

Diplomarbeit

Frank Saunus

Entwicklung dynamisch erstellter Map-Mashups
mittels XML-Konfigurationsdatei

Frank Saunus

Entwicklung dynamisch erstellter Map-Mashups
mittels XML-Konfigurationsdatei

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Cristoph Klauck
Zweitgutachter : Prof. Dr. rer. nat. Michael Neitzke

Abgegeben am 28. August 2008

Frank Saunus

Thema der Diplomarbeit

Entwicklung dynamisch erstellter Map-Mashups mittels XML-Konfigurationsdatei

Stichworte

Map-Mashup, XML, JAXB, Tomcat

Kurzzusammenfassung

Map-Mashups sind heute in vielen Bereichen des Internets anzutreffen. Sie bieten die Möglichkeit unzählige Dinge auf Karten anzuzeigen und so für den Nutzer Informationen interessant aufzubereiten. Diese Art der Darstellung hat natürlich auch ihre Grenzen. Die auf der Karte angegebenen Informationen haben zunächst einmal keinen Bezug zum Standort des Benutzers. Dafür muß er selber sorgen, indem er den für sich geeigneten Kartenausschnitt einstellt und so der Karte die gewünschten Informationen entnimmt.

Diese Arbeit beschäftigt sich mit der Entwicklung einer Webanwendung, die mittels einer einfach zu erstellenden XML-Konfigurationsdatei dynamische Map-Mashups erstellt. Diese Mashups können dann in bestehende Webauftritte integriert werden und bieten einem späteren Nutzer nach der Eingabe von Adressdaten zusätzliche, bereichsbezogene Informationen.

Frank Saunus

Title of the paper

Development of dynamically created map mashups through XML-configuration files

Keywords

Map mashup, XML, JAXB, Tomcat

Abstract

Map mashups can nowadays be found in many areas of the internet. They provide the possibility to show many things on maps and can process the information in an interesting way for the user. This kind of presentation has of course its limitations. Initially, the informations in the map have no connection to the users location. He has to set the appropriate map area by himself to extract the desired information.

This work deals with the development of a Web application which creates dynamically map mashups through easy to create XML-configuration files. These mashups can be embeded in existing Web presences and offer a future user additional area based information after suply of address data.

Inhaltsverzeichnis

Inhaltsverzeichnis	4
Abbildungsverzeichnis	8
Tabellenverzeichnis	9
1 Einleitung.....	10
1.1 Motivation	10
1.2 Verbreitete Lösungsansätze	11
1.2.1 <i>www.direct-getraenke.de</i>	11
1.2.2 <i>www.broetchenbursche.de</i>	12
1.2.3 <i>www.lehmannsbio.de</i>	12
1.2.4 <i>www.mr-pizza.com</i>	12
1.2.5 <i>pizza.de</i>	12
1.2.6 <i>Fazit...</i>	13
1.3 Zielsetzung	13
1.4 Gliederung der Arbeit	14
2 Anforderungen an die Webapplikation	16
2.1 Anwendungsfälle	16
2.2 Szenarien	17
2.3 Funktionale Anforderungen	21
2.4 Nicht-Funktionale Anforderungen	21
3 Webtechnologien	23
3.1 Servlets.....	23
3.2 JSP.....	25
3.3 Cascading Stylesheets	26
3.4 XML / Java Binding.....	27
3.5 JavaScript / DOM	27
3.6 AJAX.....	28
4 Design	29
4.1 Map-Mashup-Service	29
4.1.1 <i>Speicherort der XML-Konfigurationsdatei</i>	30
4.1.2 <i>Entscheidung Servlet / JSP</i>	30
4.1.3 <i>Persistente Daten</i>	31
4.1.4 <i>Sequenzdiagramme</i>	31
4.1.5 <i>Klassendiagramm</i>	35

4.1.6	<i>Listener</i>	36
4.1.7	<i>Filter</i>	37
4.1.8	<i>Fazit Design Map-Mashup-Service</i>	37
4.2	Tool zur Erstellung der Konfigurationsdatei	37
4.2.1	<i>Entscheidung Servlet / JSP</i>	37
4.2.2	<i>Custom Tags</i>	38
4.2.3	<i>Cascading Stylesheet</i>	38
4.2.4	<i>JavaScript</i>	38
4.2.5	<i>Sequenzdiagramme</i>	39
4.2.6	<i>Klassendiagramm</i>	41
4.2.7	<i>Usability des XML-Konfigurationstools</i>	42
4.2.8	<i>Fazit</i>	43
4.3	XML Schemata	44
4.3.1	<i>Russian Doll</i>	44
4.3.2	<i>Salami Slice</i>	44
4.3.3	<i>Venetian Blind</i>	44
4.3.4	<i>Garden of Eden</i>	44
4.3.5	<i>Fazit</i>	45
4.4	Sicherheit	45
4.4.1	<i>HTTPS</i>	45
4.4.2	<i>Parameter überprüfen</i>	45
4.4.3	<i>Löschen der Kundendaten</i>	46
4.4.4	<i>Fazit</i>	46
4.5	Fehlerbehandlung und Logging	46
4.5.1	<i>Exceptions</i>	46
4.5.2	<i>Error Page</i>	47
4.5.3	<i>Logging</i>	48
4.5.4	<i>Fazit</i>	48
5	Auswahl der Systemkomponenten	49
5.1	Webserver / Applikationsserver	49
5.2	Java Binding	49
5.3	Map API	50
5.4	Logging	50
5.5	Fazit	50
6	Erstellen der XML-Schemata	51

6.1	Schema user.xsd.....	51
6.2	Schema mapmashup.xsd.....	51
6.3	Testen der Schemata.....	52
6.4	Fazit.....	52
7	Einrichten der Entwicklungsumgebung.....	53
7.1	JDK.....	53
7.2	JAXB.....	53
7.3	Name-Service.....	54
7.4	Tomcat.....	54
7.4.1	SSL Zertifikat erstellen.....	55
7.4.2	Konfigurationsdateien.....	56
7.5	Eclipse... ..	61
7.6	Google-Maps API.....	61
7.7	Fazit.....	62
8	Implementierung.....	63
8.1	Map-Mashup-Service.....	63
8.1.1	Listener InitMms.....	63
8.1.2	Filter MmsFiler.....	64
8.1.3	Filter DownloadFilter.....	64
8.1.4	Servlet StartServlet.....	64
8.1.5	Servlet ResponseServlet.....	65
8.1.6	Klasse Document.....	65
8.1.7	JavaScriptdatei für Map-API.....	67
8.2	Tool zur Erstellung einer XML-Konfigurationsdatei.....	67
8.2.1	ConfigToolServlet.....	67
8.2.2	XMLDownloadServlet.....	68
8.2.3	Klasse XmlToolData.....	68
8.2.4	Custom Tags.....	69
8.2.5	include-Direktive.....	70
8.2.6	JavaScriptdatei mms.js.....	70
8.2.7	Cascading Stylesheet ConfigToolFormate.css.....	71
8.3	Exceptions.....	71
8.3.1	MmsException.....	71
8.3.2	XmlFileException.....	71
8.4	Logging.....	71

8.5	Deployment	72
8.6	Fazit.....	72
9	XML-Konfigurationstool Usability	73
9.1	Überprüfen der Usability des XML-Konfigurationstools	73
9.1.1	<i>Fazit...</i>	74
10	Tests.....	75
10.1	Einbindung in Webauftritt.....	75
10.1.1	<i>...mit dynamischer Seitengenerierung und Zugriff auf Eingabedaten.....</i>	<i>75</i>
10.1.2	<i>...mit statischen HTML-Seiten und Zugriff auf area-Parameter</i>	<i>76</i>
10.1.3	<i>...als Frame in einem Frameset.....</i>	<i>76</i>
10.1.4	<i>Fazit...</i>	<i>77</i>
11	Fazit und Ausblick	78
12	Literaturverzeichnis	81
13	Glossar.....	83
14	Angaben zum CD-Inhalt.....	84

Abbildungsverzeichnis

Abbildung 1: Grafische und textuelle Beschreibung des Liefergebietes.....	11
Abbildung 2: Eingabe des Standortes per Maus oder Eingabefeld	13
Abbildung 3: Anwendungsfall-Diagramm	17
Abbildung 4: Sequenzdiagramm Map-Mashup-Seite anfordern.....	32
Abbildung 5: Sequenzdiagramm Standortbezogenen Text abfragen / Daten speichern.....	33
Abbildung 6: Sequenzdiagramm Kundendaten anfordern.....	34
Abbildung 7: Klassendiagramm Map-Mashup-Service.....	35
Abbildung 8: Sequenzdiagramm Erstellung einer XML-Konfigurationsdatei	39
Abbildung 9: Sequenzdiagramm XML-Konfigurationsdatei herunterladen	40
Abbildung 10: Klassendiagramm XML-Konfigurationsdatei-Tool.....	41
Abbildung 11: Benutzung des XJC-Schema Compilers	54
Abbildung 12: Erstellung eines Keystores mit dem keytool-Programm	56
Abbildung 13: Map-Mashup-Seite in Frameset	77

Tabellenverzeichnis

Tabelle 1: Standortbezogenen Text abfragen	17
Tabelle 2: Map-Mashup-Seite anfordern	18
Tabelle 3: Daten speichern.....	18
Tabelle 4: Kundendaten anfordern	19
Tabelle 5: XML-Konfigurationsdatei erstellen	19
Tabelle 6: XML-Konfigurationsdatei herunterladen	20
Tabelle 7: Systemparameter einstellen	20
Tabelle 8: Benutzerdaten pflegen.....	20
Tabelle 9: Vergleich Java Binding Frameworks.....	49
Tabelle 10: Usabilitybewertung	73

1 Einleitung

1.1 Motivation

Das World Wide Web spielt in der heutigen Zeit eine wichtige Rolle in der Informationsbeschaffung. Ursprünglich wurde es Anfang der 1990 Jahre von Tim Berners Lee entwickelt, um den Informationsaustausch zwischen verschiedenen Abteilungen des CERN zu verbessern. Das revolutionäre am WWW war die direkte Verknüpfung der Informationen untereinander. Dieser einfache Zugriff auf Informationen machte das World Wide Web auch für die Allgemeinheit interessant. Im Jahre 2007 hatten bereits 64,9 % der deutschen Haushalte einen Internetzugang. Diese große Anzahl an Internetnutzern ist natürlich auch eine Chance für kleinere Unternehmen, um ihre Dienstleistungen im Internet anzubieten. Häufig ist das räumliche Gebiet in denen potenzielle Kunden beliefert werden können aber begrenzt oder es gibt andere Faktoren wie Liefertage oder Mindestbestellmenge, die Einfluss auf den Service des Dienstleistungsanbieters haben. In diesen Fällen müssen die potentiellen Kunden aufgeklärt werden, in welchen Gebieten der Service verfügbar ist und welche Nebenbedingungen dabei gelten. Abhängig davon, wie kompliziert einzelne Gebiete aufgebaut sind und wie viele es gibt, ist es möglich diese Informationen als einfachen Text bereitzustellen. Ein Servicegebiet, dessen Ausdehnung nicht durch einen einfachen Begriff wie den eines Stadtteils oder eines Ortes genau beschrieben werden kann, erfordert aber in vielen Fällen die Abbildung einer Karte auf der das Servicegebiet eingetragen ist. Gibt es noch zusätzliche Servicefaktoren, wird der beschreibende Text zu dem Liefergebiet am Ende doch wieder kompliziert – oder man nutzt mehrere Karten und kann so für jede Karte noch eine andere Randbedingung mit angeben. Bei komplizierter aufgeteilten Servicegebieten bleibt am Ende aber immer das Problem, dass sich ein potentieller Kunde entweder durch lange Texte arbeiten muss oder aus mehreren Karten die richtige für seinen Standort herausfinden muss.

1.2 Verbreitete Lösungsansätze

An dieser Stelle soll ein Überblick gegeben werden, wie standortspezifische Informationen normalerweise vermittelt werden. Dazu wurden von mir 5 Webseiten ausgesucht, die einen jeweils anderen Lösungsweg beschreiten.

1.2.1 www.direct-getraenke.de

Wie auf der nachfolgenden Abbildung zu sehen ist, wird bei diesem Webauftritt zu jedem Liefergebiet eine Übersichtskarte angezeigt. In der Übersichtskarte ist das jeweilige Liefergebiet farblich markiert und daneben befindet sich ein Textabsatz in dem die Liefertage und das Liefergebiet beschrieben werden. Es gibt auf der Webseite insgesamt 5 Liefergebiete und ein Bereich, in dem nicht oder nur auf Anfrage geliefert wird. Obwohl man sich anhand der Übersichtskarten recht schnell einen Überblick verschaffen kann in welchem Bereich geliefert wird, ist es notwendig sich mit der Beschreibung Klarheit über die einbezogenen Ortsteile zu verschaffen.

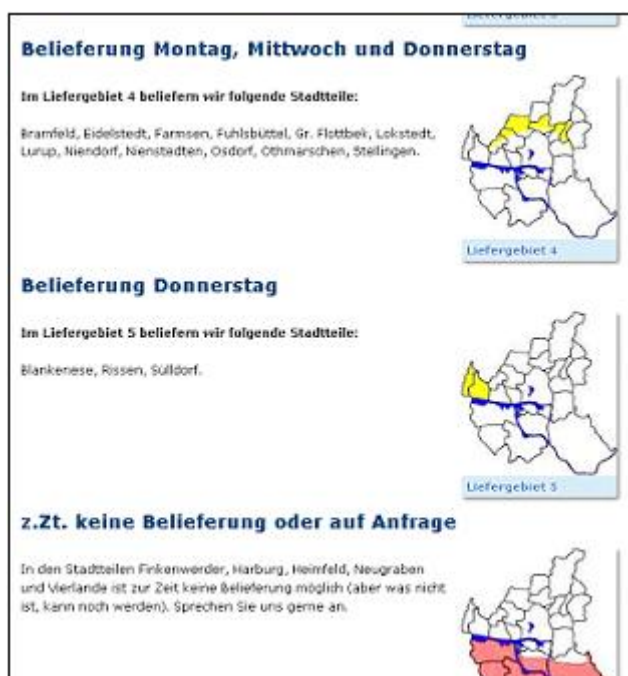


Abbildung 1: Grafische und textuelle Beschreibung des Liefergebietes

1.2.2 www.broetchenbursche.de

Bei diesem Webauftritt wird auch wieder eine Übersichtskarte eingesetzt, aber es ist hier nicht möglich zu erkennen, ob eine bestimmte Adresse beliefert wird. Das liegt daran, dass hier in einer Deutschlandkarte einfach die Bundesländer, in denen in einigen Bereichen geliefert wird, farblich hervorgehoben werden. Um genauere Informationen über den Lieferbereich zu erhalten, muss man auf das gewünschte Bundesland klicken. Dadurch wird eine neue Webseite eingeladen, auf der die verschiedenen Gebiete aufgelistet sind, in denen ausgeliefert wird.

1.2.3 www.lehmannsbio.de

Bei www.lehmannsbio.de erfolgt die Aufklärung über Liefergebiete zunächst textuell. Da es sich hier bei den Gebieten nicht um einfach zu benennende Stadtteile oder Ortschaften handelt, wird die Beschreibung kompliziert. Daher gibt es einen Link in der Beschreibung zu einer Karte auf der das Liefergebiet dargestellt ist. Eine Aussage zu Liefertagen ist aber weder dem Beschreibungstext noch der Übersichtskarte zu entnehmen.

1.2.4 www.mr-pizza.com

Hier werden zwei Ortschaften beliefert – Frankfurt und Bad Vilbel. Diese zwei Lieferbereiche werden noch einmal in weitere Gebiete aufgeteilt und anhand dieser Liefergebiete wird dann festgelegt, wie hoch der Mindestbestellwert ist und wie hoch der Zustellaufschlag. Die gesamten Informationen zum Liefergebiet werden auf dieser Webseite ausschließlich textuell bereitgestellt. Aufgrund der vielen verschiedenen Nebenbedingungen und 61 kleineren Liefergebieten ist es jedoch aufwändig die benötigten Informationen aus den Beschreibungen zu erhalten.

1.2.5 pizza.de

Dies ist eine Suchmaschine für Pizza Lieferservices. Wie man in Abbildung 2 sieht, kann man entweder die Postleitzahl seines Standortes eingeben oder auf einer Deutschlandkarte den Standort nach und nach auf eine Postleitzahl und auch auf eine einzelne Ortschaft innerhalb der Postleitzahl eingrenzen. Zum Schluss werden die verfügbaren Lieferdienste mit Mindestbestellsumme und Anfahrtskosten aufgelistet.

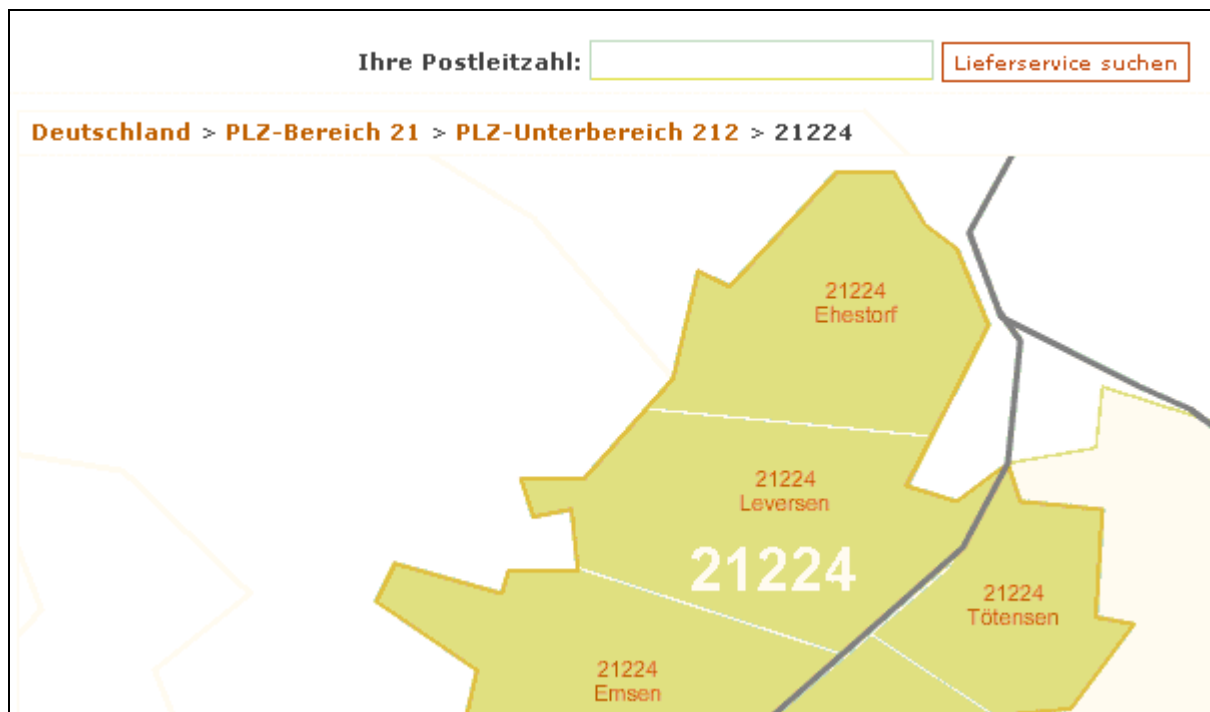


Abbildung 2: Eingabe des Standortes per Maus oder Eingabefeld

1.2.6 Fazit

Von einem Webseitennutzer wird leider viel zu häufig verlangt, dass er die für ihn wichtigen Informationen aus langen Texten oder einer Vielzahl an Grafiken herausfiltert. Bei der Zuordnung einer Adresse zu einem Servicebereich ist es nur sehr schwer möglich dies übersichtlich zu gestalten. Je mehr Servicegebiete und Nebenbedingungen vorhanden sind, desto aufwändiger wird es für den Webseitennutzer, um an die gewünschten Informationen zu gelangen. Einzig das Angebot von pizza.de macht einen positiven Eindruck. Die Möglichkeit, standortbezogene Informationen und eine grafische Rückmeldung über den eingegebenen Standort zu erhalten ist hier gut gelöst worden.

1.3 Zielsetzung

Es soll der Prototyp einer Webapplikation zur dynamischen Erstellung von Map-Mashup-Webseiten erstellt werden. Die zu erstellenden Webseiten sollen durch eine einfach zu erstellende XML-Datei konfigurierbar sein und es soll möglich sein, diese Konfigurationsdatei jederzeit zu verändern.

Die erstellten Webseiten sollen es auf einfache Art und Weise ermöglichen, Bereichsbezogene Informationen zu einer eingegebenen Adresse auszugeben. Die

Webanwendung soll dabei nicht nur Webseitenbetreiber unterstützen deren Inhalte dynamisch erstellt werden, sondern es soll auch möglich sein, die erzeugten Webseiten in Webauftritte mit statischen HTML-Seiten einzubinden.

1.4 Gliederung der Arbeit

1. Kapitel: Einleitung

Die Einleitung führt in das Thema der Arbeit ein. Sie wird mit der Motivation begonnen und gibt dann einen kurzen Überblick, wie das Problem der Bereitstellung von Informationen zu einem bestimmten geografischen Bereich auf anderen Webseiten gelöst wird. Anschließend wird das Ziel dieser Arbeit festgelegt und ihre Gliederung in Kurzform beschrieben.

2. Kapitel: Anforderungen an die Webapplikation

Um das Ziel dieser Arbeit zu erreichen, werden Anwendungsfälle und Szenarien erstellt und so die funktionalen und nicht-funktionalen Anforderungen an die Webapplikation gewonnen.

3. Kapitel: Webtechnologien

An dieser Stelle werden Technologien vorgestellt, die für diese Webapplikation von Bedeutung sind.

4. Kapitel: Design

Aus den Anforderungen und den vorher beschriebenen relevanten Webtechnologien wird hier das Design festgelegt. Dazu gehören das Design des Map-Mashup-Services, das Design des XML-Konfigurationstools, das Design des XML-Schemas der Konfigurationsdatei, das Sicherheitskonzept und wie mit Laufzeitfehlern umgegangen wird.

5. Kapitel: Auswahl der Systemkomponenten

Aus den vorgestellten Technologien wird sich hier für eine bestimmte Implementation entschieden.

6. Kapitel: Erstellen der XML-Schemata

Es werden die Schemata für die Benutzerdatei und für die Konfigurationsdatei nach dem vorgegebenen Designschema erstellt und getestet.

7. Kapitel: Einrichten der Entwicklungsumgebung

Es werden die benötigten Tools installiert und konfiguriert.

8. Kapitel: Implementierung

Es werden die beiden Komponenten Map-Mashup-Service und XML-Konfigurationstool implementiert.

9. Kapitel: Test

Es wird getestet, ob sich die erstellten Map-Mashup-Seiten in andere Webauftritte integrieren lassen.

10. Kapitel: Fazit und Ausblick

Es wird ein Überblick über die Arbeit gegeben und die erreichten Ziele bewertet. Zum Schluss wird noch ein Ausblick über mögliche Erweiterungen des Systems gegeben.

2 Anforderungen an die Webapplikation

Um die Anforderungen an die Webapplikation zu erhalten, wurden Anwendungsfälle und verschiedene Szenarien erstellt. Die sich daraus ergebenden funktionalen und nicht-funktionalen Anforderungen an die Webapplikation werden abschließend in einer Liste aufgeführt. Da nicht alle nicht-funktionalen Anforderungen aus den Anwendungsfällen hervorgehen, wurde diese Liste direkt mit weiteren nicht-funktionalen Anforderungen erweitert.

2.1 Anwendungsfälle

Anwendungsfälle stellen das System aus Sicht seiner Nutzer dar. Es werden Aktionen dargestellt, die ein Anwender mit dem System ausführen kann, um zu einem für ihn nützlichen Ergebnis zu gelangen. Das nachfolgende Anwendungsfall-Diagramm zeigt die verschiedenen Akteure des Systems und ihre möglichen Aktionen mit dem System. Diese sind für den Webseitennutzer das Anfordern einer Map-Mashup-Seite und das Abfragen eines standortbezogenen Textes. Die Aktion „Standortbezogenen Text abfragen“ kann durch Angabe von Speicherparametern durch die Aktion „Kundendaten speichern“ erweitert werden. Der Webseitenanbieter kann Kundendaten anfordern und eine XML-Konfigurationsdatei erstellen. Hat er alle erforderlichen Informationen für die Konfigurationsdatei eingegeben, dann wird ihm zusätzlich die Aktion „XML-Konfigurationsdatei herunterladen“ angeboten. Dem Administrator stehen die beiden Aktionen „Benutzerdaten pflegen“ und „Systemparameter einstellen“ zur Verfügung.

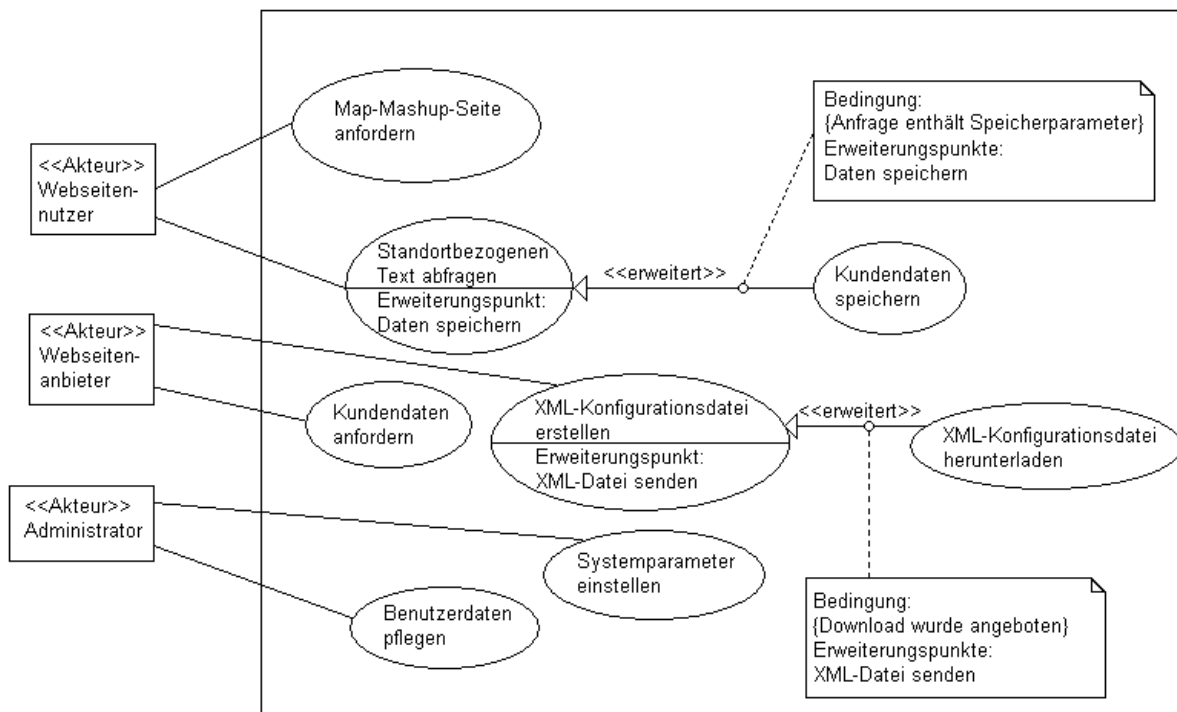


Abbildung 3: Anwendungsfall-Diagramm

2.2 Szenarien

Szenarien beschreiben Abläufe von Anwendungsfällen. Dabei werden der typische Ablauf und Abweichungen beschrieben.

Anwendungsfall	Standortbezogenen Text abfragen	
Akteur	Webseitenutzer	
Vorbedingung	Webnutzer hat Map-Mashup-Seite empfangen	
Auslöser	Es wird standortbezogener Text angefordert	
Nachbedingung	Standortbezogener Text wurde übertragen	
spezielle Anforderungen	Informationsaustausch ist verschlüsselt	
Hauptzenario	1	Aus Anfrage Standortinformationen entnehmen
	2	Anhand der Standortinformationen Text bereitstellen
	3	Text an Nutzer senden
Erweiterungspunkt	4	Daten speichern

Tabelle 1: Standortbezogenen Text abfragen

Anwendungsfall	Map-Mashup-Seite anfordern	
Akteur	Webseitennutzer	
Vorbedingung	Keine	
Auslöser	Es wird eine Map-Mashup-Seite angefordert	
Nachbedingung	Die angeforderte Seite wurde übertragen	
spezielle Anforderungen	Keine	
Hauptszenario	1	Aus der Anfrage Informationen zur gewünschten Seite entnehmen
	2	Anhand der Informationen XML-Konfigurationsdatei einladen
	3	Aus den Angaben in der XML-Konfigurationsdatei Map-Mashup-Seite aufbauen
	4	Map-Mashup-Seite an Webseitennutzer senden
Fehlerszenario 1		Anfrage enthält keine Informationen
	1.1	Webseitennutzer Willkommen-Seite senden
Fehlerszenario 2		Informationen aus Anfrage sind nicht nutzbar
	1.1	Webseitennutzer Fehlerseite senden
Fehlerszenario 3		XML-Konfigurationsdatei lässt sich nicht einladen
	2.1	Webseitennutzer Fehlerseite senden
Fehlerszenario 4		XML-Konfigurationsdatei ist fehlerhaft
	3.1	Nutzer Fehlerseite senden

Tabelle 2: Map-Mashup-Seite anfordern

Anwendungsfall	Daten speichern	
Akteur	Webseitennutzer	
Vorbedingung	Es wurde standortbezogener Text angefordert	
Auslöser	In der Anfrage nach standortbezogenen Text befinden sich Parameter zur Datenspeicherung	
Nachbedingung	Eingegebene Daten des Webseitennutzers wurden im Dateisystem gespeichert	
spezielle Anforderungen	Informationsaustausch ist verschlüsselt	
Hauptszenario	1	Aus Anfrage werden notwendige Informationen entnommen
	2	Es werden Standortinformationen hinzugefügt
	3	Informationen werden im Kundendatenordner unter angegebenen Dateinamen im angegebenen Unterordner gespeichert

Tabelle 3: Daten speichern

Anwendungsfall	Kundendaten anfordern	
Akteur	Webseitenanbieter	
Vorbedingung	Keine	
Auslöser	Es werden Kundendaten angefordert	
Nachbedingung	Die angeforderten Kundendaten wurde gesendet	
spezielle Anforderungen	Informationsaustausch ist verschlüsselt	
Hauptszenario	1	Aus der Anfrage werden Informationen zum Dateinamen und zum Unterordner entnommen
	2	Das File wird gesendet
Fehlerszenario 1		Es liegen keine Informationen zum Dateinamen oder zum Unterordner vor
	1.1	Es wird eine Fehlermeldung zurückgeliefert
Fehlerszenario 2		Die Datei existiert nicht
	2.1	Es wird eine Fehlermeldung zurückgeliefert

Tabelle 4: Kundendaten anfordern

Anwendungsfall	XML-Konfigurationsdatei erstellen	
Akteur	Webseitenanbieter	
Vorbedingung	Keine	
Auslöser	Es wird die Konfigurationstoolseite angefordert	
Nachbedingung	Die Konfigurationstoolseite wurde verlassen	
spezielle Anforderungen	XML-Konfigurationsdatei soll möglichst einfach zu erstellen sein	
Hauptszenario	1	Es wird die erste Seite der Konfigurationstoolseite zurückgeliefert
	2	Es werden die nächsten Seiten zurückgeliefert
	3	Es wird die XML-Konfigurationsdatei zum herunterladen angeboten
	4	Es wird die XML-Konfigurationsseite verlassen
Alternativszenario 1		Es wird auf die „zurück“-Schaltfläche gedrückt.
	2.1– 3.1	Es wird die vorherige Seite angezeigt
Alternativszenario 2		Es wird auf das Logo gedrückt
	1.1 - 3.1	Die XML-Konfigurationsseite wird verlassen
Erweiterungspunkt	3.1	XML-Datei senden

Tabelle 5: XML-Konfigurationsdatei erstellen

Anwendungsfall	XML-Konfigurationsdatei herunterladen	
Akteur	Webseitenanbieter	
Vorbedingung	Download wurde angeboten	
Auslöser	Es wurde auf die Schaltfläche zum Herunterladen der XML-Konfigurationsdatei gedrückt	
Nachbedingung	XML-Konfigurationsdatei wurde gesendet	
spezielle Anforderungen	Keine	
Hauptzenario	1	Aus den eingegebenen Daten wird eine XML-Konfigurationsdatei erstellt
	2	Die erstellte Datei wird gesendet

Tabelle 6: XML-Konfigurationsdatei herunterladen

Anwendungsfall	Systemparameter einstellen	
Akteur	Administrator	
Vorbedingung	Keine	
Auslöser	Systemparameter sollen eingestellt werden	
Nachbedingung	Systemparameter wurden vom System übernommen	
spezielle Anforderungen	Keine	
Hauptzenario	1	Parameterwerte werden geändert
	2	Parameterwerte werden dem System bekannt gemacht
	3	Die geänderten Werte werden vom System genutzt
Fehlerszenario 1	2.1	Die Einstellungen weisen Fehler auf Das System gibt eine Fehlermeldung aus und setzt die fehlerhaften Parameterwerte auf Standardwerte
Fehlerszenario 2	2.1	Die Einstellungen weisen Fehler auf Das System gibt eine Fehlermeldung aus

Tabelle 7: Systemparameter einstellen

Anwendungsfall	Benutzerdaten pflegen	
Akteur	Administrator	
Vorbedingung	Keine	
Auslöser	Benutzerdaten sollen verändert, hinzugefügt oder gelöscht werden	
Nachbedingung	Benutzerdaten wurden verändert, hinzugefügt oder gelöscht	
spezielle Anforderungen	Keine	
Hauptzenario	1	Daten werden bearbeitet
	2	Daten werden dem System bekannt gemacht
	3	Die geänderten Werte werden vom System genutzt
Fehlerszenario	2.1	Die Einstellungen weisen Fehler auf Das System gibt eine Fehlermeldung aus

Tabelle 8: Benutzerdaten pflegen

2.3 Funktionale Anforderungen

Funktionale Anforderungen treffen eine Aussage über zu erfüllende Funktionen einer Anwendung. Diese Anforderungen wurden in den Anwendungsfällen und Szenarien herausgearbeitet und werden jetzt noch einmal aufgelistet.

1. Webanwendung ist konfigurierbar

Ergibt sich aus dem Anwendungsfall „Systemparameter einstellen“

2. Benutzerdaten können gepflegt werden

Ergibt sich aus dem Anwendungsfall „Benutzerdaten pflegen“

3. Die zu erstellende Seiten können mit XML-Konfigurationsdatei angepasst werden

Ergibt sich aus dem Schritt 3 im Hauptszenario des Anwendungsfalls „Map-Mashup-Seite anfordern“

4. Die eingegebene Daten der Webseitennutzer können gespeichert werden

Ergibt sich aus dem Anwendungsfall „Daten speichern“

5. Webseitenanbieter können auf eingegebene Daten der Webseitennutzer zugreifen

Ergibt sich aus dem Anwendungsfall „Kundendaten anfordern“

6. Es werden standortbezogene Texte ausgegeben

Ergibt sich aus dem Anwendungsfall „Standortbezogenen Text abfragen“

7. Die erstellte Seite ist ein Map-Mashup

Ergibt sich aus den Schritten 3 und 4 im Hauptszenario des Anwendungsfalls „Map-Mashup-Seite anfordern“

2.4 Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben wie eine Anwendung eine Funktion erfüllen soll. Da nicht alle nicht-funktionalen Anforderungen in den Anwendungsfällen und Szenarien auftauchen, wurden die fehlenden Anforderungen der Liste hinzugefügt.

1. Eingabedaten werden geschützt übertragen

Ergibt sich aus den speziellen Anforderungen „Informationsaustausch ist verschlüsselt“ in den Anwendungsfällen „Standortbezogenen Text abfragen“, „Daten speichern“ und „Kundendaten anfordern“

2. Erstellung der XML-Konfigurationsdatei ist möglichst einfach

Ergibt sich aus der speziellen Anforderung „XML-Konfigurationsdatei soll möglichst einfach zu erstellen sein“ im Anwendungsfall „XML-Konfigurationsdatei erstellen“

3. Einfache Integration in Webauftritte

Die Zielgruppe der Webanwendung liegt bei kleinen und mittleren Unternehmen. Es ist daher möglich, dass die Unternehmenswebseite nicht professionell betreut wird und es daher niemand gibt, der fundierte Kenntnisse über HTML besitzt.

4. Benutzung von Open Source Technologien

Neben der Möglichkeit Open Source Software kostenlos zu nutzen, ist auch die Offenlegung des Quellcodes ein wichtiger Grund, um Open Source Software zu nutzen. So können Programme den eigenen Bedürfnissen angepasst werden und Programmfehler werden von der Entwicklergemeinde oft sehr zeitnah korrigiert.

5. XML-Konfigurationsdatei kann jederzeit verändert werden

Damit die Webseitenanbieter die standortspezifischen Texte aktuell halten können, ist es erforderlich, dass sie jederzeit einen einfachen Zugriff auf die XML-Konfigurationsdatei bekommen.

3 Webtechnologien

In diesem Kapitel werden Webtechnologien vorgestellt, die für diese Arbeit eine wichtige Rolle spielen.

3.1 Servlets

Servlets wurden in dieser Arbeit eingesetzt, um Webseiten zu erstellen, die nur einen geringen Anteil an statischen Inhalten, wie z.B. HTML-Tags, besitzen.

Servlets sind serverseitig laufende Java-Programme, die Client-Anfragen entgegennehmen und dynamisch erstellte Webseiten zurückliefern. Sie werden vor allem dann benutzt, wenn die zu erstellende Webseite wenig statischen Inhalt besitzt. Das Problem mit den statischen Inhalten ist, dass sie durch einfache print-Befehle in einen Ausgabestrom geschrieben werden. Das macht zum einen das Servlet unübersichtlicher und zum anderen ist diese Methode auch fehlerträchtig, weil der Inhalt der print-Befehle nicht daraufhin überprüft wird, ob sie korrekten HTML-Code ergeben. Für Webseiten mit einem etwas größeren Anteil an statischen Inhalten werden daher andere Technologien benutzt.

Servlets laufen in Containern, durch die sie auch erstellt und beendet werden. Die besondere Stärke der Servlets ist dabei, dass sie auf die gesamte Java-API zugreifen können. Ein Servlet muss das Interface `Servlet` implementieren. Dieses besteht aus den 5 Methoden:

- `void init (ServletConfig config)`
Sie wird aufgerufen, wenn das Servlet neu erstellt wird
- `ServletConfig getServletConfig()`
Diese Methode gibt ein `ServletConfig`-Objekt zurück, mit dem man z. B. Zugriff auf Initialisierungsparameter bekommt
- `String getServletInfo()`
Diese Methode nutzen z.B. Web Server Tools, um Informationen über Servlets auszugeben
- `void service(ServletRequest req, ServletResponse res)`
Hier werden das `ServletRequest`- und das `ServletResponse`-Objekt zur Verfügung gestellt, mit denen man Daten aus der Anfrage des Clients lesen und Daten an den Client senden kann

- `void destroy()`

Diese Methode wird aufgerufen, bevor das Servlet beendet wird

In Web-Applikationen werden sehr häufig `HttpServlets` benutzt, um Anfragen eines Client zu bearbeiten. Sie implementieren zusätzliche Methoden wie `doDelete`, `doPut`, `doGet` und `doPost`, um so direkt auf die entsprechende Anfragemethode des Clients zugreifen zu können.

Filter wurden in der Version 2.3 der Servlet Spezifikation eingeführt. Sie ermöglichen es Request- und Response-Objekte zu verändern, bevor sie an ein Servlet weitergeleitet oder zum Client gesendet werden. Dadurch ist eine transparente Manipulation der Request- und Response-Objekte möglich. Eingesetzt werden sie z.B. bei der Kompression von Daten, der Authentifizierung von Nutzern oder bei der Anpassung des Ausgabeformats an unterschiedliche Clients. Es können mehrere Filter einer Ressource zugeordnet werden oder auch ein Filter mehreren Ressourcen. Filter implementieren die Methode `doFilter`, die von einer Filter-Kette aufgerufen wird. In der Filterkette befinden sich alle Filter, die ein Request- oder Response-Objekt nacheinander durchlaufen muss.

Listener gehören ebenfalls seit Version 2.3 zur Servlet Spezifikation. Mit ihnen ist es möglich auf verschiedene Ereignisse zu reagieren. Es existieren dazu die folgenden Interfaces:

- `javax.servlet.ServletContextListener`:
Ein ServletContext wurde erstellt oder wird demnächst gelöscht
- `javax.servlet.http.HttpSessionListener`:
Eine HTTP-Session wurde erstellt, verworfen oder ungültig
- `javax.servlet.ServletContextAttributeListener`:
In einem Servlet-Context wurde ein Attribut hinzugefügt, entfernt oder verändert
- `javax.servlet.http.HttpSessionAttributeListener`:
In einer HTTP-Session wurde ein Attribut hinzugefügt, entfernt oder verändert

Servlets, Filter und Listener werden in der `web.xml` Konfigurationsdatei definiert. Die genauen Spezifikationen von Servlets kann man unter [JCP] nachlesen.

3.2 JSP

Webseiten mit einem größeren Anteil an statischen Inhalten wurden in dieser Arbeit als Java Server Page realisiert.

Java Server Pages sind eine Weiterentwicklung der Servlet-Technologie und auch ihr Haupteinsatzzweck besteht darin, dynamisch Webseiten mit Java zu erzeugen. Sie werden beim ersten Aufruf in ein Servlet umgewandelt. Im Gegensatz zu Servlets können in JSP's aber HTML-Tags direkt benutzt werden. So ist eine normale statische HTML-Seite auch eine gültige JSP-Seite. Dies vereinfacht die Benutzung von statischen Inhalten in Webseiten natürlich sehr, zumal es auch viele Tools gibt, mit denen JSP's grafisch bearbeitet werden können. Notwendiges dynamisches Verhalten kann in JSP's z.B. durch Scriptlets erzeugt werden. Eine Java Server Page Seite kann unter anderem aus folgenden Elementen bestehen:

HTML-Tags:

Sie werden nicht verarbeitet und genauso wieder ausgegeben

Scriptlets:

Sie enthalten Java-Code, der ausgewertet wird. Durch Nutzung des Objektes „out“ werden Teile der zu erstellenden Seite hinzugefügt.

Ausdrücke:

Der String-Wert des Ausdruckes wird der zu erstellenden Seite hinzugefügt.

JSP-Direktiven:

Sie geben Anweisungen an den JSP-Container, um z.B. benötigte Java-Klassen oder Tag-Libraries in die Seite einzubinden, machen aber selber keine direkten Ausgaben auf die zu erstellende Seite.

JSP-Tags:

Dies sind spezielle Tags, mit denen man auf häufig gebrauchte Funktionen einfach zuzugreifen kann. Einige dieser Funktionen sind das Einbinden von anderen Ressourcen zur Laufzeit (z.B. JSP-Fragmente), das Manipulieren von Java-Beans oder das Weiterleiten von Anfragen an andere Ressourcen.

Custom-Tags:

Mithilfe von eigenen Tags ist es möglich, Java-Code aus den JSP's auszulagern und so auf die Benutzung von Scriptlets und Ausdrücken zu verzichten.

3.3 Cascading Stylesheets

Cascading Stylesheets wurden hier eingesetzt, um eine größere Kontrolle über das Aussehen einzelner Web-Elemente zu bekommen. Mit ihnen konnten Webseiten ein einheitliches Layout erhalten und Änderungen konnten zentral erfolgen.

CSS sind der Versuch das Layout einer Webseite von seinem Inhalt zu trennen. Das Layout der Elemente wird in einem eigenständigen Bereich im HTML-Dokument festgelegt. Dieser Bereich kann sich dabei innerhalb des HTML-Dokumentes befinden und durch ein style-Tag eingeschlossen sein oder er befindet sich in einem extra Dokument und wird über das link-Tag eingebunden.

Die Verbindung zwischen dem HTML-Element und dem CSS erfolgt auf verschiedene Art und Weise.

1. Typ des Elements:

In Stylesheets kann auf alle Elemente eines bestimmten Typs zugegriffen werden. Dafür muss der Typ des Elements gefolgt von den gewünschten Formatierungen angegeben werden. Durch `p{color:#000000;}` erhalten z.B. alle p-Elemente eine schwarze Schriftfarbe (sofern sie nicht noch anders formatiert werden).

2. Klasse des Elements :

HTML-Elemente können mit dem Attribut „class“ einer Klasse zugewiesen werden. Auf diese Klasse kann wiederum in Stylesheets bezug genommen werden. Das HTML-Element `<p class="hervorheben">hallo</p>` kann z.B. mit dem Stylesheeteintrag `.hervorheben{color:#ff0000;}` rot hervorgehoben werden.

3. ID des Elements :

HTML-Elementen kann mit dem Attribut „id“ eine eindeutige ID zugewiesen werden. Auf diese ID kann wieder in Stylesheets zugegriffen werden. Dem HTML-Element `<p id="blau">hallo</p>` kann mit dem Stylesheeteintrag `#blau{color:#0000ff}` eine blaue Schriftfarbe zugewiesen werden.

Diese Arten der Formatzuweisung an HTML-Elementen haben unterschiedliche Wertigkeiten. Je spezifischer eine Zuweisung ist, desto werthaltiger ist sie auch. Eine Formatzuweisung über den Typen eines Elements erhält so die Wertigkeit 1, eine Zuweisung über die Klasse eines Elements die Wertigkeit 10 und eine Zuweisung über die ID eines Element die Wertigkeit 100. Somit ist gewährleistet, dass die

spezielle Formatierung eines Elements nicht durch eine allgemeinere überschrieben werden kann. Haben zwei Formatierungsangaben die gleiche Wertigkeit, so hat die später auftretende Formatierungsangabe den Vorrang.

3.4 XML / Java Binding

Mit der Hilfe von XML / Java Binding wurden Daten aus Konfigurationsdateien verarbeitet und aus eingegebenen Daten Konfigurationsdateien erstellt.

XML steht für „Extensible Markup Language“ und ist ein Standard den das W3-Konsortium 1998 verabschiedet hat. Da XML aus einfachen Text besteht ist eine Portierung zwischen verschiedenen Systemen problemlos möglich. Alle Systeme, die XML standardkonform benutzen, können so auf gleiche XML-Dokumente zugreifen. Ein weiterer wichtiger Baustein in der Benutzung von XML-Dokumenten ist die Möglichkeit, sie mit Hilfe von XML-Schema Dateien auf ihre Gültigkeit zu überprüfen. Dadurch wird gewährleistet, dass XML-Dokumente nur erlaubte Elementtypen besitzen, die in einer erlaubten Struktur angeordnet sind und das die Elemente nur zugelassene Werte annehmen dürfen.

Java Binding übernimmt den Prozess des Umwandeln eines XML-Dokuments in ein Java-Objekt und umgekehrt. In dem Java-Objekt werden die Elemente des XML-Dokumentes auf weitere Java-Objekte abgebildet. Die Struktur des XML-Dokumentes bleibt dabei in dem Objektbaum erhalten. Für den Datenaustausch kann nun mit Getter- und Setter-Methoden auf die Objekte zugegriffen werden. Diese Systemunabhängigkeit hat dazu geführt, das XML heute eine wichtige Rolle beim Datenaustausch im Internet besitzt.

3.5 JavaScript / DOM

Die clientseitige Anwendungslogik wurde mit JavaScript realisiert und zusammen mit dem DOM wurde es erreicht, dass Inhalte und Aussehen von Webseitenelementen zur Laufzeit verändert werden konnten.

JavaScript wurde von dem Browserhersteller Netscape entwickelt. Es handelt sich um eine Programmiersprache, deren Code erst zur Laufzeit von einem JavaScript Interpreter in ausführbaren Maschinencode übersetzt wird. Dies führt dazu, dass JavaScript-Code systemunabhängig ist. JavaScript ist von der European Computer Manufacturers Association unter der Bezeichnung ECMA-262 als ein

Industriestandard definiert. Für den Internet Explorer beschreibt Microsoft jedoch eigene Wege. Er interpretiert eine eigene Scriptsprache „JScript“. JScript und JavaScript sind einander sehr ähnlich und es ist nur dann eine Anpassung des Scriptes erforderlich, wenn Objekte und Methoden genutzt werden, die in der jeweils anderen Scriptsprache nicht oder auf andere Weise implementiert sind.

Das Document Object Model DOM ist eine vom W3-Konsortium entwickelte Schnittstelle, um standardisiert den Inhalt, die Struktur und das Layout von XML- und HTML-Dokumenten dynamisch zu verändern.

3.6 AJAX

Ajax wurde im Zusammenspiel mit einigen Funktionen der Map-API benutzt, um zu erreichen, dass auch bei längeren Antwortzeiten auf Benutzereingaben reagiert werden kann.

Der Begriff Ajax wurde zuerst von Jesse James Garret in dem Artikel „Ajax:A New Approach to Web Applications“ [Gar05] genannt. Die ausgeschriebene Bezeichnung „Asynchron JavaScript and XML“ zeigt schon, dass Ajax aus mehreren Komponenten besteht. Diese sind:

- XML für den Datenaustausch
- DOM für den dynamischen Zugriff auf den Dokumentenbaum
- XMLHttpRequest-Objekt für die asynchrone Client-Server-Kommunikation
- JavaScript für die Steuerung aller Vorgänge

Das Zusammenspiel dieser Komponenten ermöglicht es Webanwendungen zu entwickeln, die sich wie ein installiertes Programm verhalten. So kann der Inhalt einer Webseite selektiv verändert werden und ein notwendiger Datenaustausch erfolgt asynchron im Hintergrund. Durch diese Maßnahmen kann eine Webanwendung direkt auf Benutzereingaben reagieren und Informationen anzeigen, ohne dass auf den Aufbau einer ganzen Webseite gewartet werden muss.

4 Design

Nachdem in Kapitel 2 die Anforderungen an das System herausgearbeitet wurden, sollen diese nun durch das Design umgesetzt werden. Das System teilt sich in zwei Hauptkomponenten auf. Die erste Komponente ist der Map-Mashup-Service, der dafür zuständig ist dynamische Map-Mashup-Seiten zu erstellen und mit den Clients der Webseitennutzer zu kommunizieren. Die zweite Komponente ist ein Tool zur Erstellung der XML-Konfigurationsdatei. Mit diesem Tool können Webseitenanbieter auf einfache Weise die XML-Konfigurationsdatei erstellen, um das Aussehen und Funktionen der dynamisch erstellten Webseite zu bestimmen.

4.1 Map-Mashup-Service

Der Map-Mashup-Service besteht aus mehreren Komponenten:

- Document-Objekt:
Bereitstellen von Methoden, um angeforderte Map-Mashup-Seiten erstellen zu können.
- User-Objekt:
Bereitstellen von Methoden, um auf Benutzerdaten zugreifen zu können.
- DownloadFilter:
Bereitstellen der Eingabedaten
- MmsFilter:
Kontrolle der Anfrage-Parameter
- FileLogger:
Loggen von Systemmeldungen
- FileDelete:
Automatisches Löschen von Eingabedaten
- InitMms:
Initialisierung der Document-, User-, FileDelete- und Logger-Komponente
- StartServlet:
Bereitstellen der Map-Mashup-Webseite
- ResponseServlet:
Speichern der Eingabedaten & Bereitstellen der Standortinformationen

4.1.1 Speicherort der XML-Konfigurationsdatei

In der nicht-funktionalen Anforderung Nr. 5 aus Kapitel 2.4 wurde gefordert, dass die Nutzer des Map-Mashup-Services die Möglichkeit bekommen sollen, die XML-Konfigurationsdatei jederzeit zu ändern. Um das zu erreichen, gab es zwei Möglichkeiten:

1. Speichern der XML-Konfigurationsdatei auf dem Map-Mashup-Server und bereitstellen einer up- und download-Funktion oder einer Möglichkeit die Konfigurationsdatei online zu bearbeiten
2. Der Map-Mashup-Service Nutzer stellt die XML-Konfigurationsdatei an einer beliebigen Adresse ins Internet und der Map-Mashup-Service nutzt sie von dort

Die erste Möglichkeit hätte für den Nutzer den Vorteil gehabt, dass er die XML-Konfigurationsdatei nicht selber hätte online stellen müssen. Da er aber sowieso eine Webseite besitzt, dürfte das für ihn eigentlich kein Problem darstellen. Auf der anderen Seite überwiegen aber die Vorteile der 2. Möglichkeit, denn der Nutzer kann selber entscheiden, wo er die XML-Konfigurationsdatei online stellt. So bietet sich dem einen oder anderen Benutzer vielleicht die Möglichkeit, die Datei automatisch zu aktualisieren. Ein weiterer Vorteil besteht für den Map-Mashup-Service, der nur eine einzige Benutzerdatei verwalten muss und auch keine Möglichkeiten zum up- und download der XML-Konfigurationsdateien einrichten muss. Aufgrund dieser Vorteile wurde sich dafür entschieden, dass die Benutzer die XML-Konfigurationsdatei online stellen.

4.1.2 Entscheidung Servlet / JSP

Die Entscheidung, ob Servlets oder JSP's verwendet werden sollen, wurde davon abhängig gemacht, wie groß der Anteil an statischen Inhalten in der zu erstellenden Webseite war.

Beim Servlet `startServlet` werden der gesamte Inhalt des HTML-Body Elements, große Teile der Style-Informationen und einige JavaScript Teile dynamisch erstellt, daher ist der Anteil an dynamisch erstellten Inhalten sehr groß. Durch die Ausgliederung der restlichen Style-Informationen und des JavaScript-Codes in eigene Dateien, konnte der Anteil an statischen Inhalten noch weiter verringert

werden und so das `StartServlet`-Servlet sehr kompakt und übersichtlich gehalten werden.

Das `ResponseServlet`-Servlet enthält ebenfalls nur wenig statischen Inhalt und ist auch sehr übersichtlich.

4.1.3 Persistente Daten

Der Map-Mashup-Service braucht nur sehr wenige Daten pro Nutzer. Dies ist die Benutzerkennung, die Webadresse der XML-Konfigurationsdatei und ein Passwort. Die Pflege der Benutzerdaten erfolgt über den Administrator. Damit es den Benutzern möglich ist die Webadresse der XML-Konfigurationsdatei zu verändern, müssen sie sich gegenüber dem Administrator autorisieren. Das geschieht über eine Email in der die Benutzerkennung und das Passwort enthalten sein müssen.

Die Benutzerdaten werden in einer XML-Datei bereitgestellt, die vom System gegen ein Schema geprüft wird. Dadurch wird gewährleistet, dass die Pflege der Benutzerdaten einfach über einen Texteditor erfolgen kann und die vom System genutzten Daten gültig sind.

Ein späterer Wechsel auf eine datenbankorientierte Benutzerverwaltung ist ebenfalls einfach möglich, da der Zugriff auf die Benutzerdaten ausschließlich in der Klasse „User“ stattfindet und somit nur dort Methoden verändert werden müssten.

4.1.4 Sequenzdiagramme

Die folgenden Sequenzdiagramme stellen den Nachrichtenaustausch zwischen verschiedenen Objekten des Systems dar. Die betrachteten Szenarien enthalten dabei den Standardablauf ohne Fehler.

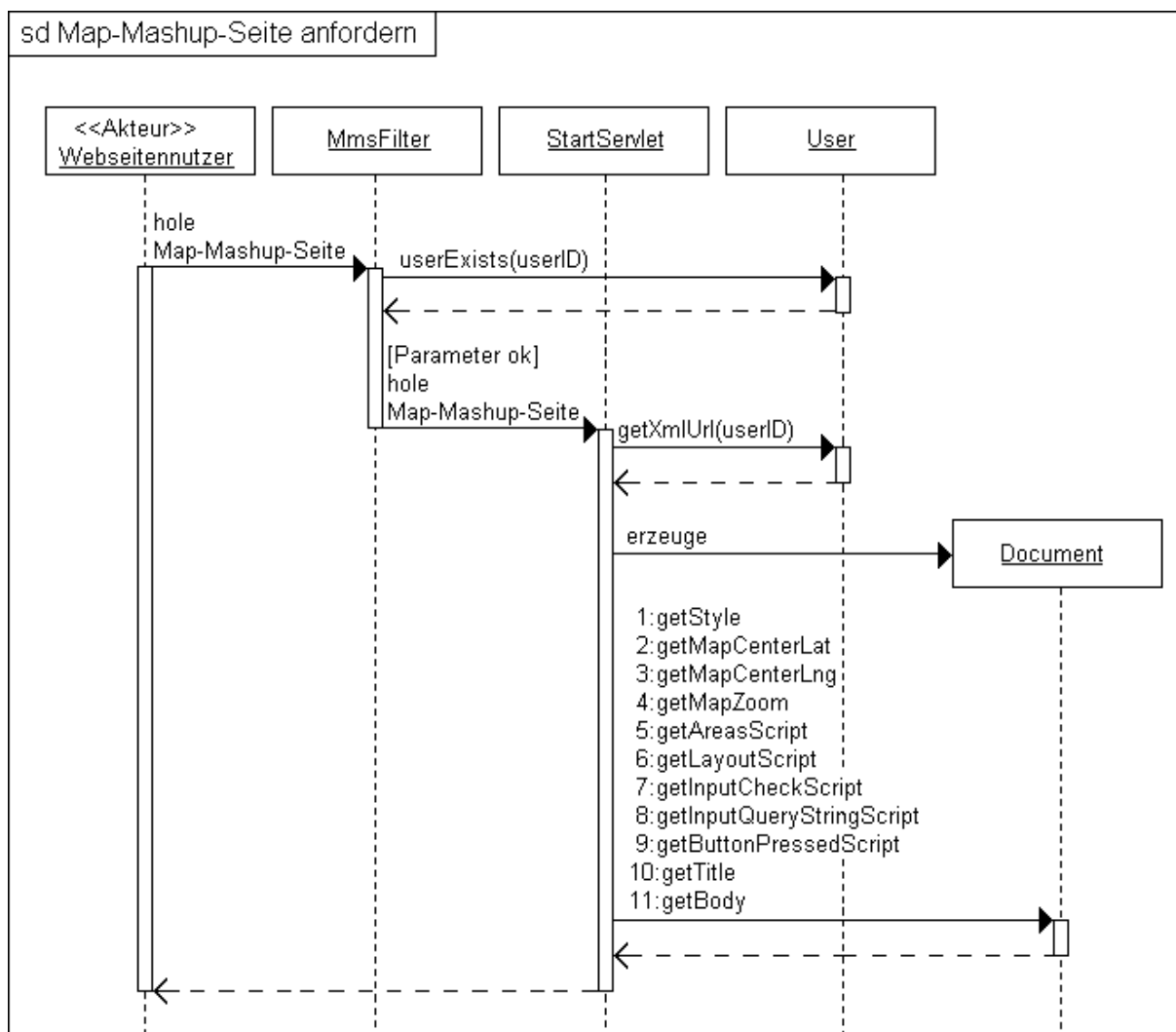


Abbildung 4: Sequenzdiagramm Map-Mashup-Seite anfordern

In dem vorangestellten Sequenzdiagramm fordert ein Webseitennutzer eine Map-Mashup-Seite an. Der Filter `MmsFilter` erhält die Anfrage als erstes. Er überprüft, ob alle erforderlichen Anfrageparameter enthalten und gültig sind. Wenn das der Fall ist, dann gibt er die Anfrage an das `StartServlet`-Servlet weiter. Dieses erhält von dem `User`-Objekt anhand des Parameters „`userID`“ die URL der benötigten XML-Konfigurationsdatei und erstellt so ein neues `Document`-Object. Mit Hilfe des `Document`-Objects baut das `StartServlet`-Servlet die Antwort auf die Anfrage des Webseitennutzers zusammen und sendet es ab.

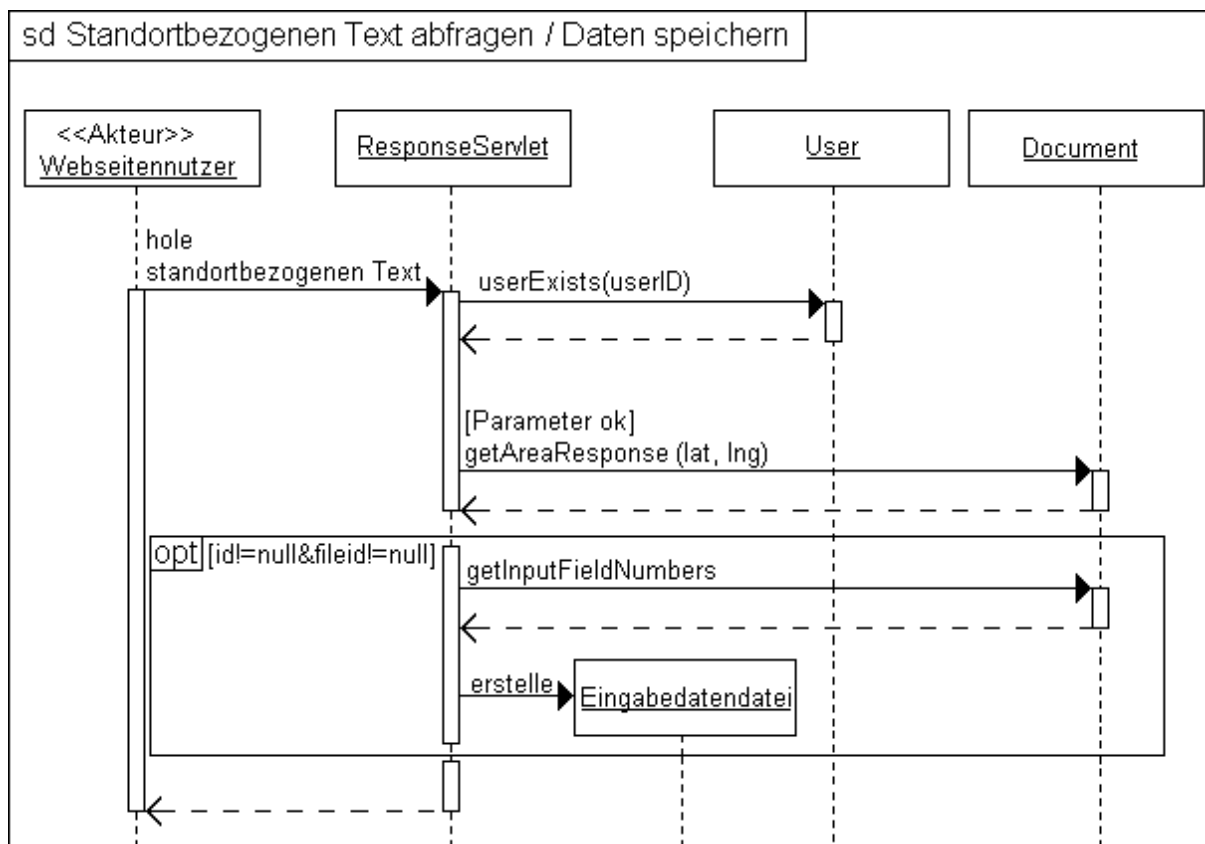


Abbildung 5: Sequenzdiagramm Standortbezogenen Text abfragen / Daten speichern

In dem in Abbildung 5 zu sehenden Sequenzdiagramm wird die Anfrage eines Webseitennutzers nach einem standortbezogenen Text dargestellt. Die Anfrage wird von dem ResponseServlet-Servlet entgegengenommen und der Parameter „userID“ wird mit Hilfe des User-Objektes überprüft. Wenn dieser Parameter gültig ist, dann werden mit den Parametern „lat“ und „lng“ die Standortinformationen vom Document-Objekt abgefragt.

Wenn in der Anfrage die Parameter „id“ und „fileid“ enthalten sind, werden vom Document-Objekt Informationen zu den Dateneingabefeldern abgefragt. Mit diesen Informationen werden die vom Webseitenbenutzer in die Eingabefelder eingegebenen Daten gewonnen und in einer Datei gespeichert die aus der fileid und der Endung „.xml“ besteht. Nach diesem optionalen Vorgang wird auf die Anfrage des Webseitennutzers geantwortet und ein XML-Dokument gesendet, das die gewünschten Standortinformationen enthält.

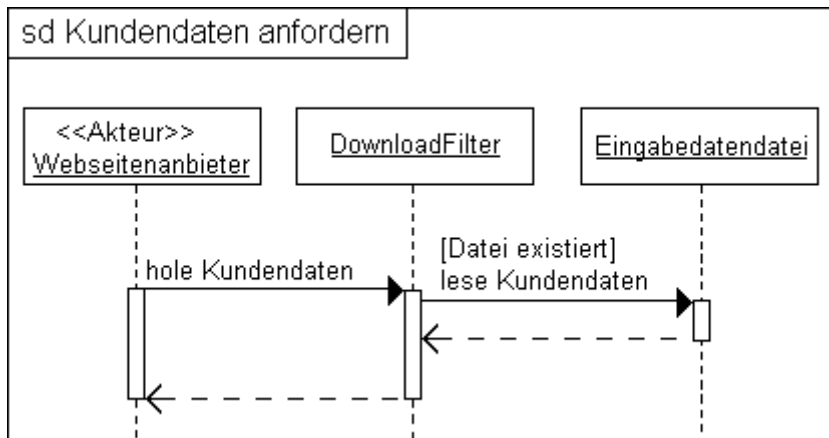


Abbildung 6: Sequenzdiagramm Kundendaten anfordern

In dem Sequenzdiagramm „Kundendaten anfordern“ versucht der Webseitenanbieter eine Eingabedatendatei herunterzuladen. Die Anfrage wird vom Filter `DownloadFilter` entgegengenommen und es wird überprüft, ob die Datei existiert. Wenn sie existiert, dann wird die Datei eingelesen und als Antwort auf die Anfrage an den Webseitenanbieter gesendet.

4.1.5 Klassendiagramm

In dem nachfolgenden Klassendiagramm wird gezeigt, wie einzelne Klassen der Map-Mashup-Webanwendung voneinander abhängen. Das Diagramm ist vereinfacht dargestellt und daher fehlen einige Klassen und deren Beziehungen an dieser Stelle.

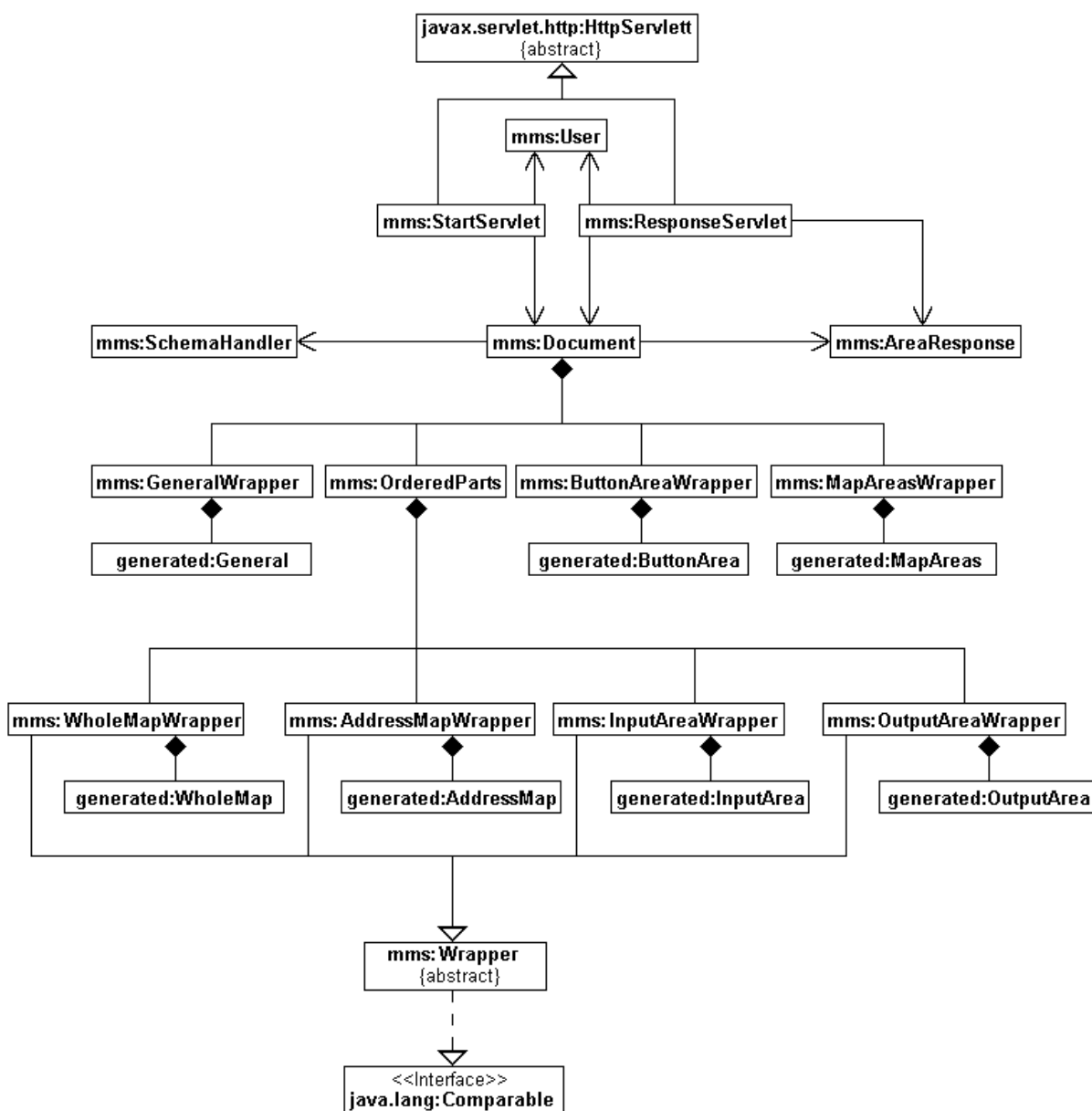


Abbildung 7: Klassendiagramm Map-Mashup-Service

Die beiden Klassen `StartServlet` und `ResponseServlet` erweitern die abstrakte Klasse `HttpServlet`. Sie benutzen die Klasse `User`, um zu einem gegebenen Parameter `userID` die URL der XML-Konfigurationsdatei zu erhalten und sie benutzen die Klasse `Document`, um Informationen über den Aufbau der zu

erstellenden Map-Mashup-Seite und Standortinformationen zu erhalten. Diese Standortinformationen werden mit Hilfe der Klasse `AreaResponse` vom `Document`-Objekt zum `ResponseServlet`-Objekt übertragen. Die Klasse `Document` hat Zugriff auf die Klasse `SchemaHandler`, um einen `JAXBContext` zu erhalten und so einen Marshaller oder Unmarshaller erstellen zu können. Ein `Document`-Objekt besteht aus den folgenden Komponenten.

GeneralWrapper: Diese Komponente besteht aus einem `General`-Objekt und stellt Informationen über die Hauptüberschrift, den Browsertitel, die Hintergrundfarbe und das Hintergrundmuster der zu erstellenden Map-Mashup-Seite bereit.

ButtonAreaWrapper: Diese Komponente besteht aus einem `ButtonArea`-Objekt und stellt Informationen über die optionalen Schaltflächen am unteren Rand der Map-Mashup-Seite bereit.

MapAreasWrapper: Diese Komponente besteht aus einem `MapAreas`-Objekt und stellt Informationen über die verschiedenen Kartenbereiche bereit.

OrderedParts: Diese Komponente besteht selber aus den nachfolgend beschriebenen vier Komponenten, die von der abstrakten Klasse `Wrapper` abgeleitet wurden. Die Klasse `Wrapper` implementiert das Interface `Comparable` und sorgt so dafür, dass die von `Wrapper` abgeleiteten Klassen sortiert werden können.

WholeMapWrapper: Besteht aus einem `WholeMap`-Objekt und stellt Informationen zur Übersichtskarte bereit.

AddressMapWrapper: Besteht aus einem `AddressMap`-Objekt und stellt Informationen zur Adresskarte bereit.

InputAreaWrapper: Besteht aus einem `InputArea`-Objekt und stellt Informationen zum Dateneingabebereich bereit.

OutputAreaWrapper: Besteht aus einem `OutputArea`-Objekt und stellt Informationen zum Datenausgabebereich bereit.

4.1.6 Listener

Der Listener `InitMms` implementiert das Interface `ServletContextListener` und erhält so Nachrichten beim Starten und Beenden der Webapplikation. Diese Nachrichten werden genutzt, um beim Start die Klassen `User`, `Document`,

`FileLogger` und `FileDelete` zu initialisieren und beim Beenden der Webapplikation Ressourcen wieder freizugeben.

4.1.7 Filter

Der Filter `MmsFilter` wird dann aktiv, wenn auf die Ressource „/start“ zugegriffen wird. Er wird vor dem `StartServlet`-Servlet aktiv, das auch unter dieser Ressource angesprochen wird. Die Aufgabe des Filters `MmsFilter` ist es, die Anforderung daraufhin zu überprüfen, ob alle notwendigen Parameter angegeben wurden und ob sie gültig sind.

Der Filter `DownloadFilter` wird aktiv, wenn auf eine Ressource unter „/download“ zugegriffen werden soll. Er überprüft, ob die angeforderte Ressource existiert und schreibt im Erfolgsfall die angeforderte Datei in den Antwortstrom.

4.1.8 Fazit Design Map-Mashup-Service

Es wurden die benötigten Komponenten, ihre Struktur, ihre Aufgaben und die Beziehungen zwischen ihnen dargestellt. Obwohl einige Klassen automatisch mit dem Binding-Kompiler erstellt wurden und es so erforderlich war für sie Wrapper-Klassen zu erstellen, konnte die Komplexität einzelner Komponenten durch Delegation verringert werden.

4.2 Tool zur Erstellung der Konfigurationsdatei

Das Tool zur Erstellung der XML-Konfigurationsdatei für die Map-Mashup-Seite soll dazu dienen, XML-Konfigurationsdateien über eine einfach zu benutzende Webanwendung zu erstellen.

4.2.1 Entscheidung Servlet / JSP

Ebenso wie bei dem Map-Mashup-Service wurde auch hier die Entscheidung für die Benutzung von Servlets oder JSP davon abhängig gemacht, wie viele statische Elemente in den einzelnen Dokumenten enthalten sind. Da die einzelnen Dokumente des Tools überwiegend aus statischen Elementen bestehen, wurden diese als JSP entworfen. Die Klasse `ConfigToolServlet` dient als Controller des

Konfigurationstools und besitzt keine statischen Inhalte. Sie wurde daher als Servlet realisiert.

4.2.2 Custom Tags

Bei einigen JSP's werden dynamische Elemente benötigt. Dies kann einerseits mit Scriptlets realisiert werden, die Java-Code enthalten. JSP's mit Scriptlets sind durch die Vermischung von HTML-Tags und Java-Code allerdings unübersichtlicher und aufwendiger zu warten. Durch die Verwendung der Standard-Tag-Library kann das aber verhindert werden. Sie ermöglicht es, eigene Tags zu erstellen und so den Java-Code aus den JSP's zu entfernen. So werden die JSP's übersichtlicher und ermöglichen es Personen ohne Java-Kenntnisse sie zu warten. Ein weiterer Vorteil ist die Trennung der Logik von der Darstellungskomponente, weil somit Teile der Logik wiederverwendet werden können.

4.2.3 Cascading Stylesheet

Cascading Stylesheets ermöglichen es die Aufgaben der Darstellungskomponente weiter aufzuteilen. Es kann jetzt unterschieden werden was dargestellt werden soll und wie es dargestellt werden soll. Für das „was“ ist weiterhin der HTML-Teil zuständig, wie etwas dargestellt werden soll, dafür ist ein Cascading Stylesheet verantwortlich. Durch die Nutzung von CSS ist es einfacher Möglich ein einheitliches Layout zu erstellen. Dafür wird ein zentrales CSS erstellt, das alle JSP's referenzieren. Damit ist sichergestellt, das Layoutänderungen an einer zentralen Stelle vorgenommen und Elemente in verschiedenen JSP's angesprochen werden können.

4.2.4 JavaScript

Für einige Funktionen des Tools ist es notwendig JavaScript-Code auf dem Client-Webbrowser ablaufen zu lassen. Es ist daher unbedingt notwendig, dass JavaScript im Webbrowser aktiviert ist. Dies wird mit Hilfe einer JavaScript-Funktion auf allen JSP's kontrolliert und nur dann die Webseite angezeigt. Der JavaScript-Code wird auf zwei Arten in die JSP's eingebunden. In einer externen JavaScript-Datei steht Code, der von mehreren JSP's genutzt wird. Code der nur von einer einzigen Seite

gebraucht wird oder der für jede Seite unterschiedlich ist (z.B. `checkForm` um Eingabefelder zu überprüfen) wird direkt im Bereich Head des Dokuments eingefügt.

4.2.5 Sequenzdiagramme

Die folgenden beiden Sequenzdiagramme stellen den typischen, fehlerfreien Ablauf der Erstellung einer XML-Konfigurationsdatei dar.

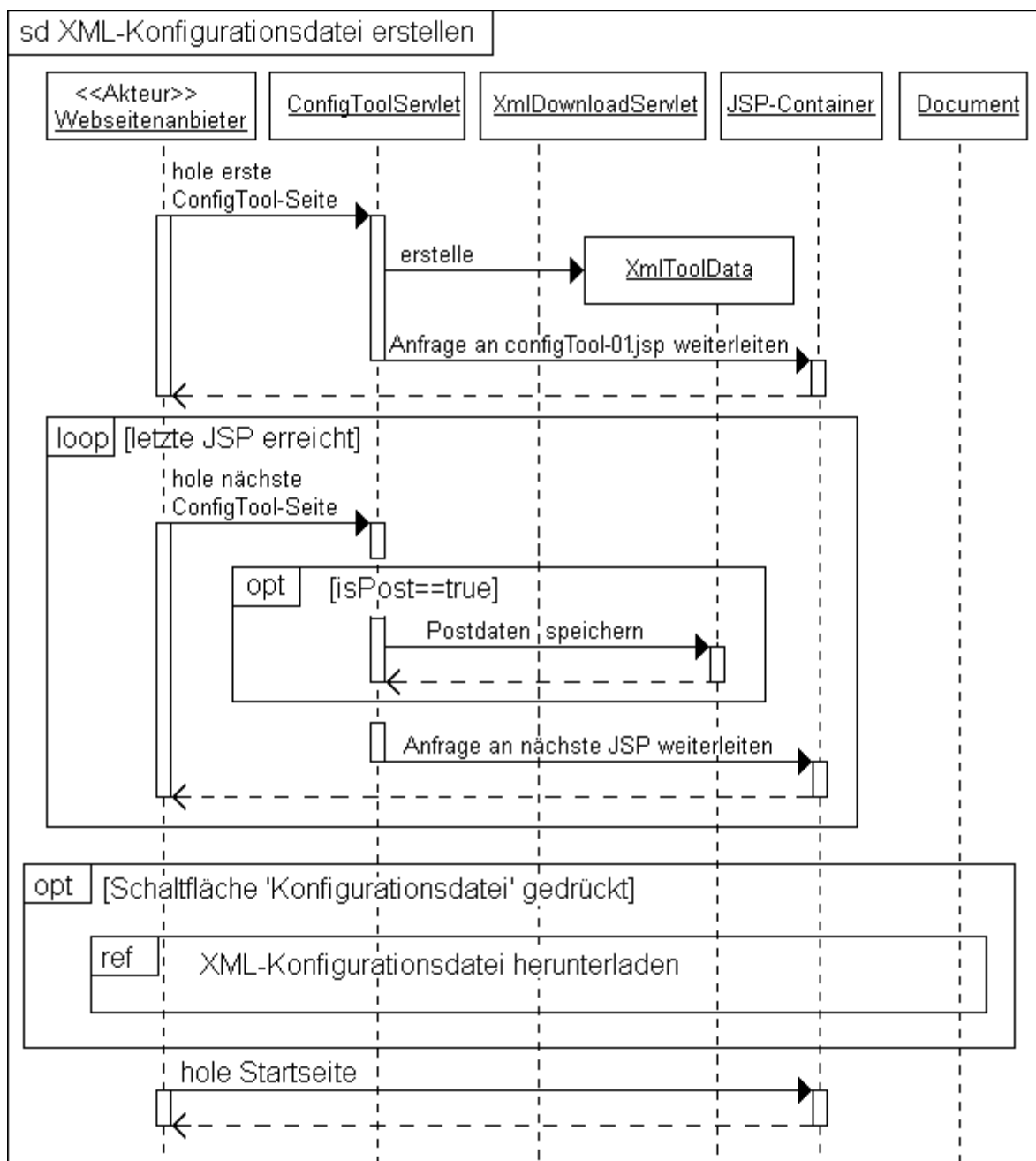


Abbildung 8: Sequenzdiagramm Erstellung einer XML-Konfigurationsdatei

In dem Sequenzdiagramm „Erstellung einer XML-Konfigurationsdatei“ fordert ein Webseitenanbieter zunächst die erste Seite der ConfigTool-Webanwendung an. Das

Servlet „ConfigToolServlet“ empfängt die Anfrage und erstellt ein XMLToolData-Objekt. Danach wird die Anfrage an die erste JSP-Seite weitergeleitet. Die folgenden Anfragen werden ebenfalls von dem ConfigToolServlet-Servlet entgegengenommen. Ist die Anfragemethode vom Typ „post“, dann werden aus der Anfrage die Postdaten entnommen und in dem XmlToolData-Objekt gespeichert. Nach diesem optionalen Zwischenschritt werden alle Anfragen auf die gewünschte JSP-Seite weitergeleitet. Drückt der Webseitenanbieter auf der letzten JSP-Seite auf die Schaltfläche „Konfigurationsdatei“, dann wird der Anwendungsfall „XML-Konfigurationsdatei herunterladen“ ausgeführt. Mit dem Anfordern der Startseite wird vom JSP-Container die Übersichtsseite des Projektes zurückgegeben.

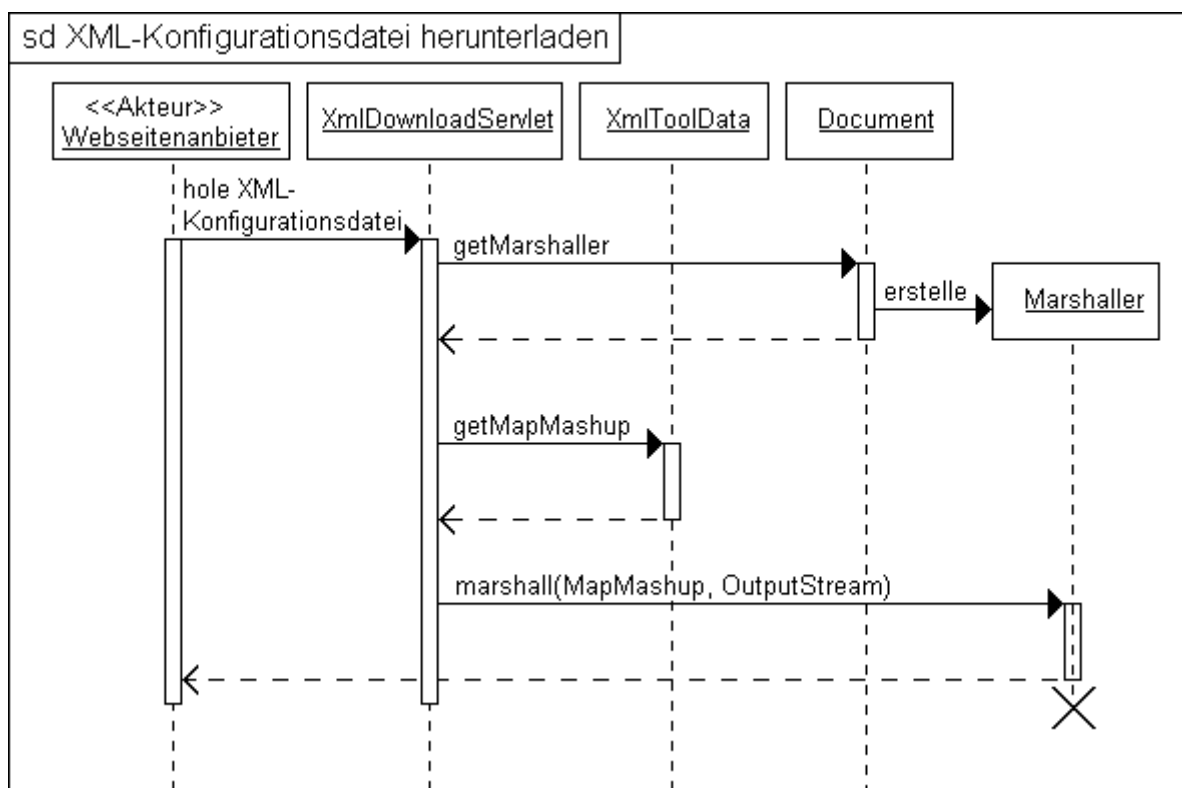


Abbildung 9: Sequenzdiagramm XML-Konfigurationsdatei herunterladen

In dem abgebildeten Sequenzdiagramm „XML-Konfigurationsdatei herunterladen“ fordert ein Webseitenanbieter die zuvor erstellte XML-Konfigurationsdatei an. Die Anforderung wird vom XMLDownloadServlet-Servlet entgegengenommen, das daraufhin vom Document-Objekt ein Marshaller-Objekt anfordert. Das Document-Objekt erstellt das Marshaller-Objekt und gibt es an das XMLDownloadServlet-Servlet. Nach Erhalt des Marshaller-Objektes fordert das XMLDownloadServlet-

Servlet vom `XmlToolData`-Objekt ein `MapMashup`-Objekt an, das alle Konfigurationswerte in einer vom Marshaller geforderten Struktur enthält. Dem Marshaller-Objekt wird das `MapMashup`-Objekt und ein Ausgabestrom-Objekt übergeben und so ein XML-Dokument als Antwort auf die Anfrage des Webseitenanbieters erstellt.

4.2.6 Klassendiagramm

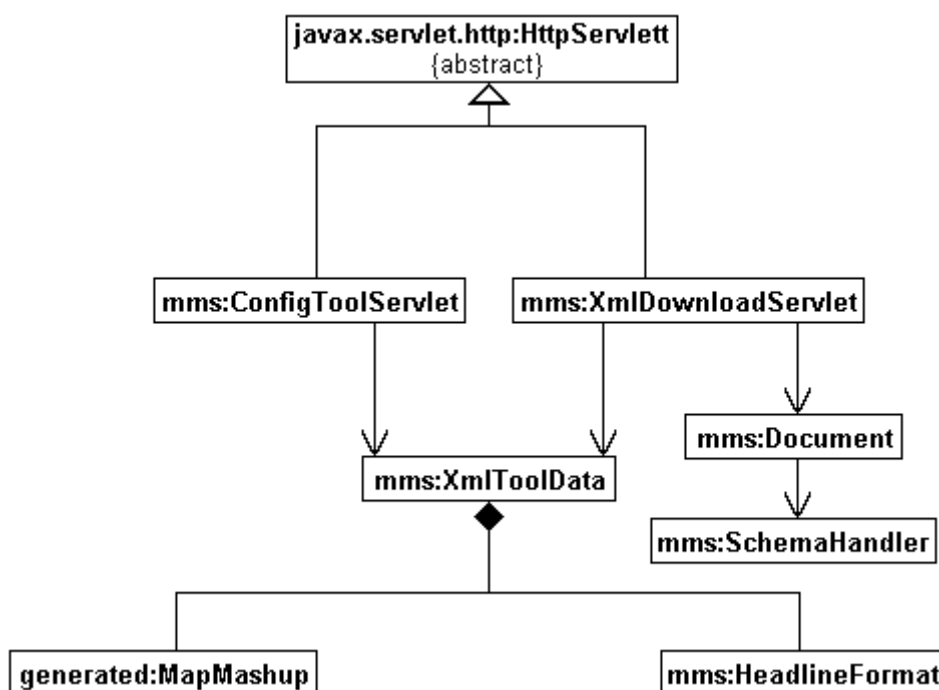


Abbildung 10: Klassendiagramm XML-Konfigurationsdatei-Tool

Die beiden Klassen `ConfigToolServlet` und `XmlDownloadServlet` erweitern die abstrakte Klasse `HttpServlet`. Das `XmlDownloadServlet`-Servlet wird aktiv, wenn auf die Ressource „/configtool/mmsconfig.xml“ zugegriffen wird. Es nutzt ein `Document`-Objekt und ein `XmlToolData`-Objekt, um eine XML-Konfigurationsdatei zu erstellen und als Antwort zu versenden. Ein `XmlToolData`-Objekt besteht aus einem `MapMashup`-Objekt und einem `HeadlineFormat`-Objekt. Das `MapMashup`-Objekt speichert alle notwendigen Parameter zur Erstellung einer Map-Mashup-Seite in einer Objektstruktur, die der Marshaller nutzen kann, um daraus ein XML-Dokument zu erstellen. Die Klasse `HeadlineFormat` dient zur Speicherung eines einheitlichen Überschriftenformats für die Bereiche

Übersichtskarte, Adresskarte, Dateneingabebereich und Datenausgabebereich. Obwohl es möglich ist, für jeden dieser Bereiche ein eigenes Format für die Überschrift anzugeben, wurde darauf im XML-Konfigurationstool verzichtet, um einerseits die Anzahl der notwendigen Eingaben zu verringern und andererseits es einfacher zu machen ein einheitliches Design anzufertigen.

4.2.7 Usability des XML-Konfigurationstools

Usability bedeutet auf deutsch Gebrauchstauglichkeit und wurde in der Norm DIN EN ISO 9241-11 folgendermaßen definiert:

"Das Ausmaß in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen"

Aus dieser Definition kann man die wesentlichen Faktoren der Gebrauchstauglichkeit entnehmen: Effektivität, Effizienz und Zufriedenheit.

- Effektivität: Der Grad der Zielerreichung hinsichtlich Vollständigkeit und Genauigkeit
- Effizienz: Das Verhältnis des Aufwandes zu seinem Ergebnis, den ein Nutzer betreiben muss
- Zufriedenheit: Ein Nutzer besitzt nach Erreichen seines Zieles eine positive Einstellung gegenüber dem Produkt

Um die in Kapitel 2.4 Punkt 2 definierte nicht-funktionale Anforderung zu erreichen, wurde sich bei dem Design der Konfigurationstool-Webseiten auf die in [Dah06] vorgestellten 10 Usability Heuristiken orientiert. Sie stammen von Jakob Nielsen, der eine führende Rolle in dem Bereich Usability spielt. Sie lauten im einzelnen:

1. Einfache und natürliche Dialoge:
Der Dialog soll nur die benötigten Informationen darstellen und dem Lösungsablauf des Benutzers angepasst sein.
2. Ausdrucksweisen des Anwenders benutzen:
Es soll die Fachsprache der Anwender benutzt werden.
3. Minimale mentale Belastung des Benutzers:
Der Anwender soll sich nicht mehr wie 7 ± 2 Informationseinheiten merken müssen.

4. Konsistenz einhalten:
Elemente werden immer gleich dargestellt und bezeichnet.
5. Rückmeldungen geben:
Auf Benutzeraktionen erfolgt eine Rückmeldung.
6. Klare Auswege zeigen:
Es muss eine Möglichkeit geben, zu einem bekannten Startpunkt zurückzukehren.
7. Abkürzungen anbieten:
Eingaben werden z.B. durch bestimmte Tastenkombinationen für erfahrene Benutzer beschleunigt.
8. Gute Fehlermeldungen ausgeben:
Die Fehlermeldung sollte im Idealfall Hinweise zur Beseitigung des Fehlers geben.
9. Fehlervermeidung betreiben:
Dafür sorgen, dass der Anwender bei der Eingabe erst gar keine Fehler macht.
10. Hilfe und Dokumentation anbieten:
Hilfe für alle Bereiche einer Anwendung anbieten und die Dokumentation immer aktuell halten.

4.2.8 Fazit

Das Design des XML-Konfigurationsdatei-Tools nutzt das `XMLToolData`-Servlet als zentrales Servlet, das sämtliche Anfragen entgegennimmt und an die erforderlichen JSP's weiterleitet. Dieses an das Model-View-Controller Pattern 2 angelegte Design erlaubt die Trennung von Präsentation und Logik im XML-Konfigurationsdatei-Tool. Durch die Nutzung von XML und XML-Schema ist es möglich die erstellten XML-Konfigurationsdateien noch weiter von Hand anzupassen. Dies ist möglich, da XML-Dokumente mit einfachen Texteditoren bearbeitet werden können und eine Überprüfung der XML-Konfigurationsdatei automatisch vom System vorgenommen wird. Es ist aber auch möglich eine XML-Konfigurationsdatei über einen Online-Validator wie z.B. [Val] zu überprüfen und so eine Aussage über die Validität des Dokumentes zu erhalten.

4.3 XML Schemata

XML-Schemata werden gebraucht, um XML-Dokumente auf korrekte Syntax zu überprüfen. Diese Aufgabe kann zwar auch von DTD's übernommen werden, aber Schemata bietet bessere Möglichkeiten Eingaben zu beschränken bzw. zu überprüfen, da sie eingebaute Datentypen wie Integer, Boolean oder Decimal besitzen. Ein weiterer Punkt der für Schematas spricht ist die Möglichkeit eigene Datentypen durch Erweiterung oder Einschränkung von bestehenden Typen zu erstellen.

Für die Erstellung von XML-Schemata gibt es verschiedene Designstrategien, die sich darin unterscheiden, ob Elemente und Elementtypen global oder lokal definiert werden.

4.3.1 Russian Doll

Bei diesem Ansatz werden Elemente und Elementtypen lokal definiert. Das Wurzel-Element ist das einzige globale Element. Dies verhindert eine Nutzung aller anderen Elemente und aller Elementtypen in anderen Schemata.

4.3.2 Salami Slice

Hier werden alle Elemente global Definiert, die Elementtypen jedoch alle lokal. Dies ermöglicht die Nutzung der verschiedenen Elemente in anderen Schemata, die Elementtypen sind jedoch nicht nutzbar.

4.3.3 Venetian Blind

Beim Vorgehen nach der Venetian Blind Designstrategie wird nur ein globales Element erstellt. Alle anderen Elemente werden lokal Definiert. Im Unterschied zum Russian Doll Ansatz werden jedoch alle Elementtypen global definiert und somit wiederverwendbar.

4.3.4 Garden of Eden

Bei diesem Designmodell werden alle Elemente und Elementtypen global definiert und wiederverwendbar. Dies führt jedoch dazu, das Schemata, die nach diesem Model erstellt werden relativ groß und unübersichtlich sind.

4.3.5 Fazit

Die vorangestellten Designmodelle werden in der Praxis nur bei sehr kleinen Schemata in Reinform angewandt. Wesentlich häufiger sind unterschiedliche Mischformen. Welche Designmodelle man dann benutzt hängt zum einen davon ab, welche Daten mit dem Schema strukturiert werden sollen und ob das Schema erweitert oder als Grundlage für ein anderes Schema dienen soll.

4.4 Sicherheit

Eine wichtige nicht-funktionale Anforderung an die Webapplikation ist die Sicherheit der auf der Map-Mashup-Seite eingegebenen Nutzerdaten. Es wurden daher verschiedene Maßnahmen ergriffen, um diese Daten vor dem Zugriff aus dem Internet zu schützen.

4.4.1 HTTPS

Eine wichtige Maßnahme ist der erzwungene Transport der Daten über eine HTTPS Verbindung. Dafür wurde ein auf die URL der Webanwendung ausgestelltes Server-Zertifikat benötigt. Es ermöglicht die verschlüsselte und authentifizierte Datenübertragung zwischen der Webanwendung und dem Client, um so ein mitlesen der übertragenden Daten von dritter Seite zu verhindern.

4.4.2 Parameter überprüfen

Für die Erstellung der Kundendaten ist es erforderlich, dass in der Anfrage des Clients 3 Parameter für die Erstellung der Kundendaten angegeben sind. Der erste Parameter gibt die Benutzerkennung an, in dessen Unterverzeichnis die Datei erstellt wird. Der zweite und dritte Parameter geben eine ID und einen Dateinamen (ohne Endung) an. Diese zwei Parameter müssen dabei mindestens 12 Zeichen lang sein und dürfen die Buchstaben von a bis z (groß und klein) und die Ziffern von 0 bis 9 enthalten. Es wird überprüft, dass sich die beiden Parameter unterscheiden, um so zu vermeiden, dass über die ID auf die Datei zugegriffen werden kann. Alleine für den Dateinamen entsteht so die beachtliche Zahl von $(26 \cdot 2 + 10)^{12} = 3.226.266.762.397.899.821.056$ Möglichkeiten.

4.4.3 Löschen der Kundendaten

Kundendaten werden nur temporär gespeichert. Es kann über einen Systemparameter eingestellt werden, wie lange die Daten bereitgestellt werden. Nach Ablauf der eingestellten Zeitspanne werden die Daten automatisch gelöscht. Das automatische Löschen der Kundendaten verringert dabei das Risiko, dass eine willkürliche Datenanforderung tatsächlich auf vorhandene Kundendaten zugreifen kann.

4.4.4 Fazit

Das hier vorgestellte Sicherheitskonzept schützt bei korrekter Benutzung die Kundendaten vor unberechtigten Zugriffen aus dem Internet. Die große Anzahl von Kombinationsmöglichkeiten für die ID und den Dateinamen machen es potenziellen Datendieben schwer vor der Löschung der Kundendaten darauf zugreifen zu können. Es gibt allerdings noch eine Schwachstelle in dem System. Wenn die übergebenen Schlüssel für die ID und den Dateinamen zwar unterschiedlich sind, sich aber bei verschiedenen Aufrufen wiederholen, so wird dies nicht vom System beanstandet. Dies könnte man verhindern, indem diese Schlüsselpaare gesammelt und erst nach einer bestimmten Anzahl von verschiedenen Schlüsseln wieder freigegeben werden. In dieser ersten Version der Webanwendung werden Schlüsselwiederholungen jedoch nicht kontrolliert und es ist die Aufgabe der Webseitenbetreiber für zufällige Schlüssel zu sorgen.

4.5 Fehlerbehandlung und Logging

Ausnahmezustände des Programms werden mit Exceptions abgefangen. Wo es möglich ist wird die Ausnahme lokal gefangen und die Ursache der Ausnahme behoben. Gelingt die Behebung nicht, so wird eine neue Ausnahme geworfen, die eine genauere Beschreibung der Fehlerursache und die zugrundeliegende Ausnahme enthält.

4.5.1 Exceptions

Exceptions sind eine Unterklasse von `Throwables`. Wenn man in einem Programm selber Ausnahmen werfen möchte, so sollten es immer von `Exception` abgeleitete Klassen sein. Es gibt in der Programmiersprache Java schon einige von `Exception`

abgeleitete Klassen, die man nutzen kann, es besteht aber auch die Möglichkeit eigene, von `Exception` abgeleitete Klassen zu erstellen. Auf der Fehlerseite kann dann überprüft werden, ob es sich bei dem übergebenen Fehlerobjekt um eine Instanz der selber erstellten Exceptionklasse handelt. Ist das der Fall, kann man sicher sein, dass die Exception von eigenem Code geworfen wurde und sie eventuell gesondert behandeln.

4.5.1.1 *XmlFileException*

Eine `XmlFileException` wird geworfen, wenn im Zusammenhang mit der Xml-Konfigurationsdatei ein Fehler auftritt. Diese Art von Fehler ist auf einen bestimmten Benutzer begrenzt und hat keine Auswirkungen auf die Funktion der Webanwendung als Ganzes.

4.5.1.2 *MmsException*

Die `MmsException` wird geworfen, falls ein Fehler auftritt, der die gesamte Webanwendung betrifft. Das können z.B. fehlende oder falsche Parameter in der `web.xml` Konfigurationsdatei oder in der `user.xml` Datei sein.

4.5.2 Error Page

Eine Fehlerseite ist ein möglicher Schlusspunkt für eine Ausnahme. Ausnahmen, die nicht lokal behoben werden können, sollten auf eine Fehlerseite weitergeleitet werden. Es gibt die Möglichkeit in jeder JSP eine eigene Fehlerseite anzugeben. Das geschieht über die `page`-Direktive „`errorPage`“, mit der die URL der zu benutzenden Fehlerseite angegeben werden kann. Eine andere Möglichkeit ist durch das Setzen des Elementes „`Error-Page`“ in der Konfigurationsdatei `web.xml` gegeben. Hier kann man Ausnahmen zentral bestimmten Fehlerseiten zuordnen.

Für diese Webapplikation wird die Fehlerseite zentral in der Konfigurationsdatei `web.xml` bestimmt. Das hat den Vorteil, dass eventuelle Änderungen später nur an einer Stelle vorgenommen werden müssen und dieser Mechanismus ohne zusätzlichen Aufwand für JSP und Servlets benutzt werden kann. Ein weiterer Punkt ist, dass so sichergestellt werden kann, dass sämtliche Ausnahmen an eine eigene Fehlerseite gesendet werden und es so nicht vergessen werden kann einer JSP einer Fehlerseite zuzuordnen.

4.5.3 Logging

Logging ist dafür zuständig, Informationen über das System auszugeben und aufzuzeichnen. Es lässt sich dabei festlegen, welche Art von Informationen aufgezeichnet und ausgegeben werden sollen. Die Informationen werden daher in verschiedene Kategorien nach Wichtigkeit aufgeteilt und es kann angegeben werden, ab welcher Wichtigkeit Informationen ausgegeben werden sollen.

4.5.3.1 Konsolenlogger

Der Konsolenlogger ist nur eine Unterstützung für den Dateilogger. Er gibt die Meldungen direkt auf den Bildschirm aus und eignet sich so für die direkte Kontrolle der Webanwendung. Da eine Webanwendung aber normalerweise unbeaufsichtigt läuft, kann es leicht vorkommen, dass Meldungen durch das Scrollen des Bildschirminhaltes wieder verschwinden bevor sie gelesen werden können.

4.5.3.2 Dateilogger

Dateilogger halten die Systemmeldungen je nach Einstellung in einer oder mehreren Dateien fest. Dadurch ist es möglich auch Meldungen noch zu lesen, die vom Bildschirm schon längst wieder verschwunden sind. Dateilogger können so eingestellt werden, dass sie bei Erreichen einer bestimmten Dateigröße automatisch eine neue Datei mit leicht veränderten Namen erstellen. Die Namensänderung wird meist durch einfaches Hochzählen einer Ziffer realisiert.

4.5.4 Fazit

Mit den erstellten Exceptionklassen ist es möglich auf der Fehlerseite gezielt auf die selber geworfenen Ausnahmen zu reagieren. Die zentrale Verwaltung der Fehlerseite sorgt dafür dass alle Fehler die auftreten können zur Fehlerseite weitergereicht werden und das Logging hilft Fehler im laufenden Betrieb aufzuspüren und zu Dokumentieren.

5 Auswahl der Systemkomponenten

In den folgenden Unterpunkten wurde sich für eine bestimmte Implementation einer Komponente entschieden. Dabei wurde Wert auf Zuverlässigkeit, Verbreitung, Weiterentwicklung und eine Open-Source-Lizenz gelegt.

5.1 Webserver / Applikationsserver

Für die Erstellung der dynamischen Webseiten wird ein Applikationsserver benötigt. Die Wahl fiel hier auf den Tomcat Applikationsserver, da er die Standardimplementation der Servlet und JSP-Spezifikation ist und so ein späterer Wechsel auf einen anderen Applikationsserver einfach möglich ist. Ein weiterer Grund der für Tomcat spricht ist, dass man ihn auch als Webserver betreiben kann. Tomcat ist beim Verarbeiten von statischen Inhalten etwas langsamer als z.B. der Apache Webserver. Da bei dieser Webapplikation aber nur ein geringer Anteil der zu übertragenen Inhalte statisch ist, stellt die etwas geringere Bereitstellungszeit für statische Inhalte kein Problem dar. Tomcat ist ein sehr weit verbreiteter Applikationsserver der aufgrund seiner Zuverlässigkeit auch in kommerziellen Projekten eingesetzt wird. Tomcat ist unter einer Open-Source-Lizenz erhältlich.

5.2 Java Binding

Für die Aufgabe des Java Binding wurden an das Framework die Anforderungen gestellt, das es die Validierung von XML-Dokumenten unterstützt und das es die Benutzung von XML-Schemata unterstützt. Ein weiterer Punkt war, dass es unter einer Open-Source-Lizenz stehen sollte. Auf diese Punkte wurden 4 Frameworks untersucht:

	Castor	Zeus	JBind	JAXB
Java > XML	ja	ja	ja	ja
XML > Java	ja	ja	ja	ja
Validation	ja	ja	ja	ja
Schemata	ja	ja	ja	ja
Open Source	ja	ja	ja	ja

Tabelle 9: Vergleich Java Binding Frameworks

Da alle Frameworks die Anforderungen erfüllen, habe ich mich für das JAXB Framework entschieden. Für dieses Framework gibt es eine Referenzimplementierung von Sun, die den Status Produktionsqualität besitzt.

5.3 Map API

Auf Grundlage der Studienarbeit „Mashups - eine Übersicht“ wurde sich für die Google-Maps-API entschieden. Sie ist in der Lage Kartenmaterial in genügender Genauigkeit bereitzustellen und bietet Methoden zur Darstellung von Karten, Markern, Linien und Polygonen. Außerdem enthält sie die Möglichkeit aus Adressen die geografische Positionen zu ermitteln. Die Google-Maps-API steht nicht unter einer Open-Source-Lizenz. Es gibt jedoch zum jetzigen Zeitpunkt keine Open-Source-Map-API die Kartenmaterial und Geokodierung in der benötigten Genauigkeit anbietet.

5.4 Logging

Für das Logging wurden zwei API's in betracht gezogen. Zum einen Log4J und zum anderen die Java Logging API. Beide API's bieten das Loggen in Dateien und auf der Konsole an und ähneln sich in ihrem Funktionsumfang sehr. Für das einfache Logging, das in dieser Webapplikation benötigt wird sind daher beide API's geeignet. Es wurde sich für die Java Logging API entschieden, da sie bereits Bestandteil des JDK 5 ist und so ohne weitere Maßnahmen sofort einsetzbar ist.

5.5 Fazit

Die Ausgewählten Komponenten sind entweder Referenzimplementationen, Bestandteil des JDK oder werden in vielen verschiedenen Projekten erfolgreich eingesetzt. Daher ist es möglich für diese Komponenten an verschiedenen Stellen Hilfe zu finden. Die Zuverlässigkeit dieser von vielen Projekten genutzten Komponenten hat ein hohes Maß erreicht und da sie immer noch weiterentwickelt werden, ist zusätzlich die Möglichkeit gegeben später auf aktualisierte Versionen der Komponenten zu wechseln.

6 Erstellen der XML-Schemata

Für die Webanwendung werden zwei Schemata benötigt. Ein Schema für die Benutzerdaten und ein Schema für die Map-Mashup-Konfigurationsdaten. Beide Schemata sollen für sich alleine stehen und werden nicht von anderen Schemata referenziert. Die Entscheidung, welches Designmodell angewendet werden soll, kann sich also ausschließlich auf die zu strukturierenden Daten beziehen.

6.1 Schema user.xsd

Dieses Schema enthält nur sehr wenige Elemente, die alle von einem unterschiedlichen Typ sind. Da hier weder Elemente noch Elementtypen wiederverwendet werden, wurde für dieses Schema das Designmodell Russian Doll gewählt.

Für jeden Benutzer gibt es ein `user`-Element. Das `user`-Element besteht aus den Elementen `name`, `password` und `xmlUrl`. `name` und `xmlUrl` sind notwendig, um zu einer gegebenen Benutzerkennung später die URL der XML-Konfigurationsdatei zu erhalten.

Mit dem Element `password` kann der Administrator die Authentizität von Emails überprüfen, die er von Benutzern erhält.

Das Schema muss das XML-Dokument daraufhin untersuchen, ob die Benutzerkennungen eindeutig sind. Dies wurde durch die Benutzung eines `Unique`-Elementes erreicht.

6.2 Schema mapmashup.xsd

Bei diesem Schema werden Elementtypen erweitert und auch wiederverwendet. Für dieses Schema wurde daher das Designmodell Venetian Blind als Grundlage gewählt. Da aber nicht alle Elementtypen Wiederverwendung finden, wurden einige auch lokal definiert. Es ergibt sich somit eine Vermischung der beiden Designmodelle Venetian Blind und Russian Doll für dieses Schema.

Die Map-Mashup-Schema-Datei definiert neben dem Wurzelement „`mapMashup`“ auch Elemente für die Übersichtskarte, die Adresskarte, den Eingabebereich, den Ausgabebereich, den Schaltflächenbereich und den allgemeinen Bereich. Doppelte Elementwerte in `orderNumber` werden durch ein `Unique`-Element verhindert.

6.3 Testen der Schemata

Die erstellten Schemata wurden mit dem Online-Validator von Validome auf Validität überprüft. Um festzustellen, ob die erstellten Schemata auch semantisch korrekt sind, wurden XML-Dokumente erstellt, die jeweils gegenüber einem Schema valide waren. An diesen XML-Dokumente wurden anschließend Veränderungen vorgenommen und auf diese Weise getestet, ob z.B. doppelte Elementwerte von orderNumber wirklich festgestellt werden oder ob angegebene Pattern den gewünschten Wertebereich abdecken.

6.4 Fazit

Die in den Schemata enthaltenen Datenstrukturen sind geeignet, um die benötigten Benutzerdaten und Map-Mashupdaten aufzunehmen. Die Schemata sind syntaktisch und semantisch korrekt und können von der Webapplikation verwendet werden.

7 Einrichten der Entwicklungsumgebung

Bevor die Entwicklungsumgebung genutzt werden kann, müssen alle benötigten Komponenten installiert und eingerichtet werden.

7.1 JDK

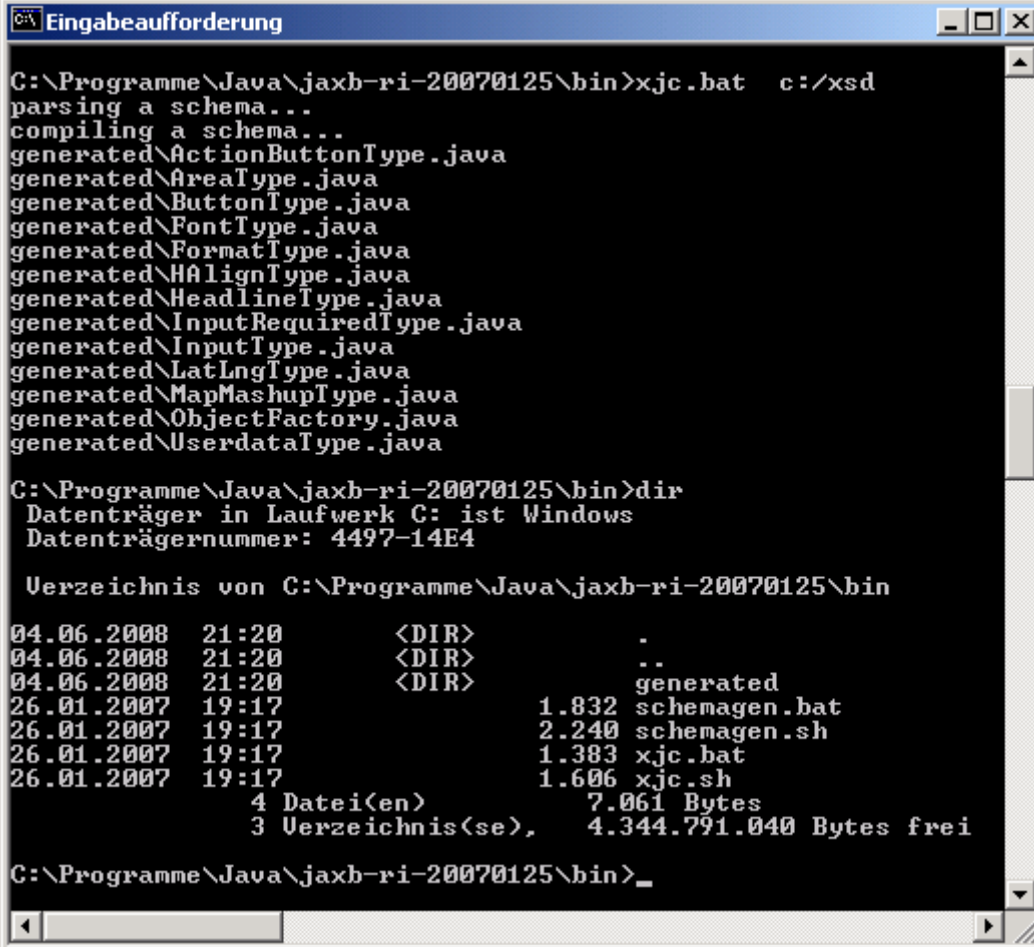
Die verwendete JDK 5 kann auf der Seite von Sun Microsystems Inc. für verschiedene Betriebssysteme kostenlos heruntergeladen werden. Nach dem Herunterladen der Installationsdatei muss diese noch ausgeführt werden und den Anweisungen darin gefolgt werden.

Anschließend sollte in Windows-Systemen noch die Path-Variable mit dem `bin`-Verzeichnis der Java-Installation erweitert werden, um so die Benutzung verschiedener Java-Komponenten von der Konsole aus zu vereinfachen.

7.2 JAXB

Die Referenzimplementation von JAXB 2.0 kann unter [JAXB] heruntergeladen werden. Die heruntergeladene `.jar`-Datei wird von der Konsole aus mit dem Befehl `java -jar JAXB2_20070122.jar` gestartet. Nachdem man den Lizenzbestimmungen zugestimmt hat wird ein Ordner in dem Verzeichnis der `.jar`-Datei erstellt. In ihm befinden sich die vier Ordner `bin`, `lib`, `docs` und `samples`. Der Ordner `docs` enthält die Dokumentation über JAXB und der Ordner `samples` Beispiele. Im Ordner `lib` befinden sich unter anderem die Dateien `jaxb-api.jar`, `jaxb-impl.jar`, `jsr173_1.0_api.jar` und `activation.jar`. Sie werden benötigt, um in einer Anwendung JAXB 2.0 nutzen zu können. In dem Ordner `/bin` befindet sich der Binding-Kompiler XJC, mit dem aus einem XML-Schema Java-Klassen erstellt werden. Es wird im einfachsten Fall nur mit dem Verzeichnis aufgerufen, in dem die zu verarbeitenden Schemadateien enthalten sind. Die erstellten Klassen werden dann in dem Paket „generated“ erstellt. Für diese Webapplikation wurden die beiden erstellten Schemata `user.xsd` und `mapmashup.xsd` in das Verzeichnis `c:/xsd` kopiert und XJC mit dem Parameter `c:/xsd` aufgerufen. In der nachfolgenden Abbildung kann man erkennen, wie XJC die erstellten Klassen dem Paket „generated“ hinzugefügt hat. Die erstellten

Klassen wurden in dem von XJC automatisch erstellten Ordner „generated“ abgelegt.



```
C:\Programme\Java\jaxb-ri-20070125\bin>xjc.bat c:/xsd
parsing a schema...
compiling a schema...
generated\ActionButtonType.java
generated\AreaType.java
generated\ButtonType.java
generated\FontType.java
generated\FormatType.java
generated\HAlignType.java
generated\HeadlineType.java
generated\InputRequiredType.java
generated\InputType.java
generated\LatLngType.java
generated\MapMashupType.java
generated\ObjectFactory.java
generated\UserdataType.java

C:\Programme\Java\jaxb-ri-20070125\bin>dir
Datenträger in Laufwerk C: ist Windows
Datenträgernummer: 4497-14E4

Verzeichnis von C:\Programme\Java\jaxb-ri-20070125\bin

04.06.2008  21:20    <DIR>          .
04.06.2008  21:20    <DIR>          ..
04.06.2008  21:20    <DIR>          generated
26.01.2007  19:17           1.832 schemagen.bat
26.01.2007  19:17           2.240 schemagen.sh
26.01.2007  19:17           1.383 xjc.bat
26.01.2007  19:17           1.606 xjc.sh
               4 Datei(en)           7.061 Bytes
               3 Verzeichnis(se), 4.344.791.040 Bytes frei

C:\Programme\Java\jaxb-ri-20070125\bin>_
```

Abbildung 11: Benutzung des XJC-Schema Compilers

7.3 Name-Service

Um die Webanwendung einer festen Webadresse zuordnen zu können, wurde auf einen Dienst zugegriffen, der es erlaubt wechselnden IP-Adressen einem festen Domain-Namen zuzuordnen.

Dieser einfache Dienst ist bei [Dyn] kostenlos, ein eingerichteter Zugang muss aber mindestens einmal alle 35 Tage einmal genutzt werden (spricht: die IP-Adresse muss erneuert werden) oder der Zugang wird gelöscht. Um die Webanwendung von anderen Rechnern im Internet zu testen, war dieser Dienst aber ausreichend.

7.4 Tomcat

Tomcat kann in der Version 5.5 unter [Tom] heruntergeladen werden. Es gibt eine Zip-Downloadversion, die nur entpackt werden muss und dann sofort einsetzbar ist.

Unter Windows-Betriebssystemen kann Tomcat einfach mit der Batch-Datei `startup.bat` gestartet und mit `shutdown.bat` beendet werden. Diese beiden Dateien befinden sich im Verzeichnis `bin` des Tomcat-Installationsverzeichnis.

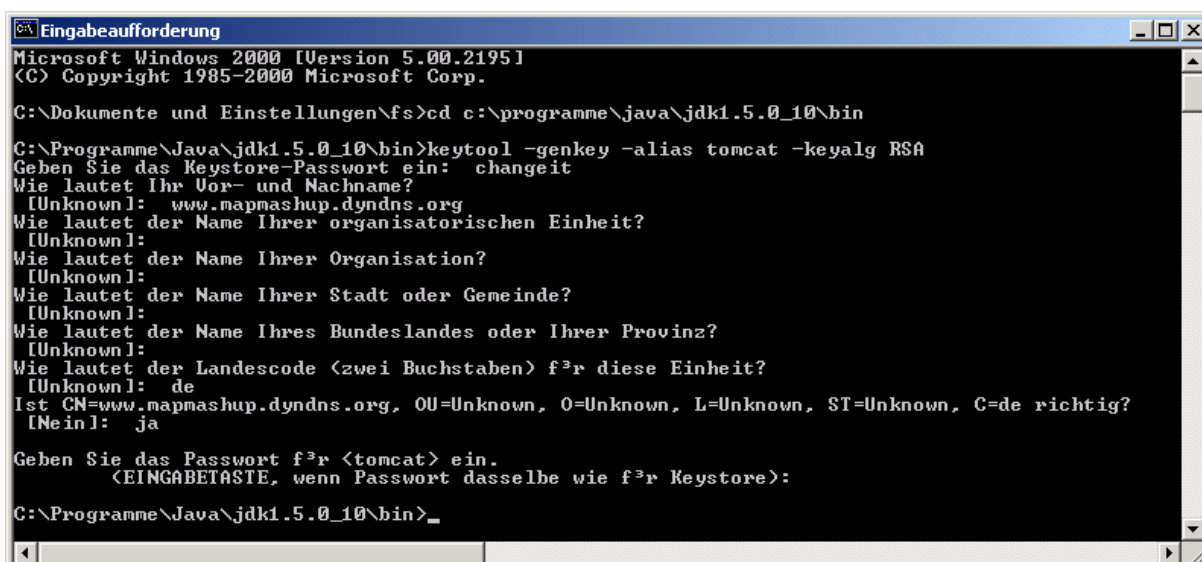
7.4.1 SSL Zertifikat erstellen

Um mit Tomcat HTTPS nutzen zu können, braucht man ein SSL Zertifikat. Diese werden von Zertifizierungsstellen ausgestellt und digital signiert. In Webbrowsern sind einige von den Browserherstellern als vertrauenswürdig eingestufte Zertifikate gespeichert. Jedes gültige Zertifikat, das der Webbrowser zu einer dieser vertrauenswürdig Root-Zertifikate zurückverfolgen kann wird von ihm ohne weiteres akzeptiert. Kann er es nicht zurückverfolgen, öffnet er ein Eingabefenster und fragt den Benutzer, ob er diesem Zertifikat vertrauen möchte. Diese Frage kann der Nutzer natürlich mit „nein“ beantworten und so ist es für jede Webanwendung besser ein SSL-Zertifikat zu besitzen, dass der Browser sofort als vertrauenswürdig einstuft.

Da ein solches SSL-Zertifikat nicht kostenlos zu erhalten war, wurde ein Testzertifikat erstellt. Dafür gibt es das Programm `keytool` im Verzeichnis „bin“ der Javainstallation. Das Programm wurde mit den folgenden Parametern aufgerufen:

- `genkey`
Dieser Parameter gibt an, dass ein Schlüsselpaar (öffentlich und privat) erstellt werden soll. Dieses Schlüsselpaar wird dem angegebenen Keystore (per default im Heimatverzeichnis des Benutzers) hinzugefügt. Existiert dort kein Keystore, so wird ein neuer erstellt.
- `alias tomcat`
Auf Keystore Elemente wird über eindeutige Aliase zugegriffen. Hier wird der Alias `tomcat` angegeben, über den später wieder auf die Schlüssel zugegriffen werden kann.
- `keyalg RSA`
Hier wird als Verschlüsselungsalgorithmus RSA festgelegt.

In der folgenden Abbildung sieht man die Ausgaben des Programms nachdem es mit `keytool -genkey -alias tomcat -keyalg RSA` aufgerufen wurde.



```
Eingabeaufforderung
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Dokumente und Einstellungen\fs>cd c:\programme\java\jdk1.5.0_10\bin

C:\Programme\Java\jdk1.5.0_10\bin>keytool -genkey -alias tomcat -keyalg RSA
Geben Sie das Keystore-Passwort ein: changeit
Wie lautet Ihr Vor- und Nachname?
[Unknown]: www.mapmashup.dyndns.org
Wie lautet der Name Ihrer organisatorischen Einheit?
[Unknown]:
Wie lautet der Name Ihrer Organisation?
[Unknown]:
Wie lautet der Name Ihrer Stadt oder Gemeinde?
[Unknown]:
Wie lautet der Name Ihres Bundeslandes oder Ihrer Provinz?
[Unknown]:
Wie lautet der Landescode (zwei Buchstaben) f³r diese Einheit?
[Unknown]: de
Ist CN=www.mapmashup.dyndns.org, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=de richtig?
[Nein]: ja

Geben Sie das Passwort f³r <tomcat> ein.
(EINGABETASTE, wenn Passwort dasselbe wie f³r Keystore):

C:\Programme\Java\jdk1.5.0_10\bin>_
```

Abbildung 12: Erstellung eines Keystores mit dem keytool-Programm

Wie man sieht wurde zuerst nach dem Passwort für den Keystore und zum Schluss nach dem Passwort für das Schlüsselpaar gefragt. Es ist wichtig, dass die Passwörter für den Keystore und das Schlüsselpaar identisch sind, da es in Tomcat nur möglich ist ein Passwort für beide einzustellen. „changeit“ ist das Standardpasswort bei Tomcat, das man in der `server.xml` im Factory-Element des Connector-Elements verändern kann. Als Besonderheit kann man erkennen, dass man auf die Frage nach dem Vor- und Nachnamen den Domainnamen angeben muss, für den das Zertifikat gültig sein soll. Hier wurde www.mapmashup.dyndns.org eingegeben.

Nach der Beendigung des Keystore-Programms befindet sich eine „.keystore“-Datei im Heimatverzeichnis des Benutzer. Dies ist auch der Ort, an dem Tomcat standardmäßig nach dem Keystore sucht.

7.4.2 Konfigurationsdateien

Tomcat wird mit xml-Dateien konfiguriert, die man mit normalen Texteditoren bearbeiten kann. Nachfolgend die beiden Konfigurationsdateien, in denen Änderungen vorgenommen wurden.

7.4.2.1 *server.xml*

Die `server.xml` ist die wichtigste Konfigurationsdatei in Tomcat. In ihr werden unter anderem die Ports definiert auf denen der Server erreichbar ist und hier kann man

auch einstellen, welche Webressourcen nur über sichere Verbindung erreichen werden können.

Standardmäßig ist nur der Port 8080 auf Tomcat erreichbar. Um das zu ändern wurden folgende Parameter des ersten Connector-Elements auf folgende Weise verändert.

- **port:**
Hier wurde der Port auf 80 gesetzt. Dies ist der Standard für HTTP.
- **redirectPort:**
Dieser Parameter ist wichtig im Zusammenhang mit HTTPS. Auf diesen Port werden Anfragen umgeleitet, die auf Webressourcen zugreifen möchten, die nur mittels einer gesicherten Verbindung erreicht werden dürfen. Er wurde auf Port 443 gesetzt, das ist der Standardport für HTTPS.

Für den Port 443 war nun ein weiteres Connector-Element notwendig. In der `server.xml` war jedoch schon ein weiteres Connector-Element vorgesehen, das auch schon für den Einsatz für HTTPS vorbereitet war. Im Connector-Element musste nur der Parameter für die Portnummer angepasst werden.

- **port:**
Der Port wurde auf 443 gesetzt, was sich auch mit dem `redirectPort` Parameter des ersten Connector-Elements deckte.

7.4.2.2 *web.xml*

Die `web.xml` ist die Konfigurationsdatei für die Webanwendung und wird auch Deployment-Deskriptor genannt. Jede Webanwendung hat ihre eigene `web.xml` Konfigurationsdatei und zusätzlich gibt es noch eine globale `web.xml` Konfigurationsdatei im gleichen Verzeichnis wie die `server.xml` Datei. Dabei ist zu beachten, dass die Konfigurationsdateien der einzelnen Webanwendungen die Einstellungen der globalen `web.xml` überschreiben.

Folgende Elemente wurden in der `web.xml` des `MapMashupService` hinzugefügt:

- **context-param:**
Sie stellen Parameter bereit, auf die man innerhalb von Tomcat-Komponenten wie z.B. Servlets zugreifen kann.

```
<context-param>
    <param-name> </param-name>
    <param-value> </param-value>
</context-param>
```

- **listener:**

Durch dieses Element werden Events der Tomcat-Komponenten an die im Kind-Element „`listener-class`“ angegebene Klasse gesendet.

```
<listener>
    <listener-class></listener-class>
</listener>
```

- **filter und filter-mapping:**

Diese beiden Elemente definieren einen Filter. Das Element „`filter-mapping`“ ist dafür zuständig festzustellen, ob auf eine Ressource zugegriffen werden soll, für die der Filter eingerichtet wurde. Stimmt sein URL-Muster mit dem Ressourcenziel im Request überein, dann wird der Request an die im „`filter`“-Element angegebene-Klasse weitergeleitet.

```
<filter>
    <filter-name></filter-name>
    <filter-class></filter-class>
</filter>
<filter-mapping>
    <filter-name></filter-name>
    <url-pattern></url-pattern>
</filter-mapping>
```

- **servlet und servlet-mapping:**

Das Element „`servlet-mapping`“ hat die gleiche Aufgabe für Servlets wie das Element „`filter-mapping`“ für Filter. Der einzige Unterschied ist nur, dass es den Request seinem eigentlichen Ziel, dem Servlet zuführt. Welches Servlet den Request erhält, steht im Element „`servlet-class`“ und ergibt sich wieder aus dem Zusammenspiel von dem Element „`servlet-name`“ die

es jeweils im Element „servlet“ und im Element „servlet-mapping“ gibt. Das Element „load-on-startup“ dient dazu festzulegen, wann das Servlet in den Arbeitsspeicher geladen werden soll. Lässt man dieses Element weg, so wird das Servlet erst dann geladen, wenn es gebraucht wird. Das ist zwar einerseits ressourcenschonend, andererseits verzögert es aber auch die Antwort bei der ersten Anforderung.

```
<servlet>
    <servlet-name></servlet-name>
    <servlet-class></servlet-class>
    <load-on-startup></load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name></servlet-name>
    <url-pattern></url-pattern>
</servlet-mapping>
```

- **error-page:**

Tritt eine Exception auf deren Typ im „exception-type“-Element angegeben ist, wird die im „location“-Element angegebene Ressource aufgerufen.

```
<error-page>
    <exception-type></exception-type>
    <location></location>
</error-page>
```

- **taglib im Element jsp-config**

Das Element „taglib“ wird benötigt, um in JSP's eigene Tags nutzen zu können. Es befindet sich innerhalb des „jsp-config“-Elements, mit dem Einstellungen der JSP-Komponente vorgenommen werden. Das „taglib“-Kindelement „taglib-uri“ gibt dabei einen Namen an, auf den in den JSP-Seiten verwiesen werden kann. Das Kindelement „taglib-location“ referenziert die Taglib-Konfigurationsdatei, in der unter anderem auch die verschiedenen Tags definiert werden.

```
<jsp-config>
```

```
<taglib>
  <taglib-uri></taglib-uri>
  <taglib-location></taglib-location>
</taglib>
</jsp-config>
```

- **security-constraint:**

Mit dem Element „`security-constraint`“ lässt sich festlegen, wer, wie auf eine Webressource zugreifen darf. Hier wird nur die Art des Zugriffs geregelt, indem für bestimmte Webressourcen eine bestimmte Sicherheit des Transports garantiert wird.

Hierfür gibt es drei Einstellungsmöglichkeiten die im Element „`transport-guarantee`“ festgelegt werden müssen:

- 1) None:

Die Übertragung der Daten erfolgt ungesichert

- 2) INTEGRAL:

Die übertragenden Daten können von Dritten zwar gesehen aber nicht verändert werden.

- 3) CONFIDENTIAL

Die Daten werden verschlüsselt übertragen und können so nicht von Dritten gelesen werden.

Das Element „`url-pattern`“ im Element „`web-resource-collection`“ legt fest, für welche Ressourcen die Einstellungen gelten sollen. Es ist dabei möglich mehrere „`url-pattern`“-Elemente zu benutzen.

```
<security-constraint>
  <display-name></display-name>
  <web-resource-collection>
    <web-resource-name></web-resource-name>
    <url-pattern></url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee></transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

7.5 Eclipse

Eclipse ist eine integrierte Entwicklungsumgebung die durch Plugins erweitert werden kann. Die Installationsversion „Eclipse IDE for Java EE Developers“ bietet neben der Unterstützung für die Programmiersprache Java auch alle benötigten Plugins, um Webanwendungen zu entwickeln. Sie kann von der Eclipse-Downloadseite [Eclip] bezogen werden. Für Windows-Betriebssysteme wird die Zip-Datei einfach in das gewünschte Verzeichnis entpackt und die enthaltende Datei `eclipse.exe` gestartet. Beim ersten Start von Eclipse wird der Workspace angelegt, in dem später die Projekte abgespeichert werden.

Für die Erstellung der Webanwendung wurde ein „Dynamic Web Project“ erstellt und ihm der Name `MapMashupService` zugewiesen. Um JAXB in der Anwendung nutzen zu können wurden die vier Dateien `activation.jar`, `jaxb-api.jar`, `jaxb-impl.jar` und `jsr173_1.0_api.jar` in das Verzeichnis `WEB-INF/lib` kopiert. Dem Dynamic Web Project wurde in der Konfigurierungsphase der eingerichtete Tomcat als Server zugewiesen.

7.6 Google-Maps API

Um die Google-Maps API nutzen zu können ist ein API Key notwendig. Diesen kostenlosen Key kann man unter [GKey] beantragen. Voraussetzung ist allerdings ein ebenfalls kostenloses Google-Benutzerkonto mit dem man eingeloggt sein muss. Für die Entwicklung der Webanwendung wurden zwei Keys beantragt. Der erste „`http://localhost`“ für erste Tests unter Verwendung des localhost und der zweite für die Verwendung mit der Webadresse „`http://www.mapmashup.dyndns.org`“. Es können beliebig viele Keys beantragt werden und auch die Nutzung ist bisher kostenlos.

Eingebunden wird das für die Google-Maps API notwendige JavaScript in ein HTML-Dokument über das Tag „`<script>`“. In diesem Tag wird die Quelle des einzubindenden JavaScript-Dokuments angegeben und dieser Link wird durch den Parameter „`key`“ erweitert. Bei Anfragen an die Google-Maps API wird der Key mit übertragen und ermöglicht es Google zu überprüfen, ob die Anforderung von einer berechtigten Webadresse stammt.

7.7 Fazit

Mit den hier beschriebenen Maßnahmen ist die Entwicklungsumgebung zur Erstellung der Webapplikation eingerichtet. Es können dynamische Webapplikationen entwickelt werden, die gesicherte Verbindungen anbieten. Die Webseiten können eine Map-API nutzen und so Map-Mashup-Funktionalität integrieren.

8 Implementierung

Die Implementierung der Webanwendung teilt sich in die zwei Bereiche Map-Mashup-Service und XML-Konfigurationsdatei-Tool auf. Die Exceptions und der Logger werden in der gesamten Webanwendung benutzt und werden daher hier gesondert aufgeführt.

8.1 Map-Mashup-Service

Der Map-Mashup-Service ist der wichtigste Teil der Webanwendung. Hier werden Anfragen zu Map-Mashup-Seiten entgegengenommen, dynamisch die gewünschten Seiten erstellt und auf die Anfragen der dynamisch erstellten Seiten reagiert.

8.1.1 Listener InitMms

Der Listener `InitMms` wird beim Start der Webapplikation erstellt und die Methode `contextInitialized(ServletContextEvent servletContextEvent)` aufgerufen. In dieser Methode werden die für die Webapplikation benötigten Initialisierungen durchgeführt. Die benötigten Parameterwerte bekommt der Listener dabei über das `ServletContext`-Object aus der `web.xml`-Konfigurationsdatei.

8.1.1.1 Initialisierung User

Über die statische Methode `initialize(String schemaLocation, String xmlFile)` werden der Klasse `User` das XML-Schema und die zugehörige XML-Datei mit den Benutzerdaten zugewiesen. Durch sie kann festgestellt werden, ob eine bestimmte Benutzerkennung existiert und unter welcher URL die Konfigurationsdatei für die Map-Mashup-Seite gefunden werden kann.

8.1.1.2 Initialisierung Document

Mit der statischen Methode `initialize(String schemaLocation)` wird in der Klasse `Document` angegeben, wo das XML-Schema Dokument zu finden ist. Der angegebene String repräsentiert dabei entweder eine URL oder einen vollständigen Dateinamen. Die Instanzvariable `Document.schemaLocation` speichert den übergebenen Wert und stellt ihn allen Instanzen zur Verfügung.

8.1.1.3 Initialisierung FileDelete

Bei der Initialisierung der Klasse `FileDelete` über die statische Methode `initialize(String downloadDir, String fileLifetime, String fileCheckIntervall)` wird eine Singleton-Instanz der Klasse erstellt und in einem eigenen Thread gestartet. In dem in `downloadDir` angegebenen Verzeichnis werden Unterverzeichnisse gesucht und in diesen Dateien gelöscht, die älter als die in `fileLifetime` angegebene Zeit sind. Die Überprüfung des Verzeichnisses findet in einem einstellbaren Intervall statt. Ein Beenden des Threads ist über die statische Methode `requestStop()` möglich.

8.1.2 Filter MmsFiler

Anfragen an die Ressource „/start“ werden zuerst an den Filter `MmsFilter` geleitet. In der Methode `doFilter` werden die Parameter der Anfrage ausgewertet. Ist in der Anfrage kein Parameter „user“ enthalten, dann wird die Anfrage an die Übersichtsseite weitergeleitet. Bei sonstigen Fehlern in den Parametern „user“, „id“ und „fileid“ werden Ausnahmen geworfen. Kann bei den Parametern kein Fehler festgestellt werden, so wird die Anfrage in der Filterkette weitergereicht.

8.1.3 Filter DownloadFilter

Anfragen an die Ressource „/download“ werden zum Filter „DownloadFilter“ geleitet. In der Methode `doFilter` wird aus der angeforderten URI der Name der gewünschten Datei extrahiert und anschließend im Downloadbereich des lokalen Filesystems nach dieser Datei gesucht. Wird die Datei gefunden, so wird sie gelesen und in den Ausgangsstrom des Response-Objekts geschrieben. Existiert die Datei nicht, so wird eine Fehlerseite mit dem Code 404 – not Found – zurückgegeben.

8.1.4 Servlet StartServlet

Anfragen, die den `MmsFilter`-Filter passiert haben kommen zum `StartServlet`-Servlet. HTTP-Post- und HTTP-Get-Anfragen werden in der Methode `doPost`

bearbeitet. Mit Hilfe der übergebenen Parameter wird ein `Document`-Objekt erstellt, das alle notwendigen Komponenten zur Erstellung der gewünschten Map-Mashup-Seite enthält. Dies geschieht über den Konstruktor des `Document`-Objektes, der als Parameter die URL der XML-Konfigurationsdatei erhält.

Um die XML-Konfigurationsdatei einzulesen, wird eine Instanz der Klasse `Unmarshaller` aus dem JAXB-Framework benötigt. Da die Klasse `Unmarshaller` aber nicht threadsicher ist, wird für jede Anfrage eine neue Instanz erstellt. Dafür wird die Klasse `SchemaHandler` benutzt, die eine Instanz der Klasse `JAXBContext` besitzt. Das Erstellen dieser Instanz ist relativ aufwändig und sie wird bei jeder Map-Mashup-Seitenanfrage benötigt. Die Klasse `JAXBContext` ist threadsicher und daher ist es kein Problem, wenn in der Klasse `SchemaHandler` eine `JAXBContext`-Instanz für alle Anfragen benutzt wird. Über die statische Methode `getJAXBContext` kann in der `SchemaHandler`-Klasse auf sie zugegriffen werden. Der `Unmarshaller` erzeugt aus der XML-Konfigurationsdatei eine Java-Objekthierarchie, die von verschiedenen Komponenten des `Document`-Objekts genutzt werden, um die dynamischen Teile der Map-Mashup-Seite zu generieren. Die festen Stylesheet Angaben und JavaScript-Funktionen werden dabei durch entsprechende Tags in das HTML-Dokument eingebunden. Das `StartServlet`-Servlet liefert als Antwort eine dynamisch erstellte Map-Mashup-Seite zurück.

8.1.5 Servlet `ResponseServlet`

Das Servlet `ResponseServlet` nimmt Anfragen der dynamisch erstellten Map-Mashup-Dokumente entgegen. Es entnimmt den Anfragen die benötigten Parameter, um den richtigen standortspezifischen Text vom `Document`-Objekt zu erhalten und an die anfragende Seite weiterzureichen. Sind in der Anfrage die Parameter „id“ und „fileid“ gesetzt, so werden der standortspezifische Text, die Inhalte der Eingabefelder der Map-Mashup-Seite und andere Daten mit der Methode `createXMLFile` im eingestellten Ordner des Dateisystems gespeichert.

8.1.6 Klasse `Document`

Mit der Klasse `Document` werden die angeforderten Map-Mashup-Seiten erstellt. Dazu lädt es die XML-Konfigurationsdatei und erzeugt mit den daraus gewonnenen

Daten verschiedene Objektinstanzen, die Teile der gewünschte Map-Mashup-Seite erzeugen können. Die Klasse `Document` benutzt diese Objektinstanzen und liefert mit der Methode `getBody` den HTML-Body zurück und mit der Methode `getStyle` die für das Layout benötigten CSS-Anweisungen. In der Klasse `Document` gibt es verschiedene Methoden, die JavaScript-Funktionen oder Attribute der Map-Mashup-Seite zurückgeben.

Die Methode `getAreaResponse(String lat, String lng)` gibt eine Instanz der Klasse `AreaResponse` zurück. Diese enthält einen standortbezogenen Text und gibt an, ob die übergebenen Breiten- und Längengrade innerhalb eines definierten Bereiches liegen und wie dieser Bereich heißt. Die Überprüfung, ob die angegebenen Breiten- und Längengrade in einem definierten Bereich liegen führt zu einem Problem. Dieses „Punkt im Polygon“ Problem ist aber bekannt und so existiert auch bereits eine Methode `contains(int x, int y)` in der Klasse `java.awt.Polygon` die genau diese Überprüfung vornimmt. Leider akzeptiert die Methode nur `int`-Parameter, die Längen- und Breitengrade sind aber Kommabehaftet.

In der XML-Konfigurationsdatei können geografische Positionen angegeben werden, deren Gradzahl größer oder gleich null und kleiner als 360 ist. Mit der Beschränkung auf bis zu 5 Nachkommastellen ergibt sich der größte zulässige Wert von 359,99999. Um die Kommazahlen in `int`-Werte umzuwandeln, werden sie mit 100000 multipliziert. Das ergibt bei dem größten zulässigen Wert 35999999 und dieser ist immer noch kleiner als der größtmögliche `int`-Wert von 2147483647.

Der Abstand zwischen zwei aufeinanderfolgenden ganzzahligen Längen- oder Breitengraden beträgt maximal 111 Km. Mit den in der XML-Konfigurationsdatei maximal zulässigen 5 Nachkommastellen ergibt sich somit eine minimale Auflösung von 1,11 Metern, die für den Zweck der Anwendung völlig ausreichend ist.

8.1.7 JavaScriptdatei für Map-API

Mit der in die Map-Mashup-Seite eingebundenen JavaScript-Datei werden das Map-, das Geocoder- und das XML-HTTP-Request-Objekt der Google-Map-API initialisiert und angesprochen. Die Initialisierung erfolgt in der Funktion `load`, die wiederum durch das `onload`-Event des HTML-Body-Elements aufgerufen wird. Nicht benötigte Ressourcen werden später wieder durch den Aufruf der Funktion `unload` durch das `unload`-Event des HTML-Body-Elements freigegeben. Durch anklicken der Formular-Absenden-Schaltfläche wird die Funktion `skload` aufgerufen. Hier wird zunächst einmal überprüft, ob alle erforderlichen Eingaben getätigt worden sind und anschließend aus den Eingabedaten eine Adresse erstellt. Die Methode `getLocations` des Map-API-Geocoder-Objekts wird anschließend mit dem Adressparameter und dem Namen der Callback-Funktion `getAddressesCB` aufgerufen. Sie ruft die übergebene Funktion `getAddressesCB` asynchron mit einem Response Parameter auf. Aus diesem Parameter werden die gefundene Adresse und ihre geografische Position entnommen. Ist die Adresse nicht eindeutig oder wurde nicht gefunden, dann wird eine Fehlermeldung im Clientbrowser ausgegeben, ansonsten wird die Funktion `mmsAjaxRequest` aufgerufen. Hier wird eine XML-HTTP-Anfrage erstellt, auf deren Antwort mit der Callbackfunktion `getAreaInfoCB` asynchron gewartet wird. Die Antwort enthält den standortspezifischen Text, mit dem anschließend der Textausgabebereich gefüllt wird. Die Übersichtskarte und die Gesamtkarte wurden bereits nach dem Aufruf der Funktion `mmsAjaxRequest` an die neue Adresse angepasst.

8.2 Tool zur Erstellung einer XML-Konfigurationsdatei

Dieses Tool dient zur einfachen Erstellung von XML-Konfigurationsdateien. Es fragt die benötigten Daten nacheinander vom Benutzer ab und gibt vorher Erklärungen zu den jeweils möglichen Einstellungen ab.

8.2.1 ConfigToolServlet

Das `ConfigToolServlet`-Servlet dient als Controller für das XML-Konfigurationsdatei-Tool. Es nimmt die Anfragen entgegen und leitet sie an die entsprechenden JSP's weiter. Es initialisiert das `XMLToolData`-Objekt und stellt es

jedem Request zur Verfügung. Die beiden Methoden `doGet` und `doPost` führen die Methode `doControll` aus und übergeben ihr dabei das Response- und das Request-Objekt. Ein weiterer Parameter, den die Methode `doControll` erwartet ist ein Boolean, der angibt, ob es sich um eine Post-Anfrage handelt. Jede JSP, zu der weitergeleitet werden soll, hat eine eigene Methode, die als Parameter wieder die Objekte Request, Response und ein Boolean `isPost` erwarten. Die Methode `doControll` ruft entsprechend der gewünschten JSP die zugehörige Methode der JSP auf. In Methoden von JSP's die HTTP-Post Anfragen erhalten können, wird geprüft, ob es sich um eine Post-Anfrage handelt. Ist das der Fall, so werden die entsprechenden Parameter ausgelesen und an das `XMLToolData`-Objekt übergeben. Anschließend wird die Anfrage unabhängig von der HTTP-Methode an die gewünschte JSP weitergeleitet.

8.2.2 XMLDownloadServlet

Das Servlet `XMLDownloadServlet` dient dazu, aus den eingegebenen Konfigurationsdaten eine XML-Datei zum Download anzubieten. Das `XMLDownloadServlet`-Servlet erhält alle Anfragen, die auf die Ressource `/configtool/mmsconfig.xml` zugreifen wollen. Erfolgt ein solcher Zugriff, dann wird zunächst über die statische Methode `getMarshaller` die Referenz auf einen neuen Marshaller geholt. Dieses Vorgehen ist notwendig, da die Marshaller-Klasse nicht threadsicher ist. Anschließend wird die Methode `marshall` des Marshaller-Objektes benutzt, um die angeforderte XML-Konfigurationsdatei zu erstellen. Sie erwartet als Parameter eine Objektrepräsentation des XML-Dokumentes und einen Ausgangsstrom, in den sie das zu erstellende Dokument schreiben kann. Die Objektrepräsentation erhält sie mit der Methode `getMapMashupObject` des `xmlToolData`-Objekts und als Ausgangsstrom wird der Ausgangsstrom des Response-Objektes genutzt.

8.2.3 Klasse XmlToolData

In der Klasse `XmlToolData` werden die von den Benutzern eingegebenen Formulardaten gespeichert. Dazu wird mit einer `ObjectFactory`-Instanz eine Java-Objektrepräsentation eines XML-Konfigurationsdokumentes erstellt. Mit

verschiedenen Methoden werden den in der Objektrepräsentation enthaltenen Objekte Werte zugewiesen. Die Methode `getMapMashupObject` liefert die Objektrepräsentation zurück. Sie wird an anderer Stelle als Parameter der `marshall`-Methode eines `Marshaller`-Objektes übergeben, die auf diese Weise ein XML-Konfigurationsdokument erstellt.

8.2.4 Custom Tags

In den JSP's werden die zwei Custom Tags `AreaJSTag` und `AreaOptionsTag` verwendet, die in der Tag Library Descriptor-Datei `mmtaglib.tld` beschrieben werden. Die Custom Tags wurden in dem Paket `customtags` zusammengefasst und erweitern jeweils die Klasse `TagSupport`.

8.2.4.1 *mmtaglib.tld*

Hier werden den beiden Custom Tags ihre Tag-Namen und die zu benutzende Tag-Klasse zugewiesen. Ein weiterer Parameter `bodycontent` wird jeweils auf `empty` gesetzt, was bedeutet, dass die Custom Tags keinen Inhalt besitzen. Die erstellte Tag Library Descriptor-Datei wurde in dem Ordner `WEB-INF/lib` platziert und mit dem Unterelement „taglib“ des `jsp-config`-Elementes in der `web.xml` referenziert. In JSP's, die Custom Tags benutzen, wurde mit der `taglib`-Direktive die erstellte Tag Library Descriptor-Datei referenziert und so die Benutzung der Custom Tags mit einem wählbaren Präfix möglich gemacht.

8.2.4.2 *AreaJSTag*

Dieses Tag bindet ein JavaScript-Array ein, in dem alle geografisch definierten Bereiche mit Namen und geografischen Informationen aufgeführt sind. Es definiert die Methode `doStartTag` in der es die Methode `getAreasJavaScript` des `XMLToolData`-Objekts aufruft und den erforderlichen JavaScript-Code erzeugt. Der Rückgabewert dieser Methode ist die Konstante `TagSupport.SKIP_BODY` die dafür sorgt, dass weder der Körper noch das schließende Tag verarbeitet werden.

8.2.4.3 *AreaOptionsTag*

Mit diesem Tag werden die geografischen Bereichsnamen als Liste von Option-Tags ausgegeben, wobei die erste Option selektiert ist. Das `AreaOptionsTag` wird innerhalb des `select`-Tags benutzt. Es definiert die Methode `doStartTag` in der es auf die Methode `getWholeMapAreaNames` des `XMLToolData`-Objekts zugreift und die erforderlichen Ausgaben erzeugt. Auch hier ist der Rückgabewert der Methode `TagSupport.SKIP_BODY`.

8.2.5 include-Direktive

Mit der `include`-Direktive ist es möglich Text in eine JSP einzufügen und so häufig gebrauchte JSP-Bestandteile auszulagern. Dabei wird beim Übersetzen der JSP in ein Servlet der Text an die Stelle der `include`-Direktive kopiert. Eine andere Möglichkeit Text in bestehende JSP's einzufügen wäre die `jsp:include` action, sie hat jedoch den Nachteil, dass hier die Einbindung zur Laufzeit geschieht. Damit ist sie nicht so performant wie die `include`-Direktive. Die `include`-Direktive wird hier für drei JSP-Fragmente benutzt.

1. `include-noscript.jsp`
Definiert den `noscript`-Bereich, der ausgegeben wird, falls der Clientbrowser kein JavaScript unterstützt.
2. `include-partHeadline.jsp`
Definiert ein fieldset, in dem eine Überschrift eingegeben werden kann.
3. `include-partFormat.jsp`
Definiert ein fieldset, in dem die Breite und die Höhe eines Elementes ausgewählt werden kann.

8.2.6 JavaScriptdatei `mms.js`

Diese JavaScript-Datei enthält Funktionen und Variablen, die von vielen JSP's neben ihren lokal definierten Funktionen benutzt werden. Es handelt sich dabei um Funktionen, die zur Auswahl von Farben benutzt werden und um Elemente unsichtbar oder sichtbar zu machen.

8.2.7 Cascading Stylesheet ConfigToolFormate.css

Mit dieser Stylesheetdatei wird das Layout des XML-Konfigurationsdateitools bestimmt. Änderungen an der Farbe, der Position und des Aussehens von Elementen werden hier vorgenommen. Die meisten Einstellungen beziehen sich auf Elementtypen oder Klassen, einige Elemente werden aber auch direkt formatiert.

8.3 Exceptions

Es wurden die Exception-Klassen `MmsException` und `XmlFileException` implementiert, um für diese Ausnahmen Fehlermeldungen mit Angabe der Ursache ausgeben zu können. Bei anderen Ausnahmen wird die Ursache nicht angezeigt und nur eine einfache Mitteilung auf der Fehlerseite ausgegeben.

8.3.1 MmsException

Diese Art von Ausnahme wird geworfen, wenn Initialisierungsparameter der Web-Applikation fehlerhaft sind, fehlen oder auf eine benötigte Ressource nicht zugegriffen werden kann.

8.3.2 XmlFileException

Bei Fehlern in einer XML-Konfigurationsdatei wird eine Ausnahme diesen Typs geworfen. Um einen genauen Hinweis auf die Fehlerposition geben zu können wurde ein eigener `ValidationEventHandler` erstellt, durch den die genaue Fehlerposition im XML-Dokument zur Verfügung gestellt wird.

8.4 Logging

Systemmeldungen werden über einen Logger ausgegeben und gespeichert. Die Klasse `MmsLogger` nutzt die Klasse `java.util.logging.Logger`. In der statischen Methode `initialize(aLogFile, aLogLevel)` wird eine Logger-Instanz erstellt und mit zwei Handlern verbunden. Der erste Handler ist für die Textausgabe auf der Konsole zuständig und der zweite schreibt die Systemmeldungen in ein Logfile. Der Parameter `aLogLevel` gibt an, ab welchem Level Systemmeldungen ausgegeben und gespeichert werden sollen. Die benutzte Klasse `Logger` besitzt sieben unterschiedliche Log-Level, die nach Wert geordnet

folgendermaßen lauten: `FINEST`, `FINER`, `FINE`, `CONFIG`, `INFO`, `WARNING` und `SEVERE`. Wird z.B. bei der Initialisierung der Level `WARNING` angegeben, werden Systemmeldungen des Levels `INFO` und darunter nicht geloggt. Auf die erstellte Logger-Instanz kann mit der Klassenmethode `getLogger` zugegriffen werden.

8.5 Deployment

Diese Webanwendung besteht aus vielen unterschiedlichen Komponenten. Zur Vereinfachung der Installation werden die Komponenten in einer einzigen Datei zusammengefasst. Dieses Webarchiv hat die Endung `.war` und wird im Webcontainer installiert. Mit der Entwicklungsumgebung Eclipse ist es möglich, ein solches Webarchiv zu erstellen. Es musste nur im Eclipse-Project-Explorer das `MapMashupService`-Projekt markiert werden. Mit einem rechten Mausklick konnte dann im Kontextmenü der Punkt „`export > war file`“ ausgewählt werden. Als Speicherort wurde das Verzeichnis `/webapps` im Tomcat Wurzelverzeichnis gewählt. Der Name der erstellten Datei lautete `MapMashupService.war` und wurde von Tomcat sofort erkannt und automatisch installiert.

8.6 Fazit

Mit der Implementierung der Webanwendung wurde gezeigt, dass es möglich ist eine Map-Mashup-Seite mit Hilfe einer XML-Konfigurationsdatei dynamisch zu erstellen. Die Unterstützung von Java Server Pages, Servlets, Filter, Listener, Cascading Stylesheet, JavaScript und HTML durch Eclipse hat die Implementierung sehr erleichtert. Das XML-Konfigurationsdatei-Tool ermöglicht es auch ohne XML-Kenntnisse eine XML-Konfigurationsdatei zu erstellen.

Die Implementierung der Exception-Klassen und des Loggers hilft bei Laufzeitfehlern eine genauere Fehlermeldung zu erhalten und diese zu speichern. Die Google-Maps API ist einfach zu nutzen und bietet genaues Kartenmaterial und eine gute Geocodierung von Adressen. Nachteilig war allerdings, dass Versionsupdates der API nicht immer transparent waren und es so vorkam, dass fertige Funktionen geändert werden mussten, um mit der aktuellen stabilen API-Version in gewünschter Weise zu funktionieren.

Das Deployment der Webanwendung war durch die erstellte `.war`-Datei sehr einfach durchzuführen.

9 XML-Konfigurationstool Usability

Nachdem beim Design des XML-Konfigurationstools festgelegt wurde, dass sich die Gestaltung der Webseiten an den 10 Usability Heuristiken von

9.1 Überprüfen der Usability des XML-Konfigurationstools

Die Gebrauchstauglichkeit des XML-Konfigurationstools hat Einfluss auf das Erreichen der nicht-funktionalen Anforderung Punkt 2, Kapitel 2.4. Die Bewertung der Gebrauchstauglichkeit eines Produktes erfordert normalerweise Tests mit mehreren Personen. Dazu erhalten die Personen eine Aufgabe, die sie mit dem Produkt erledigen sollen. Nach Erledigung der Aufgabe füllen sie ein Formular aus und bewerten dabei einzelne Punkte der Gebrauchstauglichkeit. Die Usability wird anschließend aus den Durchschnittswerten der einzelnen Formularpunkte ermittelt.

Im Rahmen dieser Arbeit wurde der Test vereinfacht und nur mit einer Person durchgeführt. Die Person erhielt den Auftrag eine XML-Konfigurationsdatei zu erstellen und wurde dabei zwar beobachtet aber nicht unterstützt. Nachdem die Konfigurationsdatei erstellt war, musste die Person aber anschließend keinen Fragebogen ausfüllen. Sie wurde zu den einzelnen Punkten der 10 Usability Heuristiken in bezug auf die Anwendung mündlich befragt und für jeden einzelnen Punkt eine Bewertung erbeten. Mögliche Bewertungen waren gut, ok und schlecht.

	Bewertungspunkt	Bewertung	Benutzerkommentar
1	Einfache und natürliche Dialoge	gut	
2	Ausdrucksweisen des Anwenders benutzen	ok	Elementüberschriften? Was ist das?
3	Minimale mentale Belastung des Benutzers	ok	...musste mir die Erklärungen von der vorigen Seite merken
4	Konsistenz einhalten	gut	
5	Rückmeldungen geben	gut	
6	Klare Auswege zeigen	gut	
7	Abkürzungen anbieten	gut	Die Vorgaben sparen Zeit
8	Gute Fehlermeldungen ausgeben	gut	...sagen genau was zu tun ist
9	Fehlervermeidung betreiben	gut	
10	Hilfe und Dokumentation anbieten	schlecht	Mir fehlte ab und zu eine direkte Hilfe, um nicht wieder zurück auf die vorige Seite zu müssen

Tabelle 10: Usabilitybewertung

Wie in der Tabelle 10 zu sehen ist, wurde überwiegend die Bewertung „gut“ für die Usability des XML-Konfigurationstools gegeben. Lediglich die Bewertungspunkte „Ausdrucksweise des Anwenders nutzen“ und „Minimale Mentale Belastung des Benutzers“ bekamen ein „ok“. Mit „schlecht“ wurde nur der Bewertungspunkt „Hilfe und Dokumentation anbieten“ bewertet.

9.1.1 Fazit

Obwohl die Usability-Bewertung nur auf dem Urteil einer einzigen Person beruht, kann man aber dennoch schon sehen, an welcher Stelle noch Verbesserungen vorgenommen werden können. Die beiden ok Bewertungen für die Punkte 2 und 3 scheinen genau wie die schlechte Bewertung für Punkt 10 die Ursache in der fehlenden Kontexthilfe zu haben.

Die gute Bewertung der übrigen Punkte ist hier natürlich nicht repräsentativ, da z.B. von der Testperson ausgiebig von den in den Auswahlfeldern voreingestellten Werten gebrauch gemacht wurde. Eine anderen Testperson hätte diese Werte vielleicht an vielen Stellen geändert und keinen Vorteil in den Voreinstellungen gesehen.

Zusammengenommen ergibt sich aber doch eine positive Bewertung der Usability, die auch dadurch gestützt wird, dass die Testperson die Aufgabe in kurzer Zeit erfolgreich beendete.

10 Tests

In diesem Kapitel soll überprüft werden, ob sich eine erstellte Map-Mashup-Seite einfach in einen Webauftritt integrieren lässt.

10.1 Einbindung in Webauftritt

Da keine fertigen Webauftritte zur Verfügung standen an denen die Integration einer Map-Mashup-Seite getestet werden konnte, wurden Webseiten-Templates benutzt. Das sind fertige Vorlagen für Webseiten, die nur noch mit eigenen Inhalten gefüllt werden müssen. Sie werden oft bei der Erstellung von kleineren und mittleren Webauftritten eingesetzt und bieten somit eine gute Testbasis für diese Anwendung, deren Zielgruppe auch kleine und mittlere Webauftritte sind. Die hier benutzen Templates sind kostenfrei für den privaten Gebrauch und stammen von den folgenden Anbietern:

- www.freecsstemplates.org
Benutzt für den Test mit dynamischer Seitengenerierung und Zugriff auf Eingabedaten
- www.dotemplate.com
Benutzt für den Test mit statischen HTML-Seiten und Zugriff auf area-Parameter
- www.on-mouseover.de
Benutzt für den Test als Frame in einem Frameset

10.1.1 ...mit dynamischer Seitengenerierung und Zugriff auf Eingabedaten

Bei dieser Art von Webauftritt kann man Zugriff auf die in der Map-Mashup-Seite eingegebenen Formulardaten erhalten. Dazu mussten dem Link zur Map-Mashup-Seite die zusätzlichen Parameter `id` und `fileid` übergeben werden. Die beiden Parameter wurden dabei bereits serverseitig in den Link integriert und zwischengespeichert. Durch den von der Map-Mashup-Seite zurückführenden Link konnte festgestellt werden, ob Formulardaten abgespeichert wurden. Der Zugriff auf die Formulardaten erfolgte über einen Abgleich des Parameters `id` im zurückführenden Link und des zwischengespeicherten Parameters `id`. Mit der zum

zwischengespeicherten Parameter `id` gehörenden Parameter `fileid` konnte die herunterzuladende Datei bestimmt werden.

Nachdem die Daten geladen wurden, konnten diese in der Webanwendung benutzt werden.

10.1.2 ...mit statischen HTML-Seiten und Zugriff auf `area`-Parameter

Mit statischen HTML-Seiten kann zwar nicht auf die eingegebenen Formulardaten zugegriffen werden, über den von der Map-Mashup-Seite zurückführenden Link kann aber der Bereich gewonnen werden, zu der die vom Benutzer eingegebene Adresse gehört. Die vom zurückführenden Link angesprochene Seite wurde um eine JavaScript-Funktion erweitert, in der mit dem `location`-Objekt der übergebene Parameterteil eingelesen und nach dem Parameter `area` durchsucht wurde. Der Wert dieses Parameters konnte anschließend benutzt werden.

10.1.3 ...als Frame in einem Frameset

Wird die Map-Mashup-Seite als Frame in einem Frameset benutzt, befinden sich Navigationselemente oft in eigenen Frames. Es würde das einheitliche Erscheinungsbild der Webanwendung stören, wenn in der Map-Mashup-Seite eine Schaltfläche erscheinen würde, mit der man die Seite verlassen kann. In diesen Fällen kann nicht der mit dem Link übergebene Parameter `area` ausgewertet werden und falls die Frame-Seiten nicht dynamisch erstellt werden, kann auch nicht auf die Formulardaten zugegriffen werden. Die auf der Map-Mashup-Seite angezeigten Informationen stellen hier die einzige Quelle dar. In der nachfolgenden Abbildung ist ein Szenario dargestellt, in der ein Benutzer seine Standortinformationen eingegeben hat und eine einfache Auskunft zu möglichen Liefertagen in seinem Gebiet erhält. Es ist zu sehen, dass sich die Map-Mashup-Seite gut in die vorgegebene Seite integriert. Das ist unter anderem darauf zurückzuführen, dass Größen und Farben der verwendeten Elemente der Vorgabe angepasst wurden.

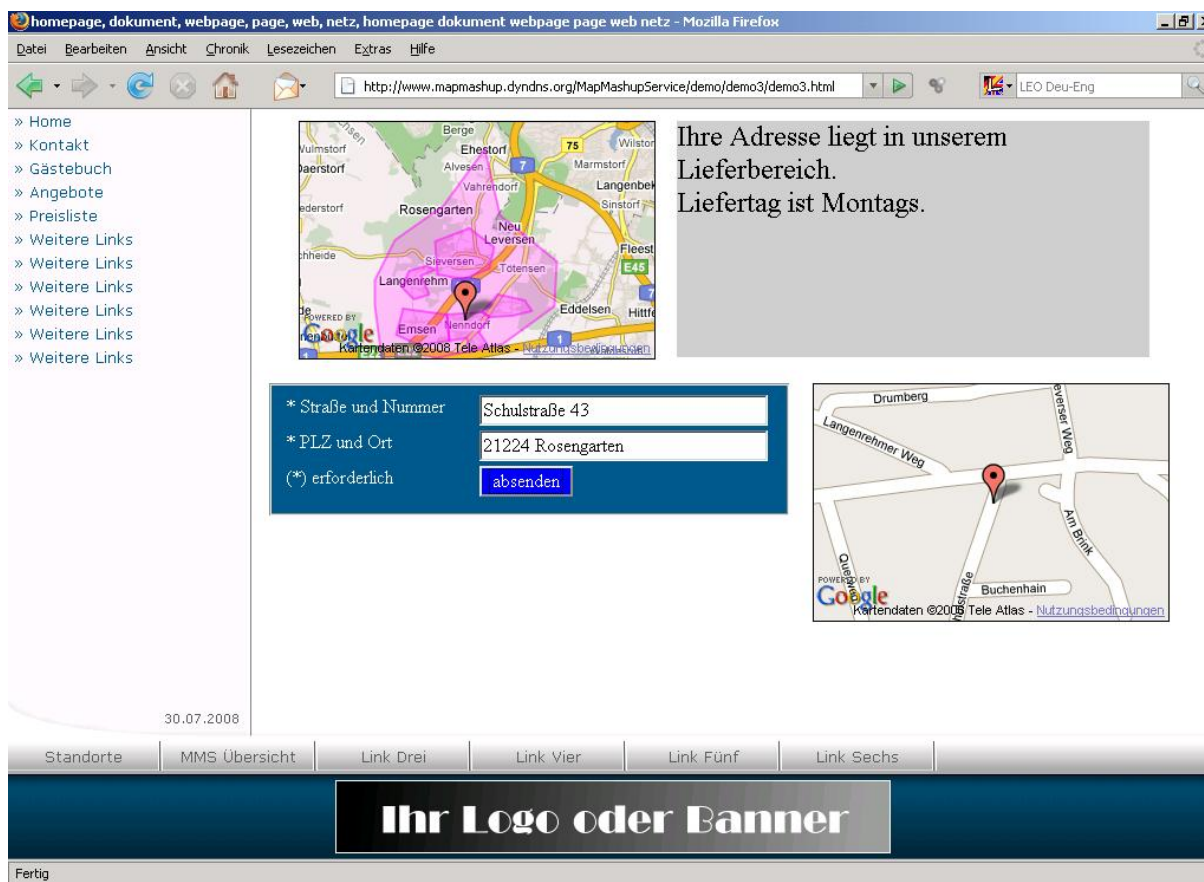


Abbildung 13: Map-Mashup-Seite in Frameset

10.1.4 Fazit

Die Integration der dynamisch erstellten Map-Mashup-Seiten ist in allen drei Fällen gelungen. Die Integration in ein Frameset war dabei am einfachsten. Es musste nur ein entsprechender Link auf die Map-Mashup-Seite gelegt werden. Auch sehr einfach war die Integration mit Zugriff auf den `area`-Parameter. Hier genügte ein Link auf die Map-Mashup-Seite und eine kurze JavaScript-Funktion auf der Webseite, die von der Map-Mashup-Seite referenziert wird. Relativ aufwendig war die Integration mit Zugriff auf die Eingabedaten. Dies liegt daran, dass es mit JavaScript nicht möglich ist Ressourcen zu laden, die nicht von dem gleichen Server stammen wie das HTML-Dokument in dem der JavaScript-Code läuft. Auch auf Cookies von anderen Servern kann nicht zugegriffen werden. Durch diese Einschränkungen war es notwendig den Dateidownload auf der Serverseite durchzuführen, was zusätzliche Client-Server Kommunikation erforderte.

11 Fazit und Ausblick

Mit dieser Arbeit wurde erfolgreich der Prototyp einer Webanwendung erstellt, die dynamisch Map-Mashup-Seiten mit Hilfe von XML-Konfigurationsdateien erstellt. An die Webanwendung wurden die folgenden funktionalen Anforderungen gestellt:

- Webanwendung ist konfigurierbar:
Dies wird über die `context-param`-Elemente im Deployment Descriptor der Webanwendung realisiert.
- Benutzerdaten können gepflegt werden:
Die Benutzerdaten werden in einem XML-Dokument, der `user.xml`, verwaltet und können dort auch verändert werden.
- Map-Mashup-Seiten können mit XML-Konfigurationsdatei angepasst werden:
Zu jeder Map-Mashup-Seite gehört eine XML-Konfigurationsdatei, durch die der Aufbau der Seite bestimmt wird.
- Eingabedaten der Webseitennutzer können gespeichert und wieder abgerufen werden:
Durch Übergeben der Parameter `id` und `fileid` werden die Eingabedaten der Webseitenbenutzer eine einstellbare Zeit lang gespeichert und können durch Angabe des Dateinamens auch wieder geladen werden
- Auf der erstellten Seite werden standortbezogene Texte ausgegeben:
Im Ausgabebereich der Seite wird nach Überprüfung der eingegebenen Adresse ein Text ausgegeben, der davon abhängt, in welchem geografischen Bereich sich die Adresse befindet.
- Die erstellte Seite ist ein Map-Mashup:
Auf der erstellten Seite werden mit Hilfe der Google-Maps-API eine Übersichtskarte und eine Karte mit der Position der eingegebenen Adresse angezeigt.

Damit werden alle funktionalen Anforderungen von der erstellten Webanwendung erfüllt. Als nächstes werden die nicht-funktionalen Anforderungen überprüft:

- Eingabedaten werden geschützt übertragen:
Die Eingabedaten werden über HTTPS verschlüsselt übertragen.

- Erstellung der XML-Konfigurationsdatei ist möglichst einfach:
Es wurde das XML-Konfigurationsdatei-Tool erstellt, mit dem es möglich ist ohne XML-Kenntnisse eine XML-Konfigurationsdatei zu erstellen. Die Gebrauchstauglichkeit wurde in einem Test nachgewiesen.
- Einfache Integration in Webauftritte:
Es wurde die Integration einer Map-Mashup-Seite in einen Webauftritt in drei Szenarien getestet. Die Integration war bei allen Szenarien grundsätzlich einfach, es war allerdings etwas aufwändig die Eingabedaten abzuspeichern und wieder einzulesen.
- Benutzung von Open Source Technologien:
Bis auf die Map-API stehen alle benutzten Werkzeuge und Systemkomponenten unter einer Open Source Lizenz.
- XML-Konfigurationsdatei kann jederzeit verändert werden:
Da die XML-Konfigurationsdatei vom Webseitenanbieter selber verwaltet wird, hat er alle Möglichkeiten um sie jederzeit zu verändern.

Die nicht-funktionalen Anforderungen konnten nicht vollständig erfüllt werden, da die Integration mit Zugriff auf die Eingabedaten etwas kompliziert ist und es keine Open Source Map-API gab, die benutzt werden konnte.

In Zukunft könnte aber der Zugriff auf die Eingabedaten erleichtert werden. In dem Arbeitspapier „Access Control for Cross-side Requests“ [CSR] des World Wide Web Consortiums (W3C) wird eine Möglichkeit beschrieben, mit der von Clients auf Ressourcen verschiedener Server zugegriffen werden kann. Sollte dieses Arbeitspapier zu einer Empfehlung des W3C werden, könnten Clients direkt auf die Eingabedaten zugreifen und der Zugriffsweg über den zweiten Server wäre in diesen Fällen nicht mehr nötig.

Die verwendete Google-Maps-API ist für dieses Projekt kostenlos nutzbar. In seinen Nutzungsbestimmungen hält sich Google jedoch die Möglichkeit offen, Werbung in die Karten einzubinden. Sollte das eines Tages geschehen, bestünden die Möglichkeiten die Werbung auf den Karten zu akzeptieren, das kostenpflichtige Google Maps für Unternehmen zu nutzen oder auf eine andere Map API zu wechseln. Es wird vom Umfang der Werbung, den Kosten und der Qualität anderer Map API's abhängen, welche Entscheidung getroffen wird.

Bei der Erstellung des Systems ließen sich alle benötigten Komponenten gut integrieren und die eingesetzten Tools vereinfachten die Entwicklung. Insbesondere die Entwicklungsumgebung Eclipse in der Version für Java EE Entwickler erwies sich als besonders hilfreich, da sie Unterstützung für die verschiedensten Dokumenttypen bot.

Bei der Entwicklung des Prototypen wurde das Qualitätsmerkmal Performance nicht als eigene Anforderung aufgeführt, da zunächst einmal die Machbarkeit der Webanwendung bewiesen werden sollte. Um im Produktivbetrieb eingesetzt zu werden, könnte es daher notwendig sein die Laufzeitumgebung zu optimieren. In einer weiteren Maßnahme könnten mit einer Änderung im Programmcode Konfigurationsdateien als Objektrepräsentation zwischengespeichert werden, was einerseits zu einer Verringerung der aufwändigen Lade- und Marshalling-Operationen führen würde, andererseits aber bedeuten würde, dass Änderungen an Konfigurationsdateien nicht unbedingt sofort registriert werden.

Neben diesen Performance-Maßnahmen gibt es aber auch noch andere nützliche Erweiterungen, die in spätere Versionen eingebaut werden könnten.

- Wiedereinlesen von XML-Konfigurationsdateien zur Bearbeitung
- Auswahl von Regionen, Städten, usw. aus Liste zur Definition eines Bereiches
- Automatische Voreinstellung von Farben, Schrifttypen, Schriftgrößen und Hintergrundmuster mit Hilfe eines Vorlage-HTML-Dokuments

All diese Maßnahmen würden die Erstellung von XML-Konfigurationsdateien erleichtern und somit zu einer größeren Akzeptanz der gesamten Webanwendung führen.

12 Literaturverzeichnis

- [CBEGLW04] Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li, Chanoch Wiggers. Professional Apache Tomcat 5. Wiley Publishing, Inc., 10475 Crosspoint Boulevard, Indianapolis, IN 46256, 2004. ISBN 0-7645-5902-8
- [CSR] World Wide Web Consortium. Access Control for Cross-site Requests. W3C Working Draft, 2008. <http://www.w3.org/TR/access-control>
- [Dah06] Markus Dahm. Grundlagen der Mensch-Computer-Interaktion. Pearson Studium, 2006. ISBN 3-8273-7175-9
- [DDGLSZ03] H.M.Deitel, P.J.Deitel, J.P.Gadzick, K.Lomeli´, S.E.Santry, S.Zhang. Java Web Services For Experienced Programmers. Prentice Hall, Upper Saddle River, New Jersey 07458, 2003. ISBN 0-13-046134-2
- [Dom] World Wide Web Consortium. Document Object Model (DOM). Homepage. <http://www.w3.org/DOM>
- [Dyn] DynDNS Hauptseite <http://www.dyndns.com/>
- [Eclip] Eclipse-Downloadseite <http://www.eclipse.org/downloads/>
- [Fit] Fit für Usability Webseite <http://www.fit-fuer-usability.de/>
- [FJ03] Jayson Falkner, Kevin Jones. Servlets and JavaServer Pages: The J2EE Technology Web Tier. Addison-Wesley, 2003. ISBN 0-321-13649-7
- [FS07] David Flanigan, Lars Schulten. JavaScript kurz & gut. O'Reilly Verlag GmbH & Co.KG, 3.Auflage, 2007. ISBN 978-3-89721-531-3
- [Gam06] Johannes Gamperl. AJAX: Web 2.0 in der Praxis. Galileo Press, Bonn, 2006. ISBN 3-89842-764-1
- [GAPIDoc] Google Maps API Dokumentation. <http://code.google.com/apis/maps/documentation/index.html>
- [Gar05] Jesse James Garrett, Ajax: A New Approach to Web Applications, 2005, <http://www.adaptivepath.com/ideas/essays/archives/000385.php>
- [GKey] Webseite, um Google-Map-API-Key zu beantragen <http://code.google.com/apis/maps/signup.html>
- [JAXB] Downloadseite für JAXB 2.0 Referenzimplementierung <https://jaxb.dev.java.net/>
- [JCP] Java Servlet 2.5 Specification <http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index.html>
- [JDK5] Downloadseite für JDK5 auf der Internetseite von Sun Microsystems Inc http://java.sun.com/javase/downloads/index_jdk5.jsp
- [KS06] Ayub Khan, Marina Sum. Introducing Design Patterns in XML Schemas. Sun Developer Network Article, 2006.

- http://developers.sun.com/jsenterprise/archive/nb_enterprise_pack/reference/techart/design_patterns.html
- [Lab06] Kai Laborenz. CSS-Praxis. Galileo Press, Bonn, 4.Auflage, 2006. ISBN 3-89842-765-X
- [McL02] Brett McLaughlin. Java & XML. O'Reilly Verlag GmbH & Co.KG, 2.Auflage, 2002.
ISBN 3-89721-296-X
- [Mün06] Stefan Münz. Professionelle Websites: Programmierung, Design und Administration von Webseiten. Addison-Wesley, 2006. ISBN 3-8273-2370-3
- [SBA] Statistisches Bundesamt <http://www.destatis.de>
- [Schema0] World Wide Web Consortium. XML Schema Part 0:Primer. W3C Recommendation, 2001.
<http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>
- [Schema1] World Wide Web Consortium. XML Schema Part 1:Structures. W3C Recommendation, 2001.
<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>
- [Schema2] World Wide Web Consortium. XML Schema Part 2:Datatypes. W3C Recommendation, 2001.
<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>
- [See06] Michael Seeboerger-Weichselbaum. Programmieren mit Eclipse 3. mitp-Verlag, Bonn, 1.Auflage, 2006. ISBN 3-8266-7379-4
- [Tag] Downloadseite für Standard Tag Library 1.1
http://jakarta.apache.org/site/downloads/downloads_taglibs.html
- [TOM] Apache Tomcat Homepage <http://tomcat.apache.org/index.html>
- [Ull07] Christian Ullenboom. Java ist auch eine Insel. Galileo Computing, 7.Auflage, 2007.
ISBN 978-3-8362-1146-8. <http://www.galileocomputing.de/openbook/javainsel7>
- [Val] Homepage des Online Validators von Validome <http://www.validome.org>

13 Glossar

AJAX	Asynchron JavaScript and XML
API	Application Program Interface
CERN	Conseil Européen pour la Recherche Nucléaire zu deutsch : europäische Organisation für Kernforschung
DOM	Document Object Model
JAXB	Java Architecture for XML Binding
JDK	Java Development Kit
JSP	Java Server Page
Logfile	Datei, in der bestimmte Aktivitäten eines Computersystems protokolliert werden.
Map-Mashup	Eine Webseite, die Informationen aus mehr als einer Quelle enthält und auf der Informationen zusätzlich mit Hilfe von Karten dargestellt werden
marshalling	Ist der Prozess um eine Objekt-Repräsentation in ein XML-Dokument umzuwandeln
SSL	Secure-Sockets-Layer hilft Übertragungen im Internet zu schützen
unmarshalling	Ist der Prozess um ein XML-Dokument in eine Objekt-Repräsentation umzuwandeln
valide	Hier: Besagt, dass ein XML-Dokument wohlgeformt ist und einer angegebenen Struktur gehorcht.
W3C	World Wide Web Consortium
Webseitenanbieter	Hier ist damit eine Person gemeint, die eine eigene Webseite betreibt und den Map-Mashup-Service nutzt, um auf seiner Webseite standortbezogene Informationen anzubieten
Webseitennutzer	Hier ist damit eine Person gemeint, die eine Webseite nutzt, deren Inhalt zumindest teilweise vom Map-Mashup-Service erzeugt wurde.
XML	Extensible Markup Language

14 Angaben zum CD-Inhalt

Auf der dieser Arbeit beiliegenden CD befinden sich folgende Daten:

Im Wurzelverzeichnis befinden sich die beiden Verzeichnisse `diplomarbeit` und `implementierung`.

Das Verzeichnis `diplomarbeit` enthält die Diplomarbeit im PDF-Format.

Das Verzeichnis `implementierung` enthält die Verzeichnisse `sourcecode` und `javadoc`.

Das Verzeichnis `sourcecode` enthält die Verzeichnisse `src` und `webcontent` in denen sich die Implementierung des Map-Mashup-Services befindet.

Im Verzeichnis `javadoc` befindet sich die Java-Dokumentation der Pakete `customtags`, `filter`, `generated`, `listener`, `mms` und `servlet`.

.

Versicherung über Selbstständigkeit

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, 28. August 2008

Ort, Datum

Unterschrift