrage und deren übermittelten Daten, welche JSP-Seite aufzurufen ist. Dabei wird jedoch die Darstellung vom Framework nicht unterstützt. Für die Darstellung wird in der Regel nur JSP genutzt. Die bekannten Nachteile sind genannt worden.
In JSF stehen die Komponenten in Vordergrund. JSF ähnelt eher der ereignisbasierten GUI- Entwicklung in Java. JSF bietet eine große Anzahl von Komponenten, die von SUN in der Standardimplementierung und von anderen Herstellern implementiert werden. Hervorzuheben ist die Erweiterung von JSF- Komponenten. Durch sie ist es möglich, eigene Komponenten zu entwickeln. Durch Standardvalidatoren, die Plausibilitätsprüfungen für Eingaben übernehmen, wird die Softwareentwicklung beschleunigt. Das Fehlermanagement ist im JSF durch einfache Textausgaben beim Anwender einheitlich. Durch “Ressource- Bundles” ist eine Internationalisierung der Anwendung möglich. Aufgrund dieser Vorteile dient dieses Framework als Plattform für die Entwicklung der Webapplikation, weil sie die Anforderungen an ein modernes Framework besser als Struts erfüllt. (vgl. Kapitel 2.3)

5 Konzeptausarbeitung

5.1 JavaBeans in der Applikation

Beans dienen zur Datenhaltung der Webapplikation. Die Beans werden aus den formulierten Anforderungen erfaßt.

1. **LoginBean**: Verwaltet die Userinformation (Username, Passwort, Rechte).
2. **PlantConfigBean:** Verwaltet die Anlagen, Schadstoffkomponenten
3. **DragAndDropBean:** Verwaltet die Postionsdaten einer grafischen Komponente
4. **StartUpdateBean:** Legt fest, ob mit der Aktualisierung begonnen wird
5. **PolygonBean:** Verwaltet die aktuellen Daten der Polygonansicht
6. **BalkenBean**: Verwaltet die aktuellen Daten der Balkenansicht
7. **DataTableBean:** Verwaltet die aktuellen Daten der Tabellenansicht

Diese Beans enthalten zusätzlich Geschäftslogiken, die Daten bei einer Anfrage abrufen. Die Gestaltung dieser Events aus der Bean bedarf einer besonderen Betrachtung, da JSF keine Komponentenarchitektur in Form von EJBs vorsieht. Daher wird ein Pattern vorgestellt, das dieses auf Klassenebene löst.

5.2 Das Facade Pattern

Wie aus der Abbildung 3.7 hervorgeht, muß noch die Anbindung der Geschäftslogik an die JavaBeans erörtert werden. In JSF erfolgen durch Events in Form von “actionlistener”. Man könnte die Events direkt aus den Beans heraus aufrufen, doch wird dies als schlechtes Klassendesign angesehen, (vgl. Kapitel JSF Aufbau 4.1.1) da die Komplexität und Abhängigkeiten der Bean- Klassen untereinander erhöht wird. Eine große und unübersichtliche Bean-Klasse ist die Konsequenz. Hier hilft das Facade Pattern den Zugriff auf diese Klassen zu vereinfachen. Die Facade- Klasse als zusätzliche Indirektionsstufe zur

Bean erleichtert den Austausch von Funktionalitäten, ohne die Bean ändern zu müssen [17]. Das Facade Pattern fördert die lose Kopplung zwischen Klassen und ein einfache Schnittstelle zur Geschäftslogik verbirgt die darunterliegenden Komponenten.
Als Beispiel ist die Datenbankverbindung (DBWrapper) zu nennen, die gewiße Funktionalität in den Beans (PolygonBean, BalkenBean, DataTableBean) enthält. Um den Zugriff auf die verschiedensten Klassen zu vereinfachen und zu verbergen, wird eine einfache Schnittstelle (PlantConfigFacade) geschaffen, die aus der Bean PlantConfig aufgerufen wird, d.h. die PlantConfig delegiert die Funktionalität an PlantConfigFacade weiter. Sie implementiert die Funktionalität und ruft Datenbank-Klassen und andere notwendigen Klassen auf. Durch das Facade Pattern wird auch auf Klassenebene die Schichten der multi/ tier Architektur voneinander getrennt. Das untersteicht nochmal den Austausch von Komponenten.

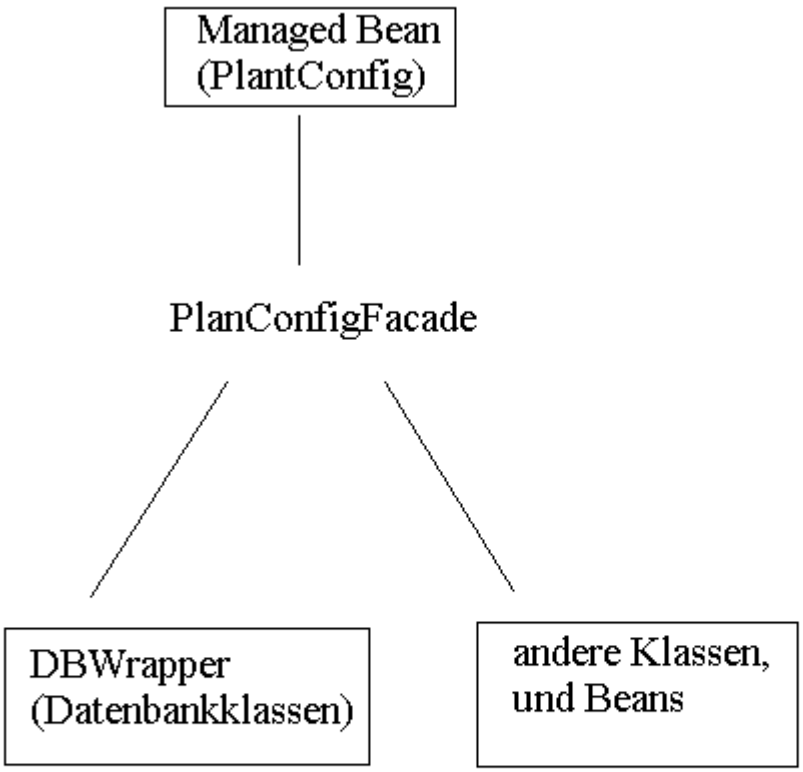

Abbildung 5.1: Facade- Pattern

5.3 JSF RI

Die aktuelle JSF Spezifikation ist 1.2_04. JSF RI ist die von SUN implementierte Version. Sie bietet einen Satz an Standard UI- Komponenten, die als JSP- Tags (jsf_html) in eine HTML- Seite angebunden werden können. Über (jsf_core) Tags werden übergreifende Funktionalitäten wie Validatoren , Konverter und Internationalisierung eingebunden. Jeder Tag hat seine eigenen, spezifischen Attribute, die das Verhalten und Aussehen steuern. Über Backing Beans kann man diese Attribute an Klassen (ValueBinding) binden. Einfache Texteingabe und Ausgabe werden unterstützt, genauso wie z.B. (RadioButtons, Checkboxes und ListMenu), welche für den Konfigurationsdialog verwendet werden, letztendlich werden die JSP -Tags vom Renderer in HTML Elemente umgewandelt.
In JSF werden 2 Arten von Events unterschieden, *action events* und *value- changed events*. Action Events werden über Buttons oder Links ausgelöst. Value- changed events werden dann ausgelöst, wenn sich der Wert einer Input- Komponente verändert.
Allerdings existieren keine Komponenten für modale Formulare, Menüleisten mit Menüpunkte. Abhilfe schaffen hier andere Hersteller, die erweiterte Komponenten für JSF entwickeln. Einer davon ist RichFaces als Opensource von JBOSS.

5.4 RichFaces

RichFaces ist eine reine Komponentenbibliothek zur Unterstützung weiterer UI- Komponenten und daher auf eine Implementierung einer JSF Spezifikation angewiesen. In diesem Fall wird die JSF RI 1.2 verwendet.
RichFaces erweitert auch die AJAX4JSF Bibliothek um AJAX- Funktionalität.[18]
Ein Vorteil ist, daß es keine Probleme mit anderen 3rd- party JSF- Komponenten anderer Hersteller gibt. Laut Hersteller ist RichFaces kompatibel zur JSF RI 1.2. Erwähnt sei nochmal, daß JSF 1.2 eine Spezifikation, die Implementierung ist von Hersteller zur Hersteller verschieden. Im Verlauf der Entwicklung wurden auch andere Implementierungen von anderen Herstellern ausprobiert, die sich leider miteinander nicht vertrugen. Dies hängt damit zusammen, daß die Renderer (JSF Ausgabe nach HTML) anders waren als die Ausgaben von JSF RI 1.2.
Die Auswahl an Komponenten in RichFaces ist flexibler und umfangreicher als die JSF RI. Ein weiterer Vorteil sind Skins, d.h. das Gesamtaussehen kann beeinflußt werden. Eine “corporate identity” nach unternehmensspezifichen Merkmalen läßt sich einfach realisie-

ren. Man kann eigene Skins kreieren oder vorgefertigte nutzen.
Modulare Formulare und Tabellen, die sich per AJAX selbst aktualisieren, werden genauso unterstützt wie Menüzeilen, Dropdown- Menüs und noch vieles mehr. Aus diesem Grund wird diese Bibliothek zusätzlich mit JSF RI eingebunden.
Die Anbindung ist einfach und es müssen selbst keine weitere Einstellungen am Webserver oder in der web.xml vorgenommen werden.

5.5 AJAX4JSF

5.5.1 Problem

Einer der Augenmerke dieser Arbeit ist die AJAX Funktionalität, denn die graphischen Elemente sollen sich selbstständig ohne Seitenaufbau aktualisieren können. Generell kann eine weit verbreitete Opensource Lösung genutzt werden, aber Javascript als typenlose Sprache gilt als schwer wartbar und fehlerträchtig. Der Lernaufwand ist ebenfalls hoch. Um Entwicklungszeit zu sparen, ist eine fertige Engine wünschenswert, die mit JSF harmoniert und Javascript verbirgt. Einer dieser Lösung nennt sich Ajax4jsf, ebenfalls von JBOSS.[19]

5.5.2 Lösung

Ajax4jsf ist ein Opensource Framework welches AJAX Funktionalität in bestehende JSF-Applikationen integriert, ohne daß der Entwickler mit Javascript arbeiten muß.

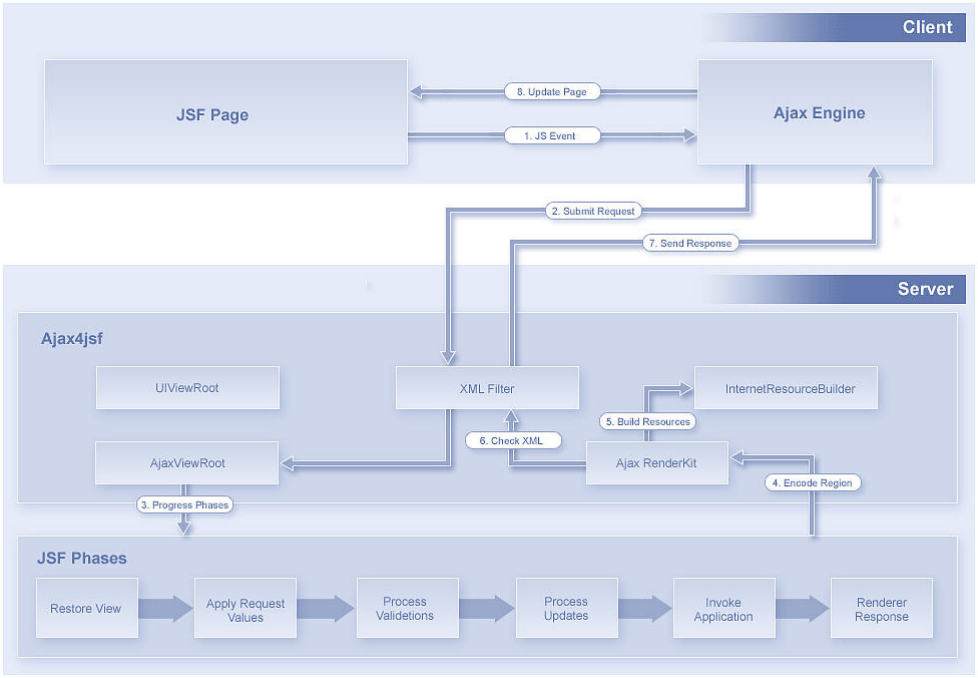

Abbildung 5.2: AJAX4JSF Request [19]

Wie aus der Abbildung 5.5.2 zu sehen ist, nutzt Ajax4jsf das JSF inklusive dessen LifeCycle (4.1). Dadurch bleiben insbesondere die Vorteile von JSF intakt. Ajax4Jsf wird als a4j abgekürzt.
a4j bietet ein interessantes Konzept. Während andere JSF- Implementierungen auf visuelle Komponenten setzen, basiert A4J auf funktionale (nicht- visuelle). Jeder dieser Komponenten steht im direkten Zusammenhang mit der Server- Interaktion. Diese Komponenten sind unsichtbar und können mit vielen JSF Komponenten gemischt werden. Dadurch ist es möglich, eigene Komponenten zu entwickeln, die AJAX- Funktionalität bieten. Bestes Beispiel dafür ist RichFaces.
Die Nutzung dieser Bibliothek enthält weitere Vorteile, die für diese Webapplikation die Entwicklung beschleunigen und verbessern.

- Möglichkeit vor und nach dem Request Aktionen auszuführen
- Beans mittels JSON (Javascript Object Notation) zum Client zu schicken
- Optimierung des Requests, um die Netzwerklast niedrig zu halten

Die zentrale Komponente, die für diese Applikation gebraucht wird 2.1, ist <a4j:poll>. Mit dieser Komponente kann man im Hintergrund periodisch eine Anfrage zum Server und eine Aktion ausführen. Komfortabel aktualisiert die Webapplikation mittels der

Möglichkeit, nach dem Request eine Javascript- Aktion durchzuführen, die Anzeige. Die Rückgabe der Serverdaten erfolgt im JSON Format. Dies ist besonders für den Polygonzug und das Balkendiagramm von Bedeutung, da hier die visuellen Komponenten nicht in JSF sind. Die Tabellenansicht profitiert von RichFaces, welche eine integrierte AJAX Funktionalität enthält, daher wird sie keine solche Funktionalität brauchen.
Um die Anforderung einer Webappliaktion mit minimaler Verzögerung nachzukommen, muß zwischen Client und Server eine effiziente Übertragung mit wenig Netzwerlast erfolgen. a4j besitzt solch eine Vorrausetzung, weil es auch Beans zum Client per JSON schicken kann.

5.5.3 Das JSON Format

JSON ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen ist. Für Maschinen ist es einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren als bei XML. Da AJAX als Antwort zum Client Daten in XML ablegt, werden zum Teil auch redundante Daten (mehrfache Tags) mitgeschickt, welche den Overhead erhöhen. Bei JSON handelt es sich um ein Textformat, das komplett unabhängig von Programmiersprachen ist [20]. JSON baut auf zwei Strukturen auf:

- *Name/Wert Paare*. In verschiedenen Sprachen wird dies realisiert als ein Objekt (object), Satz (record), Struktur (struct), Wörterbuch bzw. Verzeichnis (dictionary), Hash-Tabelle (hash table), Schlüssel-Liste (keyed list) oder als ein assoziatives Array (associative array).
- *Eine geordnete Liste von Werten*. In den meisten Sprachen wird das als Array (array), Vektor (vector), Liste (list) oder Sequenz (sequence) realisiert.

5.6 SVG

Zur Zeit existieren zwei Wege, um Grafiken im Browser anzuzeigen; eine proprietäre Javascript Lösung mit einem HTML Canvas Element , welches vorgestellt wird:

```
<html>
 <head>
 <script type="application/x-javascript">
  function draw() {
   var canvas = document.getElementById("canvas");
   var ctx = canvas.getContext("2d");
   ctx.fillStyle = "rgb(200,0,0)";
   ctx.fillRect (10, 10, 55, 50);
   ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
   ctx.fillRect (30, 30, 55, 50); }
 </script>
 </head>
 <body onload="draw()">
  <canvas id="canvas" width="300" height="300"></canvas>
 </body>
</html>
```

Abbildung 5.3: HTML- Canvas Auszug

Die Ausgabe sieht folgendermaßen aus:

Abbildung 5.4: Canvas- Grafik

Diese Lösung hat zwei Nachteile:

- Diese Lösung setzt hauptsächlich Javascript ein, um Grafiken zu erzeugen, die Nachteile von Javascript wurden bereits erörtert.
- Eine nachträgliche Änderung des Canvas Objektes gestaltet sich mit diesem Konzept schwierig.

Eine elegantere Lösung bietet SVG (Scalable Vector Graphics). Dabei handelt es sich um einen Vektorgrafikstandard, der im Jahr 2001 vom World Wide Web Consortium (W3C)

verabschiedet wurde und sich momentan in der Version 1.1 befindet [21].
Mit SVG ist es möglich, Vektorgrafiken für das World Wide Web (WWW), und für eine Vielzahl anderer Anwendungen zu beschreiben, weil SVG eine XML (Extensible Markup Language) Sprache darstellt. Die Regeln, die für XML restriktiv sind, gelten auch bei SVG. Es lassen sich textbasiert Grafiken erzeugen, welche einfache Grundformen, aber auch Füllungen mit Farbverläufen oder sogar aufwendige Filtereffekte beinhalten können [22]. Die geringe Größe sowie die als leicht zu erstellende und lesbare Textdatei sprechen für SVG. Außerdem besitzt ein SVG-Dokument wie ein XML- Dokument ein Document Object Model (DOM) und lässt sich darüber mit Skriptsprachen wie JavaScript ansprechen. Um SVGs darzustellen wird ein Web-Browser benötigt, der diese anzeigen kann. Für den Internet Explorer gibt es von Adobe den SVG Viewer als Plugin, mozillabasierte Browser unterstützen SVGs nativ.

5.6.1 Fazit

SVG bietet interessante Möglichkeiten zur Darstellung. Mit SVG kann zwar nicht vermieden werden, komplett auf Javascript zu verzichten, aber auf jeden Fall angenehmer zu entwickeln als das Canvas- Element. Mit Javascript muß nur der aktuelle DOM Baum aktualisiert werden, dies ist überschaubar. Mithilfe von <a4j:poll> kann eine Funktion nach dem Request aufgerufen werden, die dies erledigt. Ein weiterer Vorteil ist ebenfalls die Nutzung von J2SE- Technologien. Da SVG von XML abgeleitet ist, läßt sich JAXP (Java API for XML Processing) nutzen. Dadurch lassen sich auf der Server- Seite das bestehende SVG Dokument parsen und ändern. Dies ist dann von Nutzen, wenn der Anwender neue Diagramme hinzufügt, aber auch beim abspeichern läßt sich das Dokument durch JAXP serialisieren. Durch den Einsatz von SVG wird der Webserver bei der Berechnung und Anzeige von grafischen Elementen entlastet, denn die Darstellung der Anzeige wird auf den Client verlagert. Dies ist entscheidend, dadurch kann eine effiziente Kommunikation mit JSON realisiert werden, in dem der Webserver nur noch Emissionswerte zum Client schicken muß.

5.7 Fehlerbehandlung

Bei der Vorstellung von JSF wurde explizit die einheitliche Fehlerbehandlung als positives Merkmal formuliert. Das hängt damit zusammen, daß man mittels spezieller <h:message> Tags, Fehler in JSF für den Anwender auf einer Seite anzeigen kann. Java integriert ein Fehlerbehandlungkonzept; die Hierarchie der Exception Klassen. Wenn eine Routine eine

Ausnahme wirft, wird sie je nach Designkonzept zu einer oberen Instanz weitergereicht oder sofort geworfen. Dies kann auch für diese Applikation genutzt werden, in dem in der Ausnahmebehandlung eine FacesMessage erzeugt wird. Diese Message wird im JSF-Lifecycle in eine Message- Queue eingereiht. Die Message wird in der Phase "Response erstellen" ausgegeben.
Eine andere Möglichkeit wird in [10] beschrieben. Man bedient sich der Servlet- Engine, um Fehler zu behandeln und sie an eine JSF Fehlerseite zu delegieren. Zwar ist diese Lösung nicht so elegant wie die erste, aber es hat Vorteile:

- Es nutzt den JSP- Standardmechanismus (redirect).
- Es braucht keine eigene Servlet- Entwicklung.
- Es kann einfach angepaßt werden, in dem die Definition des Error- Handlers in der web.xml geändert wird.

```
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/pages/error.jspx</location>
</error-page>
```

Abbildung 5.5: Exception Handling

Aus Sicherheitsgründen wird nicht der genaue Fehlerverlauf in Form von einem Stacktrace angezeigt, wie es üblich ist, um Informationen potentiellen Angreifern nicht zugänglich zu machen.

5.8 Sicherheitskonzept

Die Userinformation ist in einer Datenbank abgelegt. Es ist aber auch möglich mittels JAAS (Java Authentication and Authorization Service) eigene User einzurichten. Dieses Feature bietet die Servlet API, sie enthält auch ein eigenes Formular zur Anmeldung, die auf diese User zurückgreift. Damit ist es nur registrierten Anwendern anhand von Regeln möglich überhaupt auf die Webapplikation zuzugreifen.

Die SessionID bleibt während der Sizung aktiv, ein Sicherheitsrisiko entsteht, wenn sich

nicht ordungsgemäß abgemeldet wird und nur der Browser geschlossen wird. Ein anderer Nutzer könnte lokal über diese Session auf Daten zugreifen , allerdings kann man dieses Risiko minimieren; die Sessiondauer bei Inaktivität kann man im Tomcat in Minuten einstellen.

Tomcat, wie alle Servlet- Container, unterstützt mehrere Methoden der Authentifizierung.

- HTTP based authentication
- Form based authentication
- HTTP digest authentication
- HTTPS client authentication

Dadurch ist eine einfache Möglichkeit gegeben, die Funktionalität “verschlüsselte Anmeldung” in die bestehende Webapplikation zu integrieren. Allerdings ist nur HTTPS client authentication die sicherste Methode, weil sie HTTP Daten über SSL verschlüsselt.
HTTPS ist ein Standard für die verschlüsselte Übertragung von Daten zwischen Browser und Webserver. Er beruht auf X.509-Zertifikaten. Grundlage ist das unsymmetrische Verschlüsselungsverfahren. Dafür sind die übertragenen Schlüssel durch einen Angreifer nicht abfangbar. Damit diese genutzt werden können, muß zunächst eine Zertifizierungsinstanz (Certificate Authority - CA) eingerichtet werden. Diese garantiert die unverfälschte Übertragung der öffentlichen Schlüssel und die Echtheit des Webservers. Das Zertifikat der CA wird in allen Browsern installiert und erscheint dort als "Vertrauenswürdige Stammzertifizierungsstelle". Der Webserver generiert ebenfalls ein Schlüsselpaar, dessen öffentlicher Teil zur CA übertragen wird. Die CA prüft die Angaben des Webserverbetreibers und signiert den Schlüssel. Das damit entstandene Webserver-Zertifikat garantiert die Echtheit des Webservers und stellt eine Garantie für Clients dar, sich mit dem richtigen Webserver verbunden zu haben.
Aufbau der HTTPS-Verbindung:
Der Browser baut daraufhin eine Verbindung über Port 443 zum Webserver auf. Der Webserver präsentiert sein Zertifikat, das der Client mit Hilfe des installierten CA-Zertifikats auf Echtheit überprüft. Danach erfolgt die nur für den Webserver lesbare Übertragung des Sitzungsschlüssels. Mit dem nun auf beiden Seiten vorhandenen Sitzungsschlüssel kann eine symmetrische Datenverschlüsselung beginnen.

5.9 Einbindung in das Gesamtkonzept

Nach dem “best of bread” Ansatz wurden die einzelnen Komponenten ausgesucht, die jeweils ihre eigene Teilfunktionalität erfüllen. Die Rechtecke entsprechen den vorgestellten Komponenten, während (Business Logic, Webserver, Client) die logische Trennung der einzelnen Schichten darstellt.

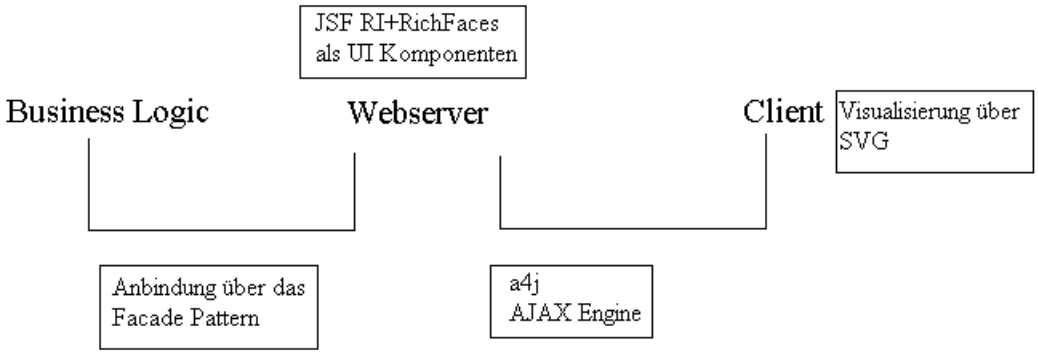

Abbildung 5.6: Gesamt-Komponentenansicht

Um das Zusammenspiel der einzelnen Komponenten zu verstehen, wird anhand eines Szenarios (Aktualisierung einer Visualisierung) verdeutlicht. JSF bildet die Plattform für die Webapplikation, die genannten Vorteile und Features sind genannt worden 4.1.
Mit der Auswahl der JSF+RichFaces Komponenten werden zusätzlich UI Komponenten zur Verfügung gestellt, die die visuelle Darstellung der Anwendung übernehmen 2.1, außer den besonderen Verlaufskurven und Balkendiagramme. Wenn der Anwender seine Schadsstoffkomponenten zur Ansicht hinzugefügt hat, läuft folgendes ab: Mit a4j wird periodisch ein AJAX Request zum Server geschickt, der diese Events, wie in JSF beschrieben, verarbeitet. Eine Bean ruft die Geschäftslogik über das Facade Pattern auf. Das Ergebnis dieser Routine wird wieder in eine Bean gespeichert. Auch hier ist a4j beteiligt, der AJAX Aufruf transportiert über das JSON Format Beans zum Client zurück. Wenn die Anfrage/ Antwort Zyklen beendet sind, wird eine Javascript Funktion aufgerufen, die die Aktualisierung des SVG- DOM Baums mit den JSON Daten vornimmt.

6 Realisierung eines Prototyps

Nach der Vorstellung der Konzepte und dessen Arbeitsweisen soll anhand der Anforderungen die praktische Umsetzung erfolgen. Eine kurze Skizze gibt Auskunft über die Navigation zu den Seiten. Die Vorteile der zentralen Navigationskonfiguration in JSF wird deutlich.

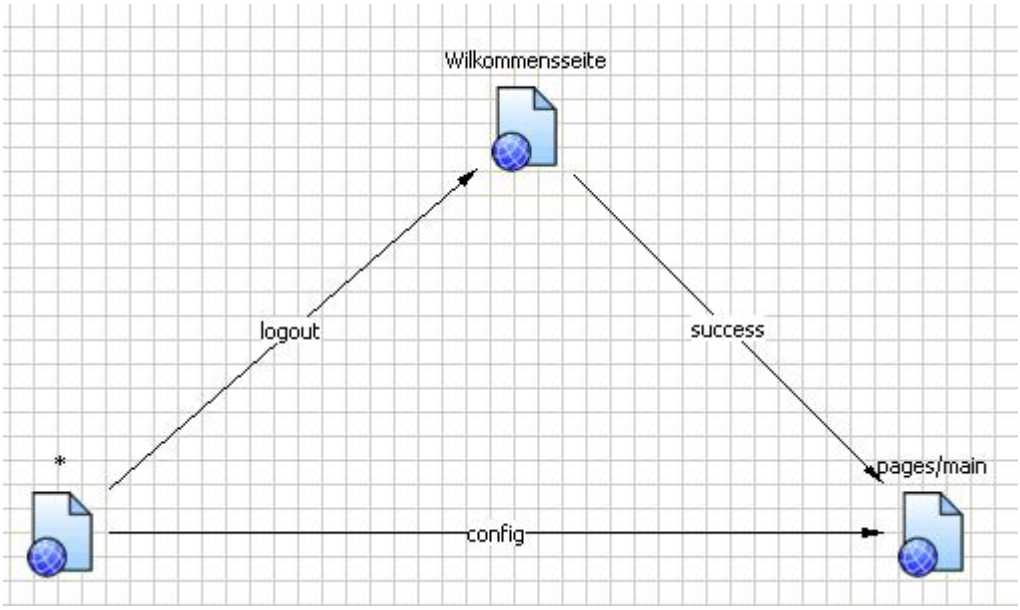

Abbildung 6.1: Navigationskonfiguration

Wenn die richtige URL im Browser eingegeben wird, erscheint die Willkommensseite (login.jspx) mit der Aufforderung zur Eingabe von Benutzernamen und Passwort. Die Formulardaten werden über ein Button verschickt, der ein Event auslöst, um den richtigen User zu verifizieren. Falls der User nicht existiert, wird eine Fehlermeldung auf der selben Seite ausgegeben. Hier zeigt sich die Einfachheit von JSF, mit <message> Tag lassen sich diese Fehler ausgeben. Wenn das event mit einem “success” beendet wird, leitet JSF an die Seite main.jspx weiter.

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
version="2.0"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:a4j="https://ajax4jsf.dev.java.net/ajax"
xmlns:rich="http://richfaces.ajax4jsf.org/rich">
```

Abbildung 6.2: Einbinden der JSP custom Tags von JSF

Jede XHTML Seite muß diese JSP-Tags vorher inkludieren, um die UI- Komponenten nutzen zu können. Anhand der Struktur wird deutlich, daß die Seiten mit Namespaces arbeiten, d.h. die Seiten sind im XHTML Standard. Dies ist für SVG und die Manipulation über DOM zwingend notwendig, daher sind alle Seiten XHTML konform.

```
<body onload="Richfaces.showModalPanel('loginPanel')">
 <a4j:loadStyle src="/css/login" />
 <h:graphicImage value="/images/meac2000.jpg"/>
 <rich:modalPanel id="loginPanel" resizeable="false"
      width="270" height="130">
  <f:facet name="header">
   <h:outputText value="Anmeldung" />
  </f:facet>
  <h:form >
   <h:panelGrid columns="2" border="0">
    <h:outputLabel for="user" value="Login-Name:" />
    <h:inputText id="user" value="#{login.username}"
         maxlength="15" required="true"/>
    <h:outputLabel for="pass" value="Passwort:" />
    <h:inputSecret id="pass" maxlength="10"
       value="#{login.password}" required="true"/>
    <h:commandButton value="Anmelden" action="#{login.doLogin}"
        actionListener="#{plant.createPlantConfig}" />
    <h:messages />
   </h:panelGrid>
  </h:form>
 </rich:modalPanel>
</body>
```

Abbildung 6.3: Ausschnitt aus login.jspx

Durch die Anbindung der JSF- RichFaces Bibliotheken lassen sich modale Formulare erzeugen, zuständig dafür ist <rich:modalPanel> Tag, welches weitere JSF UI Komponenten umschließt. Deutlich zu sehen sind, die Beans, Validatoren, Actions, welche für die Navigation verantwortlich ist, und ActionListener, die Events verarbeiten. Validatoren kommen bei der Eingabe zum Einsatz , mit “required=true” darf das Eingabefeld nicht leer sein. Es können nur maximal 10 Zeichen eingegeben werden, der Standard-Validator “maxlength” sorgt hierfür. Die eingegebenen Zeichen werden in einer Bean gespeichert, die mit “login” referenziert wird. Alle Beans müssen in der *facesConfig.xml* deklariert sein. Beim Anklicken des Buttons, wird ein Action und ein ActionListener ausgeführt. Diese Action verarbeitet die Benutzerdaten für den NavigationHandler, während der ActionListener die Konfiguration der der Anlage und Schadstoffkomponenten aus der Datenbank ausliest. Wie schon im Kapitel Design- Patterns erwähnt, werden die Geschäftslogiken als Facade- Pattern realisiert.

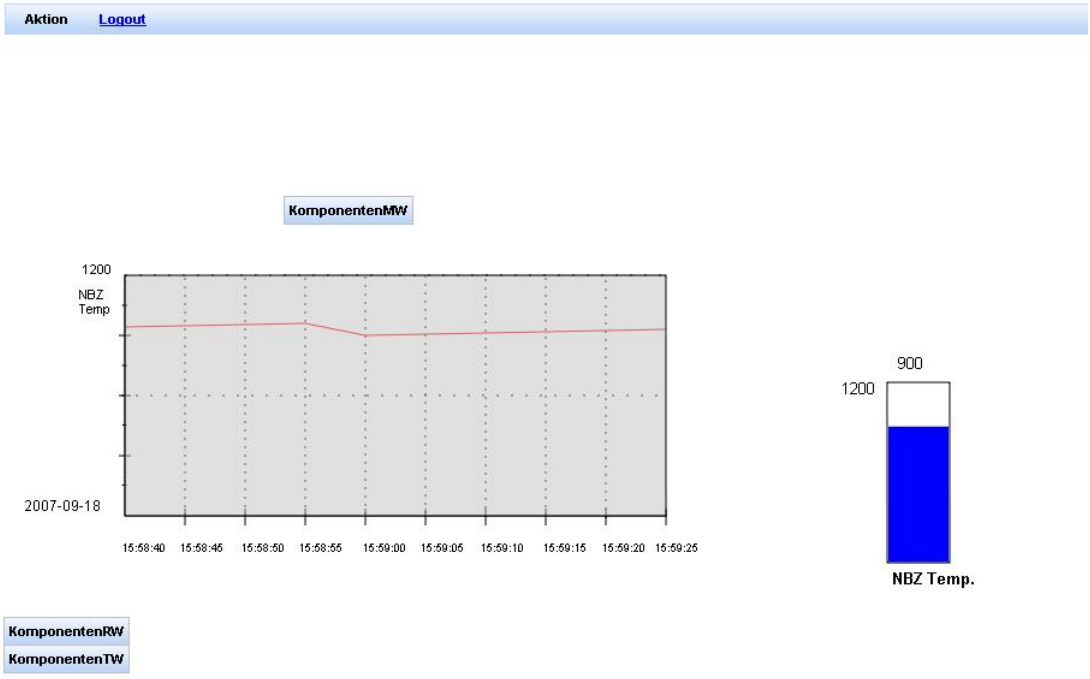

Abbildung 6.4: Die Hauptansicht

Nach erfolgreichem Anmelden gelangt man in die Hauptansicht, wo neue Schadstoffkomponenten zur Anzeige hinzugefügt werden können. Dies geschieht in der Menüzeile

Aktion-> Konfiguration. Für diese UI- Komponenten werden wiederum die JSF- RichFaces Bibliotheken genutzt.

```
<h:form>
 <rich:toolBar itemSeparator="none">
  <rich:dropDownMenu value="Aktion">
   <rich:menuItem submitMode="none">
    <h:outputText value="Echtzeit" />
   </rich:menuItem>
   <rich:menuItem submitMode="none">
    <h:outputLink value="javascript:Richfaces.showModalPanel('testPanel
        ')">
     <h:outputText value="Konfig" />
    </h:outputLink>
   </rich:menuItem>
  </rich:dropDownMenu>
  <h:commandLink action="#{login.doLogout}">
   <h:outputText value="Logout" />
  </h:commandLink>
 </rich:toolBar>
</h:form>
```

Abbildung 6.5: RichFaces UI- Komponenten

Durch ein Klick auf “Aktion” erscheint das Konfigurationsfenster. Beim Anklicken auf “ausführen” werden diese Komponenten zur Hauptansicht hinzugefügt.

Abbildung 6.6: Konfigurationsdialog

Mit einem Klick auf “logout” wird die Action logout in der Bean ausgeführt, die einen String “logout” zurückgibt und die Navigation wieder auf die Login Seite weiterleitet. Die logout Aktion löscht die SessionID.

Für die Aktualisierung der Komponenten ist die Hauptansicht verantwortlich. Der Aufbau wird ausführlich erläutert, da der zentrale Mechanismus die a4j- Komponente ist, die verschiedene Aufgaben übernimmt.
Hier muß allerdings zwischen dem AJAX- Ablauf in SVG und Tabellenansicht unterschieden werden. Um XML- Verarbeitung in SVG von der HTML Seite zu entkoppeln, werden die SVGs für die Balkenansicht und Polygonansicht in getrennten Dateien aufbewahrt. Über den <a4j:include> Tag wird wie beim <jsp:include> diese Ressourcen in die Hauptansicht (main.jspx) eingebunden. Dies hat den Vorteil, daß man mit der XML- API nur diese Dateien für z.B. Balkenansicht oder Polygonzug manipulieren muß und nicht erst die komplette XHTML Seite. Als erstes wird die Tabellenansicht betrachtet.

```
<a4j:region>
 <h:form id="mw" onmousedown="dragstart(this)" style="#{dnd.cssStyle}">
  <rich:dataTable border="1" var="list" value="#{components.componentMW
      }" id="tableMW" columns="4">
   <f:facet name="header">
   <h:outputText value="KomponentenMW" />
   </f:facet>
   <h:column>
    <h:outputText value="Komponente" />
   </h:column>
   <h:column>
    <h:outputText value="#{list.bez}" />
   </h:column>
   <h:column>
    <h:outputText value="#{list.actualValue}" />
   </h:column>
   <h:column>
    <h:outputText value="#{list.dimension}" />
   </h:column>
  </rich:dataTable>
 <a4j:poll id="button1" actionListener="#{components.dataDB}"
  interval="5000" reRender="tableMW" rendered="#{startUpdate.update
      [0]}" />
 </h:form>
</a4j:region>
```

Abbildung 6.7: Code- Tabellenansicht

Als Beispiel wird hier nur die “KomponentenMW 6.4” genommen, für die anderen Messvarianten gilt dies aber analog. Die visuelle Komponente heißt <rich:dataTable>. Die Referenz auf die Daten, hält die Bean “value = components.componentMW” bereit. Gemäß Spezifikation, muß diese Referenz auf ein Array abgebildet werden, welches mit “var=list” auf einzelene Objekte im Array zugegriffen wird. Dieses Array beinhaltet alle zu anzuzeigenden Schadstoffkomponenten.
<a4j:poll> Tag ist eine nichtvisuelle JSF- Komponente. Sie sorgt für die Aktualisierung des Wertes, in diesem Fall “list.actualValue”. Als Attribut hat sie den Intervall zwischen den Abfragen, hier beträgt sie 5 sec. ReRender Attribut gibt per id an, welche JSF- Komponente nach der Antwort per AJAX aktualisiert wird, die id referenziert genau diese dataTable. Der ActionListener wird nur dann aktiv, wenn “rendered=true” ist. Das ist dann der Fall, wenn die erste Schadstoffkomponente im Konfigurationsdialog zur dataTable hinzugefügt wurde. Das Tag <a4j:region> kann Seiten in Regionen aufteilen. Die Updates werden nur in diesen Regionen vollzogen. Dadurch wird weniger Netzwerkverkehr erzeugt und weniger Code aufgerufen, was der Performance zugute kommt.
Besonders zu erwähnen ist, daß letztendlich Javascript versteckt durch a4j zum Einsatz kommt. Der Entwickler muß sich um AJAX/ Javascript Aufrufe nicht kümmern, nur die ActionListener müßen implementiert werden, die Werte aus der Datenbank holen.
Die DragnDrop- Funktionalität wird über Javascript abgewickelt, weil dies nur auf der Client- Seite passieren kann. Dafür sind in diesem Fall die Javascript-Eventhandler nötig, die Mausereignisse registrieren und darauf eine Callback- Funktion (onmousedown="dragstart(this)") aufrufen.

Etwas anders gestaltet sich der Sachverhalt bei SVG- Inhalten. Eine andere Vorgehensweise ist gefragt, weil SVG eine nicht JSF- Komponente darstellt, welche man nicht einfach wie im letzten Beispiel mit dem reRender Attribut neuzeichnen kann.

```
<h:form>
 <a4j:poll oncomplete="updateValues(data)" data="#{createComponent.
     value}" interval="5000"                actionListener="#{
     createComponent.createComponents}" />
</h:form>
<h:form>
<a4j:poll oncomplete="updatexy(data)"data="#{valueXY}"interval="5000"
                                          actionListener="#{createComponent
    .createComponentsxy}" />
</h:form>
```

Abbildung 6.8: SVG- AJAX Aktualisierung

```
function updateValues(data){
 var balkenText= document.getElementById('textValue');
 balkenText.firstChild.data=data;
 var value = document.getElementById('value');
 value.setAttribute('height', (data/1200)*150);
}
function updatexy(data){
 var i=0,j=9;
 var newPolyline='';
 var d= document.getElementById('day');
 d.childNodes[0].firstChild.data=data.date;
 for(i ; i<10 ; i++){
  var test =i+1;
  var text= document.getElementById('x'+test);
  text.childNodes[0].firstChild.data=data.timestamp[j];
  j--;
 }
 i=0;
 j=9;
 var value=0;
 var y=100;
 for(i; i<10;i++){
  value= calculateValue(data.value[j]);
  newPolyline+=y+','+value+' ';
  y+=50;
  j--;
 }
 var polyline= document.getElementById('valuexy');
```

```
 polyline.setAttribute('points',newPolyline);
}
function calculateValue(value){
 return Math.floor((-1/6)*value+400);
}
```

Abbildung 6.9: SVG- DOM Aktualisierung

Zwei neue Attribute sind hier notwendig. Das “data” Attribut kann Beans im JSON Format nach der AJAX Antwort zur Verfügung stellen. Dies hat Vorteile. JSON ist ein schnelles und effizientes Übertragungsformat, weil dies schneller geparst werden kann als XML. Overhead wird vermieden, weil JSON durch Javascript die Objektstruktur schneller auflösen kann als bei XML. Durch JSON laßen sich beliebige Daten zum Client schicken. Das Attribut “oncomplete=updateValues(data)” nutzt diese JSON Daten, die durch das “data” Attribut vorgegeben ist, um eine Funktion, nachdem die Antwort vom Webserver erfolgt ist, aufzurufen und mittels SVG- DOM, die Werte im SVG dynamisch zu ändern. Der ActionListener erfüllt die gleiche Funktion, wie bei der Tabellenansicht mit dem Unterschied, daß die Bean “createComponent.value” mit Werten gefüllt wird, die dann als JSON Format dem oncomplete zur Verfügung stehen. Der Aufwand in SVG ist höher, da DOM Funktionalitäten für SVG implementiert werden müssen (Abbildung 6.9).

6.1 Funktionsüberprüfung

Um die Funktionen, die implemntiert wurden, auf Korrektheit zu überprüfen, greift man normalerweise auf Unit- Tests zurück. Es existieren für Java und Servlets auch Frameworks, die Unterstützung hierfür bieten (Jakarta Cactus/ JUnit). JSF ist aber nicht zu Cactus/ JUnit kompatibel. Die Ursache liegt am JSF- Lifecycle 4.1. Jede Anfrage durchläuft sie und dabei wird seitens JSF ein FacesContext erzeugt. Dies ist aber so in Unit Tests nicht möglich, da dieses Objekt Informationen über den Komponentenbaum enthält, die in Cactus/ JUnit nicht zur Verfügung stehen. Das Shale Test Framework [23] soll mit JSF Tests umgehen können, allerdings konnte dies aus Zeitgründen nicht mehr getestet werden und ist aber auch nicht Gegenstand dieses Themas. Aus diesem Grund wurde ein einfaches Verfahren für die Verfizierung genutzt. Um die Daten für die Visualisierung, die vom Client angefordert werden zu testen, wurde ein Logger eingerichtet, der jeden Datenbankzugriff auf dem Server in eine Datei protokolliert. Um sicherzugehen, daß auch die richtigen Daten angezeigt werden, wird auf dem Client per Javascript

ein “alert” Befehl zum testen benutzt, der immer die übermittelten Daten als Popup am Client anzeigt. Damit ist ein Vergleich zwischen den auf dem Server generierten Daten mit dem auf dem Client möglich.

6.2 Performanceoptimierung

Um Performance beurteilen zu können, muß man zwischen Codeoptimierung bzw. das Konzept und Optimierung, welche die äußeren Umstände betrifft, wie z.B Java- VM, unterscheiden. Zunächst wird die Datenbank, die die Emissionsdaten bereithält, untersucht. Hierbei handelt es sich um die freie mySQL Datenbank in der Version 4.1.

6.2.1 Datenbank Zugriffsoptimierung

Java Database Connectivity (JDBC) bietet in Java transparenten Zugriff auf Datenbanken 3.1. Durch Connection Pooling kann eine Optimierung vorgenommen werden. Wenn nicht Extrazeiten für das Erlangen einer Datenbankverbindung in die Antwortszeit für eine Aktion einkalkuliert werden sollen, wird auf Connection Pooling zurückgegriffen. Wenn ein mit einer Client Aufgabe assoziierter Server Thread eine Datenbankverbindung benötigt, wird ihm diese durch die Datasource aus einem Pool bereits geöffneter Connection Objekte heraus gegeben. Wenn die Connection durch Aufruf von close() geschlossen wird, wird sie nicht wirklich geschlossen, sondern nur in den freien Pool gerade unbenutzter Verbindungen zurückgestellt. Im Anwendungscode ändert sich für den Entwickler nichts – der Connection Pooling Mechanismus läuft transparent im Tomcat ab. [24]
Verändert ist hierin nur der Begriff einer Datasource. Damit der Connection Pooling Mechanismus greift, dürfen Connection Objekte nicht mehr über den Driver Manager, sondern von einer Datasource bezogen werden. Um eine Anwendung für Connection Pooling zu befähigen, müssen die Connection Objekte von einer Datasource bezogen werden. Die Datasource selbst aber wird nicht über JDBC erhalten, sondern über JNDI – Java Naming and Directory Interfaces, ein Bestandteil von J2SE. Ein Auszug aus der context.xml, der diese DataSource in Tomcat zur Verfügung stellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
 <Resource auth="Container" name="jdbc/meac2000" type="javax.sql.
    DataSource" password="toor"                driverClassName="com.
    mysql.jdbc.Driver" maxIdle="10" maxWait="2000"
                                    validationQuery="" username="root"
    url="jdbc:mysql://10.153.16.197:3306/meac2000" maxActive="1000"/>
</Context>
```

Abbildung 6.10: DataSource- Konfiguration

Falls sich die Adresse der Datenbank ändert, muß hier nur die Änderung vorgenommen werden.

Ein weitere Optimierung im Source kann mittels *Prepared Statements* erfolgen. Die SQL-Anweisungen, die an die Datenbank gesendet werden, haben bis zur Ausführung im Datenbanksystem interne Umwandlungen vor sich. Zuerst müßen sie auf syntaktische Korrektheit geprüft werden. Dann werden sie in einen internen Ausführungsplan in der Datenbank übersetzt und eventuell mit anderen Transaktionen optimal sortiert. Der Aufwand für jede Anweisung ist meßbar. Besser wäre jedoch eine Art Vorübersetzung für SQL- Anweisungen zu nutzen.
Diese Vorübersetzung ist eine Eigenschaft, die JDBC unterstützt und sich Prepared Statements nennt. Vorbereitet (engl. prepared) deshalb, weil die Anweisungen in einem ersten Schritt zur Datenbank geschickt und dort in ein internes Format umgesetzt werden. Später verweist ein Programm auf diese vorübersetzten Anweisungen, und die Datenbank kann sie schnell ausführen, da sie gecachet vorliegen. Ein Geschwindigkeitsvorteil macht sich immer dann besonders bemerkbar, wenn Schleifen Änderungen an Tabellenspalten vornehmen. Das kann durch die vorbereiteten Anweisungen schneller geschehen.[25]
Von Vorteil sind diese prepared statements deshalb, weil sie sich für diese Webapplikation eignen, denn dieselbe Abfrage wird immer wieder zyklisch vom Client ausgelöst, daher können diese Statements gecachet werden.
Ein weiterer Aspekt der prepared statements, welches allerdings nichts mit der Performance gemeinsam hat, ist die Sicherheit. Im Unterkapitel “Sicherheitsaspekte” wird dies gesondert abgehandelt.

6.2.2 Java VM- Optimierung

Die Java- VM mit ihrer Einstellungsvielfalt läßt sich auf die Bedürfnisse der jeweiligen Anwendung optimieren. Gemäß [26] sollen Serverapplikationen mit der Option -server in der VM gestartet werden. Es veranlaßt zusätzliche Optimierungen bei der Ausführung wie Loop peeling and unrolling, inlining, dead code elimination. Zusätzlich wir der Heap auf einen größeren Wert als Standard (-Xms64m -Xms256m) eingestellt (-Xms512m -Xmx512m), wobei der erste Parameter das Minimum, der zweite das Maximum an Heapgröße darstellt. Beide Werte werden auf denselben Wert gestellt, damit nicht neuer Speicher erst alloziert werden muß, falls das Minimum nicht ausreicht.
Die Werte werden gleichgroß gewählt, um

- mehr Heapspeicher für die Allokation von Objekten zur Verfügung zu stellen=> Der Garbage Collector muß weniger aufgerufen werden
- eine Vergrößerung des minimalen Heapwertes während der Laufzeit zu vermeiden => Heap wird nicht fragmentiert

Damit ist ein Grundstein für bessere Performance gelegt, welches später angepaßt werden kann. Mit den eingestellten Optionen wurde die Webapplikation erfolgreich getestet werden. Für weitere komplexere Optimierungen wird [26] verwiesen.

6.3 Sicherheit

Die Applikation unterstützt eine sichere Authentifizierung am System. Tomcat unterstützt diverse Authentifizierungsmechanismen. JDBCRealm ist eine Möglichkeit, dies zu implementieren. Allerdings erfordert dies eine genaue Tabellenstruktur in der mySQL Datenbank, die nicht zur Verfügung stand.
Eine Standard Realm wird mit jedem Tomcat mitgeliefert und befindet sich in der tomcat-users.xml. Eine Authentifizierung über JAAS in Verbindung mit JDBC ist somit nicht möglich. Aus diesem Grunde wurde ein eigenes Formular entwickelt. In Zukunft wird aber ein JDBCRealm angestrebt, falls die Tabellenstruktur neu entworfen wird. Um Verschlüsselung zu gewährleisten, wird SSL eingesetzt. Die Einträge in der web.xml sieht folgendermaßen aus:.

```
<security-constraint>
<!-- Geschützer bereich. ALle Ressourcen unter dem Verzeichnis
    protected -->                                    <web-resource-collection>
    <web-resource-name>SSL Bereich</web-resource-name>
    <url-pattern>/pages/login.jspx</url-pattern>
  </web-resource-collection>
<!--SSL erzwingen -->
 <user-data-constraint>

     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
<auth-method>BASIC</auth-method>
</login-config>
```

Abbildung 6.11: SSL- Definitionen in web.xml

Dieses Listing wird zusätzlich in die web.xml eingetragen. Dies bewirkt, daß das Login-Formular nur über HTTPS erreichbar ist. Die restlichen Seiten werden üblich über HTTP abgewickelt. Dies ist sinnvoll, weil diese Seiten keine personbezogenen Daten übertragen, sondern nur Emissionswerte (Integers/ Doubles). Dieser Verkehr muß nicht verschlüsselt werden.
Zusätzlich muß in der server.xml die SSL- Fähigkeit aktiviert werden.

```
<Connector port="443" maxThreads="150" minSpareThreads="25"
    maxSpareThreads="75" enableLookups="false" disableUploadTimeout="
    true" acceptCount="100" debug="0" scheme="https" secure="true"
    keystorePass="mypwd" keystoreFile="C:\Dokumente und Einstellungen\
    test\.keystore" clientAuth="false" sslProtocol="TLS" />
```

Abbildung 6.12: SSL- Definitionen in server.xml

Im Kapitel Performanceaspekte wurden prepared statements diskutiert und behauptet, Sicherheit zu gewinnen. Prepared Statements verhelfen zu einem Schutz vor SQL- Injections; einschleusen von eigenem SQL-Code in bestehende Variablen. Ein Beispiel wäre das Login- Formular. Würde man direkt JDBC Anfragen stellen, könnte ein Angreifer

die Abfrage so manipulieren, daß er zusätzliche Strings mit in die Anfrage anhängt und so Informationen bzw. vertrauliche Daten von der Datenbank erhält, die nicht für ihn bestimmt sind. Durch Prepared Statements sind wie schon erklärt, eine Vorkompilierung der Anfrage und Typeninformation ausgeführt. Es ist nahezu unmöglich, die Abfrage zu manipulieren, weil die Struktur der Abfrage der Datenbank bekannt sind und jegliche Änderung zu einem SQL Fehler in der Datenbank führt. Die Abfrage wird erst gar nicht ausgeführt.

7 Ergebnis

7.1 Bewertung

Mit JSF hat man eine gute Basis geschaffen, um Aufgabenteile zu trennen. Die Applikation ist so ausgelegt, daß sie um benötigte Funktionalität einfach erweitert werden kann. Die View kann ohne Auswirkungen auf die Applikation geändert werden. Andersherum kann die Logik modifiziert werden, ohne das jede Seite überarbeitet werden muß.
Speziell durch die Auswahl des Facade Patterns ist die Anbindung der Logik an Beans leicht und übersichtlich zu realisieren.
Mit dieser Arbeit wurde gezeigt, daß mit einer modernen J2EE- Webarchitektur, Emissionsdaten im Browser wie bei einer Desktopanwendung visualisiert werden können. Durch verschiedene JSF- Bibliotheken konnte gezeigt werden, wie sie miteinander zu einer Gesamtlösung kombiniert werden können. Die JSF UI- Komponenten haben zu einer beschleunigten Entwicklung verholfen, da die notwendigen Kompontenen sofort einsatzbereit waren. Die a4j Bibliothek ist das Bindeglied zwischen Client und Server, welches sich transparent um die AJAX Funktionaliät kümmert. Somit bleibt das Testen und Debuggen von Javascript aus. Der schnelle Datenaustausch zwischen Client/ Server mittels JSON trägt zur Optimierung der Performance bei. Mit SVG konnte auf einen Vektorgrafikstandard zurückgegriffen werden, der vielfältige Visualisierung erlaubt. In Zukunft sind Erweiterungen auch hier problemlos möglich, da sich SVG gut in die Webarchitektur integriert.

7.2 Ausblick

Es wurde in dieser Arbeit die Anbindung der Geschäftslogik in JSF mehrfach kritisiert. Das Problem wurde mit einem bekannten Design Pattern ausgeräumt. JBOSS hat dies erkannt und hat einen Anwendungsframework vorgestellt, welches JSF+ AJAX+ EJB+ Portlets miteinander vereinigt.

Die präsentierte Lösung bildet noch Mischcode von JSF Tags und SVG. Sie ist zwar funktional, aber wünschenswert wäre eine einheitlicher Aufbau von HTML Seiten mit nur JSF Elementen. Mit JSF kann man eigene Renderer entwickeln, die spezielle JSF-SVG Tags in HTML rendern könnten. Mangels Zeit, die zur Verfügung stand, konnte das leider nicht bewerkstelligt werden. Es existieren Tutorials, die beschreiben, wie problemlos und unkompliziert zu verfahren ist, [27] dies setzt aber hohe, fachliche Kenntnisse von Java im allgemeinen und JSF voraus.
Mit solch einer Lösung würde die Entwicklung von SVG Inhalten in JSF viel schneller vorangetrieben werden. Warum nicht schon eine Implementierung auf JSF Basis vorliegt, kann nur geschätzt werden. Der Funktionsumfang von SVG ist zu groß ist, um einen allgemeinen Custom- Tag für JSF zu entwickeln. Allein durch die Vielfalt der SVG Attribute wäre eine unüberschaubare Kombination möglich, die die Attribute der JSF- Bibliothek sprengen würde.

Literaturverzeichnis

[1] *JBOSS community driven*, http://labs.jboss.com/.

[2] *J2EE Sun BluePrints*, http://java.sun.com/j2ee/blueprints/.

[3] *Overview Java 2 Platform Enterprise Edition*, http://java.sun.com/j2ee/overview.html.

[4] *Java Servlet Technology*, http://java.sun.com/products/servlet/index.jsp.

[5] *JavaServer Pages Technology*, http://java.sun.com/products/jsp/index.jsp.

[6] B. W. Perry, *Java Servlet & JSP Cookbook*, B. McLaughlin, Ed. Sebastopol: O'Reilly, Jan. 2004 First Edition.

[7] *Struts framework and model-view-controller design pattern*, http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp.

[8] *JSP Tutorial*, http://www.jsptutorial.org/.

[9] *Design Patterns for Building Flexible and Maintainable J2EE Applications*, http://java.sun.com/developer/technicalArticles/J2EE/despat/.

[10] B. W. L. M. Bill Dudney, Jonathan Lehr, *Mastering JavaServer Faces*. Greenwich: Wiley, 2004.

[11] *The XMLHttpRequest Object*, W3C Working Draft 18 June 2007.

[12] F. W.Zammeti, *AJAX Praxis*. Heidelberg: Apress L.P., 1.Auflage 2007.

[13] *Document Object Model (DOM)*.

[14] *Adobe Flex 2*, http://www.adobe.com/de/products/flex/.

[15] *JSR 252: JavaServer Faces 1.2*, http://www.jcp.org/en/jsr/detail?id=252.

[16] *The Life Cycle of a JavaServer Faces Page*, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSFIntro10.html.

[17] R. J. J. V. Erich Gamma, Richard Helm, *Design Patterns - Elements of Reusable Object- Oriented Software*. Amsterdam: Addison- Wesley, 2000, 20. Auflage.

[18] *RichFaces Developer Guide*, http://labs.jboss.com/jbossrichfaces/docs/index.html.

[19] *Ajax4jsf Developer Guide*, http://labs.jboss.com/jbossajax4jsf/docs/index.html.

[20] *Einführung in JSON*, http://json.org/json-de.html.

[21] *Scalable Vector Graphics (SVG) 1.1 Specification*.

[22] *SelfSVG*, http://www.selfsvg.info.

[23] *Shale Test Framework*, http://shale.apache.org/shale-test/index.html.

[24] *Connection Pooling in Webanwendungen*, http://www.libra.de/target/javaArticleConnectionPooling.

[25] C. Ullenboom, *Java ist auch eine Insel*. Bonn: Galileo Press, Auflage 2007.

[26] *J2SE and J2EE Performance Best Practices, Tips And Techniques*, http://java.sun.com/performance.

[27] *Creating and Using a Custom Render Kit*, http://java.sun.com/javaee/javaserverfaces/reference/docs/customRenderKit.html.

Versicherung über Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 27. September 2007 _____

Ort, Datum Unterschrift