Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Roman Kostromin

Entwicklung eines MVC-Frameworks und seine Bewertung für die effiziente Entwicklung einer WEB 2.0 Anwendung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Roman Kostromin

Entwicklung eines MVC-Frameworks und seine Bewertung für die effiziente Entwicklung einer WEB 2.0 Anwendung

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

Betreuender Prüfer : Prof. Dr. Wolfgang Renz
Zweitgutachter : Prof. Dr. Wolfgang Gerken

Abgegeben am 18. September 2008

Roman Kostromin

Thema der Bachelorarbeit

Entwicklung eines MVC-Frameworks und seine Bewertung für die effiziente Entwicklung einer WEB 2.0 Anwendung.

Stichworte

Framework, MVC, WEB 2.0

Kurzzusammenfassung

Im Rahmen der Arbeit wurde ein MVC-basiertes Framwork, welches für die Entwicklung der Web 2.0 Anwendungen geeignet ist, gebaut. Darauf basierend wurden zwei Webapplikationen entwickelt, die den Nutzen des Frameworks zu überprüfen hatten.

Roman Kostromin

Title of the paper

The development of the MVC Framework and its evaluation for the efficient creation of a WEB 2.0 application.

Keywords

Framework, MVC, WEB 2.0

Abstract

A MVC based Framework was developed in the context of this manuscript. This Framework fits for the development of the Web 2.0 applications. The two Web applications based on it were developed for checking the usability of the Framework.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

1. Einführung

Frameworks, die auf Model View Controller (MVC) Pattern basieren, eignen sich besonders für die Entwicklung der Webapplikation mit dynamischen Inhalten, zu denen besonders die WEB 2.0 Applikationen gehören. In diesem Umfeld wurde von mir für die Firma WlaroSoft Ltd. ein Framework für die effiziente Entwicklung der webbasierten Applikationen mit Schwerpunkt User Generated Content[1] entwickelt.

1.1. Motivation

Im letzten Jahrzehnt hat das Internetumfeld ein rasantes Wachstum erlebt. Durch die stetig steigende Zahl der Internetnutzer und vor dem Hintergrund der fallenden Internetverbindungskosten ist es für die Entwickler der Webseiten interessant geworden, neue Felder zu suchen, die ein bestimmtes Maß an Innovation aufweisen. Zumal die Mehrheit der Bevölkerung eine umfassendere Verfügbarkeit des Internets erzwingt. Auch neue Technologien, die ein besseres Programmieraufwand-Nutzen Verhältnis und die Entwicklung von leichten in der Bedienung benutzerfreundlichen Oberflächen ermöglichen, steigern das Interesse der Softwareentwickler, neue Webanwendungen zu entwickeln.

Im Laufe der spontanen Überlegungen und in Gesprächen mit Bekannten und Freunden wurde darüber nachgedacht, welche Webanwendungen es auf dem deutschen Internet Markt noch nicht gibt, und welche für Nutzer des Internets, insbesondere die Zielgruppe der Jugendlichen, von Interesse sein könnten. Es wurden Ideen gesammelt und auf ihre Realisierungstauglichkeit überprüft. Unter Anderem haben sich folgende Webkonzepte herauskristallisiert: Reeperbahn -Community und Deutschland-Laune (Online-Community) Projekte. Beim ersten handelt es sich um ein Webportal, welches die Jugendlichen über

[1] User Generated Content (UGC) bezeichnet Webangebote in denen Inhalt von den Benutzern erstellt wird

das Thema Reeperbahn vereinen könnte. Es sollte Funktionen bereitstellen, die die Benutzer ansprechen und die für sie nützlich sein könnten. Es wurden verschiedene Möglichkeiten der Anbindung der sozialen Kommunikationen über das Internet, z.B. Foren, Chat, E-mail, Stellen von Fotos und Videos, Gruppenbildung u.s.w. untersucht, die in die Anwendung integriert werden sollten.

Bei dem anderen Projekt – Deutschlandlaune – handelt es sich um ein Webportal, welches die etwas romantisch angelegten Personen zur Kommunikation und Unterhaltung vereinen könnte. Z. B. In Form einer Deutschland-Karte mit der Eintragung von Städten oder anderen Örtlichkeiten auf der Oberfläche könnten die Konsumenten dieser Webseite durch den Anklick Ihres geografischen Ortes dort einen Smiley-Kreis platzieren, entsprechend der aktuellen und individuellen Stimmung. Für diese Seite bedarf es keiner Registrierung mit dem Namen. So könnten andere Anwender durch den Blick auf die Karte sehen, an welchem Ort Deutschlands welche Smileys (gut oder schlecht gelaunte) in der Mehrheit sind. Auf der Basis der persönlichen Empfindungen würden dann Kommunikationen entwickelt werden und Kontakte geknüpft werden. (Z.B. Smiley „Laune Super“ - Beitrag evtl. in Form einer Sprechblase auf der Karte, die über das Berühren des Smileys erscheint „Es geht mir gut, denn...“ Andere könnten auf diese Beiträge mit Kommentaren antworten und mit dem Smiley und Beitrag-Publizierer seine Freude teilen oder Trost spenden.

Für die Verwirklichung dieser Ideen bedurfte es eines funktionierenden Frameworks, mit Hilfe dessen eine zu hoher sozialer Kontaktinteraktion befähigte Webanwendung entstehen konnte.

Für die Entwicklung der WEB 2.0 Applikationen sollte dieses Framework eine Modul View Controlle (MVC) Architektur bereitstellen.

Die Motivation dieser Arbeit liegt darin ein funktions- und erweiterungsfähiges hochkonfiguriertes Framework effektiv zu konstruieren, in der Weise, dass die auf seiner Basis zu entwickelnden Webapplikationen auch erweiterbar und sehr effizient in der Entwicklung wären.

1.2. Zielsetzung

Die vorliegende Bachelorarbeit verfolgt gleichzeitig mehrere Ziele. Zum einen soll SWA[2] Framework entwickelt werden. Zum zweiten sollen zum Testzweck darauf basierend zwei Applikationen entwickelt werden. Zum dritten sollen durch die Analyse dieser beiden Testpiloten konzeptionelle Verbesserungen an dem Framework durchgeführt werden.

[2] SWA steht für Simple Web Architektur

Dieses Framework muss eine leichtgewichtige und leichterweiterbare Architektur für den Bau einer Webapplikation anbieten, das durch einfache Benutzung gekennzeichnet ist.

1.3. Szenario

Unter der Annahme, dass die Web 2.0-Applikationen aufgrund der ständig wachsenden Anforderungen erweitert werden müssen, soll ermöglicht werden, mit dem zu entwickelnden SWA-Framework die modular aufgebauten komplexen Webapplikationen zu entwickeln sowie diese mit einem möglichst geringem Aufwand zu warten.

1.4. Gliederung

Im ersten Kapitel wurden bereits die Motivation und die Zielsetzung der Bechalorarbeit vorgestellt.

Im zweiten Kapitel wird das Einsatzgebiet für den Framework und dann die theoretischen Grundlagen vorgestellt. Hierzu gehören das Verständnis von Model View Controller Entwurfsmuster und die Inversion of Control Paradigma.

Im dritten Kapitel wird eine Analyse der Problemstellung durchgeführt, die Anforderungen an das Framework gestellt, sowie die Untersuchung der eingesetzten Techniken durchgeführt.

Das vierte Kapitel umfasst den grundsätzlichen Aufbau des SWA Frameworks. Es werden die Architektur, die Funktionen der einzelnen Komponenten und deren Zusammenspiel erläutert.

Im fünften Kapitel wird die Realisierung beschrieben. Es werden im Kapitel Design entwickelte Konzepte und Überlegungen aus der Analyse implementiert. Ausserdem wird ein Test des Frameworks anhand der Entwicklung einer darauf basierenden Webapplikation durchgeführt.

Am Ende werden die Schlussfolgerungen der Arbeit kurz dargestellt.

Kapitel 2

2. Grundlagen

Im folgenden Kapitel werden die theoretischen Grundlagen dieser Arbeit erläutert. Einerseits wird das Model View Controller (MVC)-Entwurfsmuster vorgestellt, welches den Mittelpunkt der Arbeit darstellt. Andererseits wird die Dependency Injection als eine spezielle Anwendung der Paradigma Inversion of Control (IoC) veranschaulicht.

2.1. WEB 2.0

In diesem Abschnitt werden der Begriff und die Eigenschaften von Web 2.0 erläutert. Es wird sowohl auf das Thema Social Software eingegangen, als auch einzelne gängige Web-Anwendungen und deren Funktionen beschrieben. Das Verständnis von WEB 2.0 ist insofern von Bedeutung für die vorliegende Arbeit, als die WEB 2.0 Anwendungen ein Einsatzgebiet für den zu entwickelnden Framework darstellen.

2.1.1. WEB 2.0 versus Web

In der Fachliteratur und unter fortschrittlichen Internetnutzern wird der Begriff WEB 2.0 unterschiedlich ausgelegt. Häufig wird darunter eine Webanwendung verstanden, die durch eine hohe Benutzerfreundlichkeit gekennzeichnet ist. Manchmal wird hierbei auf Weblogs hingewiesen, die sich in letzter Zeit im Internet blitzartig vermehren. Andere sind der Meinung, dass Web 2.0 bloß eine geschickte Neuvermarktung der alten Internetdienste wäre. WEB 2.0 bezeichnet keine spezielle Technik, sondern mehr das Zusammenwirken verschiedener Methoden und Werkzeuge sowie eine vermutete soziale und wirtschaftliche Entwicklung.

Der Begriff Web 2.0 ist im Rahmen der Vorbereitung auf eine Online-Konferenz von O`Really-Verlag und MediaLive im Herbst 2004 entstanden. Die Begriffsfindung wird in der Litaretur Dale Doughety (O`Realy-Verlag) und Graig Cline (MediaLive) (Wikipedia),

aber auch Tim O`Really zugeschrieben [Richter, 2007]. Auf der Web-Seite des O`Really Verlags wird WEB 2.0 als eine (Business-) Revolution in der IT-Branche beschrieben, die aufgrund der Weiterentwicklung des Internets auf eine Plattform hin ausgelöst wurde. Es ist auch ein Versuch, die Regeln des Erfolgs dieser Plattformen zu verstehen. Die wichtigste Regel ist allerdings, Anwendungen zu entwickeln, mit denen Netzeffekte erzeugt und genutzt werden können, wobei deren Wert umso größer wird, je mehr Leute diese verwenden.[3]

Im Zeitalter Web 1.0 dominierten die statischen Webseiten das Internet. Die Rolle des Internetbenutzers beschränkte sich auf die des Konsumenten, während diese sich in letzter Zeit immer mehr auf die des Produzenten hin verschiebt. Unter Web-2.0 wird der Nutzer aktiv in die Gestaltung der Web-Inhalte einbezogen. Die Intensivierung der sozialen Verbindungen, die die Web 2.0-Dienste den Internet-Anwendern ermöglichen, sind ausschlaggebend für den Erfolg der neuen Technikdienste. Neu am Web 2.0 sind die Techniken (wie Ajax, RSS, Web Services u.a.), die eine bessere Benutzerfreundlichkeit wegen der verbesserten Integration der Schnittstellen gewährleisten. Somit ist für den Benutzer der Aufwand, eigene Inhalte, Fotos, Videos auf den Webseiten zu veröffentlichen und die Beiträge anderer zu kommentieren oder zu voten, auf das Minimale gesunken. Viele Web 2.0 Anwendungen verfügen über die Dienste (API – Application Programming Interface), mit denen verschiedene Anwendungen miteinander kombiniert werden. So z. B. hat man in YouTube[4] die Möglichkeit gleichzeitig eigene Beiträge in Facebook[5] zu veröffentlichen.

2.1.2. Social Software

Der Erfolg der neuen Webdienste ist im Wesentlichen auf die aktive Beteiligung und die sozialen Interaktionen auf den Plattformen zurückzuführen. Daher werden diese Dienste Social Software genannt. Diese Bezeichnung hat sich gegen u.a. Social Networking Dienste und Soziale Kommunikationsplattformen durchgesetzt. Sixtus definierte unter Sociale Software “Softwaresysteme, welche menschliche Kommunikation, Interaktion und Zusammenarbeit unterstützen” [Sixtus 2005]. Schmidt strukturiert in seiner Definition die Basisfunktionen von Social Software (SSW): “Social Software sind solche internetbasierten Anwendungen, die Informations-, Identitäts- und Beziehungsmanagement in den (Teil-) Öffentlichkeiten hypertextueler und sozialer

[3] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[4] YouTube ist eine Community Projekt mit dem Schwerpunkt Veröffentlichung der Video http://www.youtube.com/

[5] Facebook ist eine Community Webseite. Siehe unter http://www.facebook.com/

Netzwerke unterstützen" [Schmidt 2006]. Das Informationsmanagement umfasst folgende Funktionen: Finden, Bewerten und Verwalten von online verfügbaren Informationen mit Hilfe von Social Software. Identitätsmanagement wird von anderen Autoren (z.B. Richter, 2007) als zentrale Funktion angesehen. Sie gibt dem Nutzer die Möglichkeit (z.B. in Form eines Profils) sich selbst darzustellen und somit einer breiten Masse bewußt persönliche Daten preiszugeben. Beziehungsmanagement ist eine wichtige Funktion der Sociale Software, die eine Verknüpfung, Pflege und Präsentation intermenschlicher Kontakte mittels Webdienste unterstützt. Mit Social Software wird der Wandel des Web 2.0 vorangetrieben, da die Nutzer jetzt ihre Anforderungen den Entwicklern diktieren.

2.1.3. Das Beta-Konzept

Traditionelle Software – Konzepte trennten die Entwicklung und die Struktur ihrer Dienstleistung von den Nutzern ab. Es wurden fertige Produkte erstellt und dem Benutzer zur Anwendung bereitgestellt. Dagegen zeichnen sich Web 2.0 Anwendungen durch ihre Unfertigkeit und Vorläufigkeit aus. Sie werden erst durch eine wachsende Community (durch die Erreichung der kritische Masse[6]) für den Nutzer aber auch für den Betreiber attraktiv. Die Webseite Flickr war zuerst als ein Online - Rollenspiel geplant.[7] Als zusätzliches Parameter bot es sich an, Bilder online zu stellen. Diese Funktion – Laden der Bilder – gewann jedoch ein größeres Interesse der Nutzer als die Grundleistung (online-spiel). Daraufhin reagierten die Entwickler mit einem Umbau der Webseite entsprechend den Vorstellungen der Konsumenten und gestalteten Flickr so wie er jetzt aussieht, allerdings immer noch im Beta – Status.

Die Geschichte von Flickr ist typisch für die Zeit der Web 2.0 Dienste. Ein Projekt wird unvollendet online gestellt und weiter im Bezug auf die Bedürfnisse der Benutzer, sowie zusammen mit den Benutzern, entwickelt (Sixtus,2005). Die Anbieter sparen dadurch an Entwicklungskosten, da die Benutzer gleichzeitig Co-Entwickler sind, indem sie an der Umgebung partizipieren.

2.1.4. WEB 2.0 Erscheinungsformen

Hier werden einige gängige Webdienste erläutert, die das Zustandekommen einer Online Community unterstützen und somit zum Bereich der Sozialen Software gehören.

---

[6] die Anzahl der Mitglieder, ab der die virtuelle Community aus sich heraus am Leben erhalten bleibt. http://www.lexikon-ebusiness.de/index.cgi?pg=community

[7] http://de.wikipedia.org/wiki/Flickr

2.1.4.1. Blogs

Ein Weblog, häufiger verwendet in der Kurzform Blog, beschreibt die Publikation des Online-Tagebuches auf einer Webseite. Das Wort setzt sich aus web und log (buch) zusammen. Die Tätigkeit der Veröffentlichung der Blogs wird als Bloggen bezeichnet.

Dieses Kommuniakationsmedium, zunächst Online-Tagebücher genannt, tauchte Mitte der 90-er Jahre als Webseiten auf, auf denen die Betreiber periodisch Einträge über ihr Leben öffentlich publiziert haben.

Die Einträge in Weblogs erfolgen in umgekehrt chronologoscher Weise, so dass das Aktuellste sich am Anfang befindet. Diese Redaktionssysteme sind einfach und schnell zu bedienen. Auf die Beiträge (Posts) des Autors (Bloggers) zum bestimmten Thema können die Leser des Blogs mit Kommentaren antworten. Dadurch entsteht eine interaktive Kommunikation. Häufig verweisen die Blogger gerne mittels Tags[8] auf andere Beiträge- auch von anderen Blog-Webseiten- und verstärken dadurch die Vernetzung der Nutzer. Der auf diese Weise entstehende Kommunikationsraum wird als Blogosphäre bezeichnet. [Sixtus 2006]

2.1.4.2. Wikis

Während die Blogs überwiegend dazu dienen, dass einzelne Internetnutzer ihre persönlichen Berichte schreiben bzw. Ihre eigene Meinung zu bestimmten Themen präsentieren, verfolgen Wikis ein anderes Ziel. Mit Hilfe eines Wiki-Programms soll eine Konsolidierung des kollektiven Wissens ermöglicht werden. Verschiedene Autoren können gemeinschaftlich an der Gestaltung der Inhalte der Seiten arbeiten. Auf diese Weise werden die Erfahrungen und das Wissen mehrerer Nutzer auf einer Seite zusammengefasst. Das bekannteste Beispiel für eine Wiki-Anwendung ist Wikipedia.

“Ein Wiki ist eine Software und Sammlung von Webseiten, die von den Benutzern nicht nur gelesen, sondern meist auch direkt online geändert werden können.” (Wikipedia, http://de.wikipedia.org/wiki/Wiki)

Der Begriff “wiki“ kommt aus dem hawainischen und bedeutet schnell. Das erste wiki wurde vom Software-Entwickler Ward Cunningham 1995 mit der Bezeichnung “wikiwikiweb” entwickelt.

---

[8] Hyperlinks oder Fotos werden als Tags markiert (tagging) d.h. als Schlagwort gekennzeichnet, damit diese leichter gefunden bzw. zugeordnet werden könnten. Z.B auf del.icio.us wird mit Hilfe von Tags (Oberbegriffe der beschriebenen Themen) auf Blog-Sammlungen zu diesen Themen verlinkt.

Ein wesentliches Merkmal von der Wiki-Software ist die Versionsverwaltung – im Fall von Fehlern oder Wandalismus kann die frühere Version der Seite wiederhergestellt werden.

Wikis sind Content-Management-Systeme (CMS), die überwiegend über offenen Zugriff zugänglich sind. Die Seiten können von registrierten Mitgliedern leicht geändert werden. Es bedarf dafür keiner HTML-Kenntnisse der Nutzer und keines zusätzlichen Computerprogramms, um spezielle Inhalte im Browser hinzuzufügen und zu verändern. Die technischen Anforderungen an die Programm-Anwender sind durch eine vereinfachte Syntax sehr gering.

Im Vergleich zu konventionellen Content-Management-Systemen unterscheiden sich Wiki-Anwendungen durch ein vereinfachtes Layout.

2.1.4.3. Internetforen

Ein Internetforum (lat. forum – Marktplatz), auch “Diskussionsforum” genannt, ist ein virtueller Platz zum Austausch und Archivieren von Gedanken, Meinungen und Erfahrungen (in Anlehnung an Wikipedia). Die Foren hat es wesentlich früher als die Wikis gegeben. Hier werden Postings (Diskussionsbeiträge) geschrieben, die gelesen und beantwortet werden können. Mehrere Beiträge zum selben Thema sowie Kommentare zu diesen Beiträgen von anderen Benutzern werden als Faden (Tread) oder Thema (Topic) gekennzeichnet. Jeder Benutzer kann einen neuen Tread eröffnen und damit ein neues Thema zur Diskussion stellen. Wenn die Teilnehmer von dem genannten Thema stark abweichen, kann der Site-Moderator solche Beiträge löschen oder redaktieren.

Es gibt unterschiedliche Formen von Internetforen[9]. An dieser Stelle wird auf die Webforen eingegangen. Beim **Webforum** handelt es sich um einen Teil der Webseite, der eine Registrierung der interessierten Benutzer vorsieht. Die Beiträge werden hier direkt auf der Webseite veröffentlicht. Die Nutzer können speziell bestimmte Treads abonnieren und somit gezielt an der Diskussion der anzusprechenden Themen aktuell informiert werden und sich beteiligen.

Es lassen sich zwei wesentliche mögliche Strukturierungen der Webforen unterscheiden. Bei den klassischen Foren werden die Beziehungen zwischen den Beiträgen innerhalb eines Themas *in Baumform hierarchisch* dargestellt, so das leicht erkennbar ist, welcher Beitrag als Kommentar auf welchen anderen Beitrag publiziert wurde. Insbesondere bei komplexen Themen erscheint das vielen Benutzern als hilfreich. Jeder Diskussionsbeitrag wird auf einer eigenen Seite geladen. Beim *Bulletin Board* dagegen werden alle Beiträge

[9] u.a. **Mailinglisten** -Systeme zum Verschicken der Nachrichten an Abonnenten z.B.mit der Info, dass zum bestimmten Thema ein Forum eröffnet wurde. **Usenet** -elektronisches Netzwerk, dass Diskussionsforen (Newsgroups) jeder Art bietet. (in Anlehnung an Wikipedia)

zum bestimmten Thema auf einer Seite chronologisch und unstrukturiert dargestellt. Wenn nicht alle Postings auf eine Seite passen, wird eine Folgeseite geöffnet. Diese Form ist verglichen mit der Baumstruktur wenig übersichtlich, aber die Abrufzeiten sind geringer. Das klassische Forum hat sich aufgrund der besseren Zuordnungsbarkeit der Beziehungen in der Baumstruktur mehr durchgesetzt. Auch zählen zu weitvebreiteten WEB 2.0-Diensten andere Anwendungen wie Bilder- und Videosharing Dienste, Socialbookmarking und andere, die als einzelne Plattformen oder innerhalb grosser Portale eingesetzt werden.

2.1.5. Beispiele der WEB 2.0 Portalen

An dieser Stelle werden zwei grosse Community Projekte analysiert. Insbesondere wird auf die Möglichkeiten und Funktionen der Communitiy-Portale eingegangen.

2.1.5.1. Flickr

Der Nutzen von Flickr[10] ergibt sich aus folgenden Funktionen – die Benutzer können ihre eigenen Bilder online stellen, diese einem Begriff zuordnen, kommentieren. Kommentare können auch zu Bildern anderer Flickr-Mitglieder gepostet werden. Die Nutzer können ihre Fotoalben erstellen oder in Gruppen gemeinsame Fotoalben führen. Die Bilder können über ein übliches Web-Interface hochgeladen werden oder per E-Mail und Handy online gestellt werden.

Im Focus der Flickr-Dienste steht nicht der reine Bilderdienst, sondern die Gruppengemeinschaft. Mittels des Schlagwortes (Tag) werden die Bilder kategorisiert und einer Sammlung zugeordnet. Durch Kommentieren der Bilder entstehen soziale Online-Interaktionen. Die Teilnehmer können einander per Kontaktlisten einfügen, sich offenen und geschlossenen Gruppen anschließen und Flickr-Mails versenden. Über das Tagging finden sich leicht Benutzer mit ähnlichen Interessen, da sie die Bilder dem gleichen Schlagwort zuordnen oder darunter suchen. So werden Kontakte geknüpft und gepflegt. Flickr ist eine typische Web 2.0 Anwendung, denn sie zeichnet sich durch eine hohe Partizipation der Community an der Plattform-Struktur und eine hohe soziale und inhaltliche Vernetzungsdichte aus.

[10] Flickr ist eine Community Seite, wird von der Yahoo Company vertrieben, http://flickr.com/

2.1.5.2. StudiVZ

StudiVZ[11] zählt in Deutschland 5,54 Mio. Nutzer und ist damit die am weitesten verbreitete Community Seite [Sandhöfer, 2008]. Hauptsächlich sind auf der Seite Studenten angemeldet, aber auch Abiturienten, Mitarbeiter der Hochschulen und andere Benutzer. Im Folgenden werden die Grundfunktionen von StudiVZ erläutert.

Für jedes Mitglied wird nach der Anmeldung eine eigene Startseite eingerichtet, in dem sein Profil angezeigt wird. Für die Anmeldung soll sich der Nutzer mit seinen persönlichen Daten (vor allem mit Namen) autorisieren. Auf der Startseite präsentiert sich der Benutzer anderen Mitgliedern. Er kann selbst bestimmen, wie viele Informationen er von sich preisgibt. Allgemeines, Persönliches, Fotos, Arbeit, Hochschule, besuchte Vorlesungen. Werden vom Benutzer die von ihm besuchten Hochschulveranstaltungen eingetragen, so werden diese auf seiner persönlichen Seite angezeigt. Klickt man auf diese, werden die anderen Mitglieder, die die gleiche Veranstaltung besuchen und dies gemeldet haben, angezeigt. In der Kategorie Kontakt kann der Nutzer seine Telefonnummer und Adresse eingeben. Der Kreis derjenigen, die die Kontaktdaten einsehen können, ist vom Nutzer festzulegen. Das können z.B. Freunde und Freunde der Freunde sein. Diejenigen, die nicht berechtigt sind persönliche Informationen über den Nutzer zu lesen, bekommen nur das Bild und den Namen des Mitglieds zu sehen.

Auf die Startseite setzen die Nutzer in der Regel mindestens ein Benutzerbild, meistens werden jedoch Fotoalben zu bestimmten Kategorien angelegt. Wenn auf dem Foto ein anderes Mitglied von StudiVZ abgebildet ist, kann man auf dessen Profil mittels Tag verlinken. Das andere Mitglied bekommt eine Meldung, bei wem und auf welchem Foto er angezeigt wird, und kann diesen Link auch entfernen.

Auf der Profilseite können die Besucher der Seite, auf der so genanten “Pinnwand”, kurze Nachrichten hinterlassen.

Wenn die Besucher auf dem Profil des Mitglieds Inhalte vorfinden, die rechtswidrig sind oder anderen schaden können, können sie diese Inhalte dem Betreuer melden. Diese werden dann später entfernt.

Auf der Startseite wird angezeigt, wer die eigene Seite besucht hat, allerdings nur, wenn die Besucher in ihren persönlichen Einstellungen angegeben haben, dass es angezeigt werden darf. Die Besucher der Seite können Nachrichten an das Mitglied senden, die dann im Postfach abgerufen werden können. Auf der Startseite wird zuerst gemeldet – “Du hast 1 neue Nachricht erhalten”. Es wird auch angezeigt, ob die angekommenen Nachrichten bereits gelesen und beantwortet wurden.

---

[11] StudiVZ ist eine Community Webseite, wird von der Firma studiVZ Ltd. vertrieben, http://www.studivz.net/

Ausserdem kann die “Gruschelfunktion” genutzt werden. Abgeleitet von grüßen und kuscheln, wird die Gruschelfunktion vor allem zur Kontaktanbahnung und Sympatiebekanntgabe genutzt. Auf der Startseite kommt dann eine Nachricht vom Anbieter – “Mitglied X hat Dich gegruschelt”.

Besucht man Profile anderer, kann die Funktion “Hinzufügen von Freunden” genutzt werden. Man lädt dann das andere Mitglied zum Kreis der Freunde ein. Wird diese Einladung vom anderen bestätigt, erscheint er auf der Profilseite der “Freundesliste”.

Zusätzliche Funktion von StudiVZ ist das Eröffnen und Beitreten bestimmter Gruppen nach besonderen Interessen. Auf der Profilseite erscheint auch, ob das Mitglied einer Gruppe angehört. Über die Suchfunktion kann mit Hilfe von Schlagwörtern gesucht werden, ob Gruppen zu bestimmten Themen bereits gegründet wurden. Bei manchen Gruppen müssen neue Gruppenmitglieder vom Ersteller der Gruppen zugelassen werden, bei anderen sind alle zugänglich. Die Gruppen präsentieren sich meistens außer mit dem Oberbegriff zusätzlich mit einem Foto und einer Kurzbeschreibung des Schwerpunktes der Gruppe.

Gerne wird bei StudiVZ auch die Suchfunktion benutzt. Hier kann man über den Namen, über die Hochschule oder z. B.über Gruppen nach Mitgliedern suchen.

2.2. Model View Controller (MVC) Pattern

Im vorigen Unterkapitel wurden Eigenschaften der WEB 2.0 Applikation vorgestellt. Diese Anwendungen sind durch Darstellung des Contents, der durch den Benutzer eingegeben wird, gekennzeichnet. Da diese Applikationen über umfangreiche Benutzeroberfläche verfügen, ist es sehr wichtig diese beiden Teile von einandern zu trennen. So werden die Applikationen besser und leichter gepflegt werden können.

Es handelt sich um Applikationen, bei denen die Eingabe und spätere Wiedergabe von Informationen oft durch ein MVC basiertes Framework oder durch eine Bibliothek unterstützt werden. Es gibt eine Reihe von solchen Frameworks: JSwing[12], Struts[13], JavaServer Faces (JSF)[14] u s.w.

---

[12] JSwing ist eine Java-API, die das Programmieren von grafischen Benutzeroberflächen unterstützt

[13] Struts ist ein Framework für die Entwicklung Servlet/JSP basierten Webapplikationen

[14] JSF ist auf JSP und Servlets basierte Framework zur Entwicklung von Benutzeroberflächen in Webapplikationen

Der Konzept MVC wurde von Trygve Reenskaug[15] entwickelt. Zum ersten Mal kam ihm die Idee während der Smalltalk-Entwicklung bei Xerox PARC im Jahre 1979. Das MVC-Muster wurde ursprünglich für die Entwicklung der Smalltalk-Benutzeroberflächen verwendet. Mittlerweile ist es als Standard für den Grobentwurf der meisten komplexen Softwaresysteme bekannt.

2.2.1. MVC Entwurfsmuster

Im Informatik-Zusammenhang steht der Begriff MVC für Model View Controller. Das Model View Controller Pattern – kurz MVC-Entwurfsmuster – wird häufig in den Applikationen eingesetzt.

Wie auf der Abbildung 2.1. dargestellt ist, gehört zum Kernziel des MVC die Entkopplung von Model (die verwaltende Komponente), View (die darstellende Komponente) und Controller (die steuernde Komponente) in der Software. Alle drei Bestandteile werden voneinander getrennt implementiert. Das bringt zum einen den Vorteil, die parallele Betrachtung identischer Daten unkompliziert in einer Anwendung umsetzen zu können. Zum anderen können die Fehler leichter entdeckt und behoben werden. Weiterhin gewährleistet das MVC-Model die Angleichung entsprechender Modifikationen im Modell und in den Views.

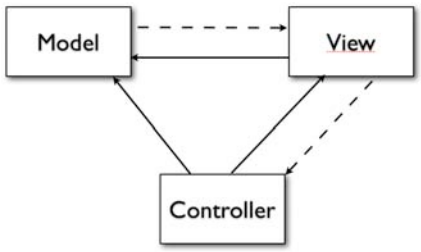

Abbildung 2.1. MVC Architekturmuster

Die verwaltende Komponente oder das **Modell** umfasst die Daten, die in der Anwendung vorkommen und die Logik, die für das Ändern des Modells zuständig ist. In Ausnahmefällen kann es aber auch die Teile der Gesamtlogik enthalten. Intern kann es zwar genauso wie die View und der Controller geändert werden, doch dürfen die Schnittstellen nicht variieren. Das Modell benachrichtigt über den Zustand der Daten, kümmert sich um dessen Verwaltung und die Modifikation. Da das Modell zudem noch mit View interagiert, reagiert es auf jegliche Änderung. Es handelt sich hierbei allerdings um

[15] Trygve Mikkjel Heyerdahl Reenskaug ist Norvegian Komputer „scientist“. Er gilt als Erfinder der MVC Pattern. s.Link. http://folk.uio.no/trygver/

eine einseitige Kommunikation, da die Views Informationen weder untereinander noch mit dem Modell austauschen, weshalb man hier von einem Beobachter-Muster spricht. Das Modell seinerseits aktualisiert fortwährend die Liste aktiver Views, die über jeden Wechsel im Modell informiert werden.

Die darstellende Komponente oder die **View** ist für die Repräsentation der Modelldaten zuständig und nimmt die Benutzereingaben entgegen. Eine View kann lediglich von einem Modell angesprochen werden. Umgekehrt allerdings kann ein Modell mit beliebig vielen Views interagieren. Sollte das Modell modifiziert werden, wird die Information ohne weiteres an die View weitergegeben. Diese erstellt daraufhin eine aktualisierte Abbildung des Modells. Eine effiziente Vorgehensweise lässt recht schnell eine View aktualisieren – eine neue wird anhand einer bereits vorhandenen View erstellt. Eine individuelle Note eines bestehenden Entwurfs bringt nötigenfalls angepasste Farben, Bilder und das Layout mit. Ein selbstständiges Update-Verhalten der View bleibt aber unabdingbar. Damit wird stets eine korrekte Darstellung des Modells gewährleistet. Doch sind die Möglichkeit einer View recht begrenzt, da sie weder die Stammdaten eines Benutzers noch die externen Informationen einsehen kann. Zudem kommuniziert die View mit dem Controller. Zwischen der View und dem Controller darf sich jedoch nur eine Basisschnittstelle befinden, ansonsten könnte ein Informationsaustausch nicht mehr gewährleistet werden.

Die steuernde Komponente oder der **Controller** ist im Grunde genommen ein Interaktions-Generator. Der Controller bestimmt, welches Modell für welche aktiven Views zuständig ist, da bei ihm alle Views gelistet sind und er den Standort des Modells kennt. Die steuernde Komponente verfügt über eine Schnittstelle zu der Übernahme und der Verarbeitung der Benutzereingaben der View. An dieser Stelle wird das Modell oft initialisiert und verändert. Um den reibungslosen Ablauf zu erreichen, muss der Controller sowohl über eine Schnittstelle zum Modell als auch über eine zur View verfügen. Die Informationen, die die steuernde Komponente benötigt, begrenzen jedoch den Umfang jeder Schnittstelle auf ein Minimum. Doch leider ist dies lediglich die utopische Theorie. In der Praxis ist der Controller auch zu Änderungen am Modell befugt.

2.2.2. Das MVC-Muster im WEB

Die Server-Klient Kommunikation läuft immer nach gleichem Prinzip: Der Klient schickt eine Anfrage an den Server ab und erhält eine Rückmeldung. Dabei ist diese Kommunikation auf eine Art und Weise eingeschränkt, denn nur wenn ein Request den Server erreicht, schickt dieser einen Response zurück. Das Bemerkenswerte ist: In den klassischen Webapplikationen kann der Webserver kein Request an den Klient schicken.

Aus dem Grund, dass der Webserver keine Kommunikation mit dem Klienten aufnehmen kann, kann der Server seinerseits keine Datenänderungen an den Klienten senden.

Deshalb ist das bereits im Abschnitt 2.2.1. erwähnte MVC-Beobachter-Muster hier nicht möglich.

In der Evalution der Webapplikationsentwicklung gibt es zwei unterschiedliche Architekturen. Model 1 und Model 2 MVC. Auf diese wird hier näher eingegangen.

Model 1

In der Architektur des Models 1, dargestellt in der Abbildung 2.2., ist allein die JSP-Webseite dafür verantwortlich, die eingehenden Anfragen des Klienten zu bearbeiten und anschließend die Antworten an ihn zu senden. Die Modell 1- Architektur wurde für anspruchslose Applikationen entwickelt und sollte vorzugsweise nicht für Software mit komplexen Anforderungen genutzt werden. Wahlloser Umgang mit dieser Architektur führt zu einer hohen Anzahl von *Scriptlets* (*Java Code)*, die in der JSP-Webseite eingebettet sind, besonders wenn dort eine immense Abfolge von Abfrage-Prozessen stattfindet.

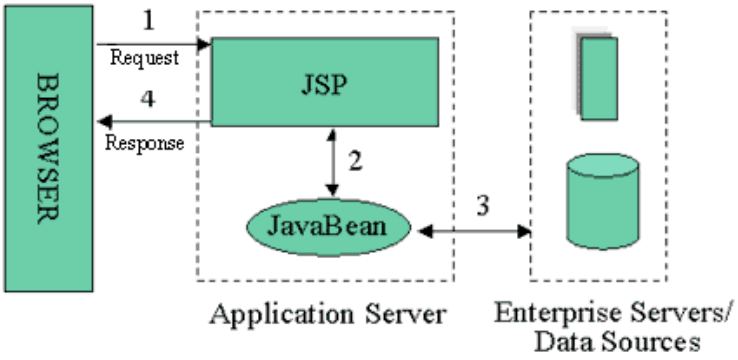

Abbildung 2.2. Model 1 Architektur mit JSP [aus Seschadri 1999]

Model 2

Wie in der Abbildung 2.3. zu sehen ist, verfügt die *Java-Architektur* über eine speziell entwickelte Komponente, die dafür geeignet ist, die Controller-Rolle zu übernehmen und dem Entwickler eine besser strukturierte Entwicklung der Applikationen zu gewährleisten.

Der wichtigste Unterschied zwischen Modell 1 und Modell 2 besteht darin, dass ein *Controller* die Klientenanfragen anstelle einer weiteren JSP-Webseite bearbeitet. Der *Controller* ist als *Servlet* implementiert. Die folgenden Schritte werden ausgeführt, wenn ein Klient eine Anfrage stellt:

– Der *Controller Servlet* bearbeitet die Klientenanfragen;

– Der Controller Servlet instantiiert adäquate *JavaBeans*, die auf den *Request-Parametern* basieren;

– *Der Controller Servlet* kommuniziert selbstständig oder mit Hilfe eines *Controller-Helfer*s durch eine Schnittstelle oder direkt mit der Datenbank, um die benötigten Daten anzupassen;

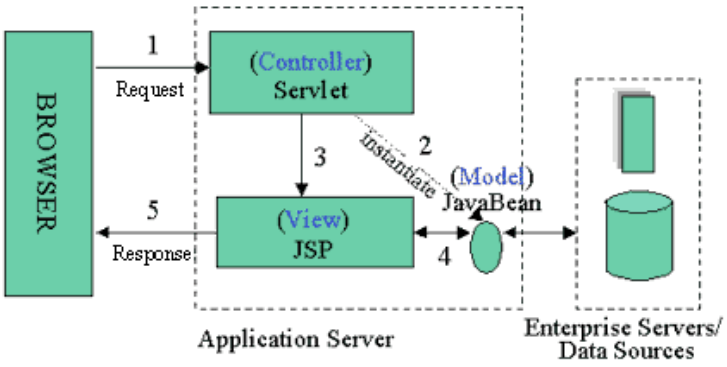

Abbildung 2.3. Model 2 Architektur mit Servlets [aus Seschadri 1999]

– Der *Controller* sortiert die sich ergebenden *JavaBeans* in eine von drei Kontexts (*Request, Session* und *Applikation*);

– Der *Controller* entsendet dann die Requests an die *View*;

– Die *View* verwendet die zuvor initialisierte *JavaBeans*, um die Daten anzuzeigen. Es ist zu beachten, dass in der JSP keine Darstellungslogik vorhanden ist. Die einzige Funktion der *JSP* in der Modell-2-Architektur besteht darin, die Daten der *JavaBeans* aus den *Request-, Session-* oder *Application-Scopes* anzuzeigen;

Erst hier wurde das *MVC-Muster* soweit möglich realisiert. Es sei an dieser Stelle erwähnt, dass man mit MVC eine unbegrenzte Anzahl von *Controller Servlets* in die Web-Applikation einbinden kann, tatsächlich kann man aber nur ein *Controller Servlet* pro Modul haben.

2.3. Inversion of Control (IoC)

Diese Paradigma wird im größten Teil der modernen Frameworks eingesetzt. IoC soll dabei behilflich sein, einige Arbeitsschritte während des Softwareentwicklungsprozesses zu vereinfachen. Da die Komponenten der MVC-Muster auch initialisiert und konfiguriert werden müssen, ist es von Bedeutung anzuschauen, wie die Konfiguration entkoppelt werden kann.

Das Prinzip der IoC Paradigma besteht in der Änderung des Kontrollflusses. Es wird auch als eine Anwendung des „Hollywood-Prinzips“: „don‘t call us, we‘ll call you“[16] bezeichnet. Dies gehört zu einem gewöhnlichen Programmierstil in der Benutzung des IoC basierten Frameworks.

Ein typisches Bespiel für einen Framework-Einsatz des IoC ist ein Java Swing. Hier werden die Listener geschrieben und dann als Callback beim Framework registriert. Nachdem ein Event auftritt, dem dieser Listener zugeordnet ist, wird der Kontrollfluss an den Listener übergeben. Am Ende der Listener-Ausführung wird der Kontrollfluss wieder zurück an das Framework übergeben.

2.3.1. Dependency Injection

Dependency Injection ist ein Entwurfsmuster, dessen Nutzen darin besteht, in einem objektorientierten System Abhängigkeiten zwischen den Komponenten oder einzelnen Objekten zu minimieren. Dies ist eine Anwendung der IoC Paradigma. Im Vergleich zum IoC beschränkt sich der Dependency Injection-Einsatz der IoC nur auf die Erzeugung und die Initialisierung von Objekten.

Es gibt drei Typen der Dependency Injection-Frameworks:

Interface Injection (Typ 1 IoC) – hier muss ein Interface entwickelt werden, um die Injection zu kennzeichnen. Der Klient muss dieses Interface implementieren. Somit wird das Setzen von Werten bekannt gemacht.

Setter Injection (Typ 2 IoC) – hier wird grundsätzlich durch eine Konvention vorgeschrieben, wie die Methoden für das Setzen benötigter Objekte geschrieben werden müssen.

Constructor Injection (Typ 3 IoC) – unterscheidet sich von den anderen Typen, und zwar durch das Setzen von sämtlichen abhängigen Objekten über einen Constructor – was oft zu einem unüberschaubarem Code führt, wenn an einen Constructor viele Parameter übergeben werden müssen. Dementsprechend kann ein solches Objekt nur erstellt werden, wenn all seine Parameter über den Constructor übergeben wurden.

[16] aus http://en.wikipedia.org/wiki/Inversion_of_control

2.3.2. Verwaltung von Gültigkeitsbereichen für die Objekte

Um steuern zu können, dass bei jeder Anfrage eines Objekts immer dasselbe Objekt geliefert oder jedes mal ein neues erzeugt werden muss, können die Dependency Injection Frameworks konfiguriert werden. Zum Teil bieten die Frameworks unter anderem die Konfiguration zur Verwaltung der Objekte kontextabhängig an.

Die Gültigkeitsbereiche haben unterschiedliche Größen und können auch andere Gültigkeitsbereiche mit einschließen. Ein Objekt darf nicht abhängig von einem anderen Objekt, das in einem der kleineren Gültigkeitsbereiche liegt, sein. Somit darf beispielsweise ein Objekt der Session Scope nicht von einem Objekt abhängig sein, das in Request Scope liegt.

Kapitel 3

3. Analyse

Im diesem Kapitel wird eine erste Analyse der Problemstellung durchgeführt. Es wird das Anwendungsgebiet analysiert und die Anforderungen an das System festgelegt.

3.1. Vorgehensmodel

Hier wird die Vorgehensweise bei der Framework-Erstellung beschrieben.

3.1.1. Ansätze für die Entwicklung eines Frameworks

Der ideale Weg zur Entwicklung eines Frameworks, wie es Ralph Johnson beschreibt, besteht aus drei wichtigen Hauptphasen [Johnson 1993], wie es in der Abbildung 3.1. dargestellt ist.

Abbildung 3.1. Idealer Weg zur Entwicklung eines Frameworks

– In der ersten Phase wird die Problematik der Domäne untersucht, wobei zunächst ihre abstrakte Sichtweise veranschaulicht werden muss. Des Weiteren werden einige beispielhafte Applikationen, die auf diesem Framework aufgebaut werden, gesammelt. Anschließend wird analysiert, ob alle Teile der herzustellenden Applikationen mit der Vision des Frameworks realisiert werden können.

– In der zweiten Phase wird das Design des Frameworks entwickelt. Dabei wird untersucht, ob es dem Aufbau der Beispielapplikationen entspricht.

– In der dritten Phase wird das Framework auf seine Eigenschaften, den Anforderungen gerecht zu werten, getestet. Der Einsatz des Frameworks wird entweder anhand der Implementierung der gesamten Applikation oder nur zum Teil getestet.

Da im Falle der Framework-Entwicklung nicht eine zu unterstützende Applikation sondern eine Domäne davon analysiert werden soll, was eine richtige Herausforderung ist, werden in der Praxis die Frameworks auf andere Weise entwickelt.

Relativ häufig wird in der Praxis folgender Weg, Frameworks zu entwickeln, angewandt:

– Es werden mindestens zwei zu entwickelnde Applikationen, analysiert.

– Es werden drei Gruppen gebildet. Die erste Gruppe ist für die Entwicklung des Frameworks verantwortlich. Andere zwei Gruppen machen nichts anderes als während der Entwicklung ständig zu versuchen, die zu unterstützenden Applikationen aufzubauen und somit das Framework zu testen.

Noch mehr Verbreitung hat in der Praxis ein weiterer Weg der Framework-Entwicklung bekommen. In dieser Variante wird zunächst ein mit Basisfunktionen ausgestattetes Framework und danach nacheinander einzelne Applikationen entwickelt. Dabei werden die fehlenden oder fehlerhaften Funktionen entdeckt. Das Framework wird erweitert, dieser Zyklus setzt sich immer weiter fort, wobei die Einfachheit vorrangig ist. Vorzuziehen ist bei der Entwicklung KISS (Keep It Simple) Prinzip.

3.1.2. Prototyping-orientiertes Vorgehensmodell

Prototyping an sich bezeichnet jede Methode, bei der Prototype erzeugt werden – genau wie in diesem Fall. Im Bereich der Softwareentwicklung ist es zusätzlich überaus wichtig, dass der Anwender bereits im Anfangsstadium ein Feedback dem Entwickler zukommen lässt. Eignet sich der Lösungsansatz – so wird am nächsten Prototyp gebastelt. Zu den Hauptzielen gehören unter Anderem das rasche Vorankommen und die Beseitigung der Kommunikationsbarrieren. Der Anwender wird quasi in die Entwicklung mit aufgenommen, was sich nicht nur auf die allgemeine Integration zwischen dem Anwender und dem Entwickler, sondern auch auf die nachträgliche Zeitersparnis auswirkt.

Die Beweggründe für die Auswahl dieses Modells werden im Folgenden veranschaulicht.

Die Prototyping-orientierte Entwicklungsmethode steht nicht im Gegensatz sondern als Ergänzung zum klassischen Phasenmodell. Die Analyse der Anforderungen und Spezifikationen können zeitlich parallel ablaufen, ebenso können der Entwurf, die

Implementierung und der Test ineinander verschmelzen. Die einzelnen Phasen sind somit nicht Abschnitte einer linearen Entwicklung, sondern werden als unabhängige Aktivitäten betrachtet, die wertvolle Ergebnisse liefern. Die unten angegebene Abbildung 3.2. zeigt die wesentlichen Schritte, die beim Prototyping wiederholt ablaufen.

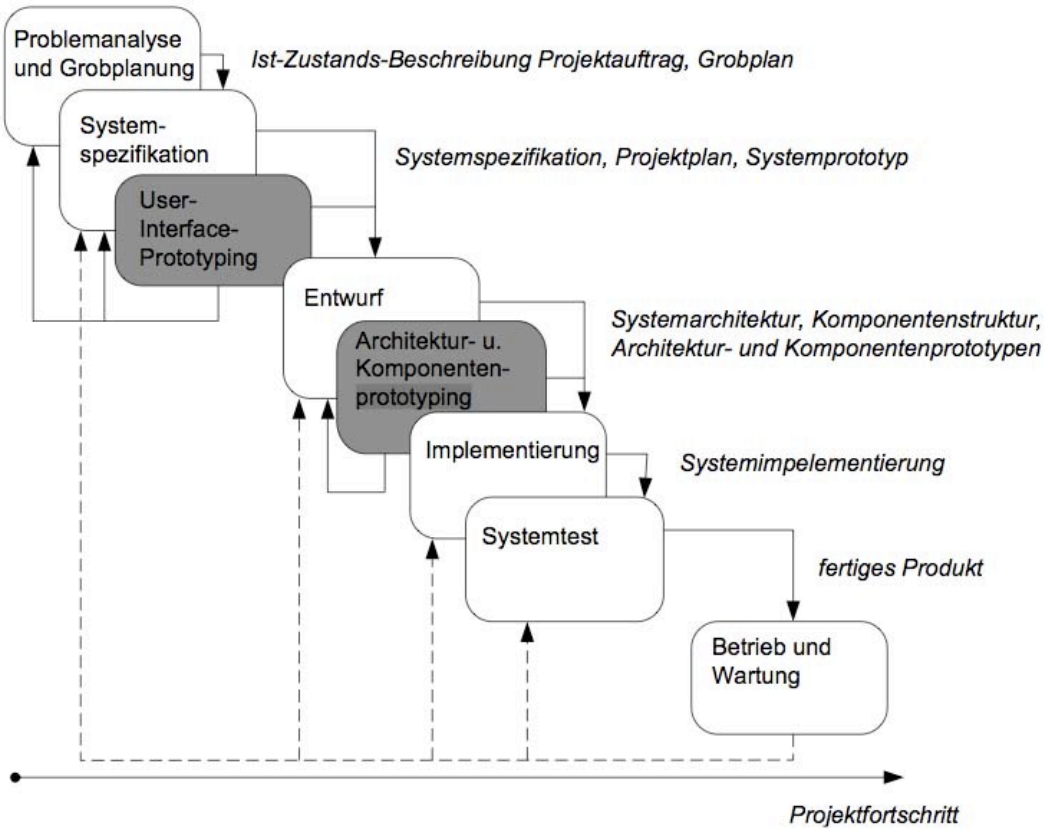

Abbildung 3.2. Prototyping-orientiertes Prozessmodel [Pomberger 2004]

Das Prototyping besteht aus dem häufigen Wiederholen signifikanter Prozesse bereits zu Beginn der Entwicklung, bis das Ergebnis eine gewisses Maß an Verwendbarkeit erreicht. Anhand der vorangegangenen Aktivitäten wird zunächst ein Entwurfs-Prototyp erstellt, nicht als eine bloße Beschreibung, sondern bereits funktionsfähig. Anschließend werden weitere Anforderungen und Funktionen spezifiziert und ein Prototyp der Benutzungsschnittstelle für die geplante Software entwickelt. Später wird unter realitätsnahen Einsatzbedingungen mit Hilfe der Anwender experimentiert und geprüft, inwiefern die Anforderungen der Nutzer erfüllt worden sind. Auf diesem Wege werden mögliche Systemfehler und der Änderungsbedarf identifiziert. Die Systemspezifikationen werden somit für die nächsten Phasen wesentlich detaillierter und kundennaher

ausgearbeitet. Ungewolltes Interagieren einzelner Bestandteile des Produkts untereinander kann frühzeitig erkannt und eliminiert werden.

Somit kann vor der Implementierung mittels Untersuchung des komplexen Architekturprototyps bereits sichergestellt werden, dass die Entscheidungen nicht fehl lagen. Die Akzeptanz des Produktes bei den Anwendern ist dadurch höher als bei klassischen Modellen. Die Kosten für die Anpassungen während der Implementierungsphase werden somit weitgehend gesenkt. Die Effizienz des Prototyping ist daher wesentlich höher als bei anderen Modellen: Der Produktionsstand ist einfacher zu erfassen, wodurch eine Qualitätssicherung schneller erfolgen kann. Bei der späteren Anwendung des Produkts wird der Pflege- und Update-Aufwand deutlich verringert.

Durch die Beteiligung der Anwender besteht aber auch die Gefahr, dass die Entwicklungskosten steigen, wenn die Ansprüche der Anwender mit jedem Prototyp stark wachsen und aufwendig realisiert werden können. Ausserdem besteht das Risiko, dass die für Prototype verwendeten weniger qualitativen Software-Technischen Mittel in die Produkt-Implementierung eingehen.

3.2. Usecases und Requirementsanalyse

An dieser Stelle werden die herauskristallisierten Anforderungen an das Framework besprochen. Die Anforderungen werden unterteilt in unterschiedliche Gruppen nach dem Relevante-Kriterium. Dabei wird größter Wert auf effiziente Arbeit mit diesem SWA Framework gelegt. Auf diesen Punkt wird nach der Beschreibung anderer Anforderungen eingegangen.

3.2.1. Zwingende Anforderungen

3.2.1.1. Erweiterbarkeit

Wie bereits erwähnt wurde, haben die modernen WEB 2.0-Anwendungen ständigen Bedarf an Weiterentwicklung. Dabei bestimmt der Anwender entweder explizit oder implizit die Weiterentwicklungsrichtung. Mit implizit ist gemeint, dass das Benutzerverhalten analysiert wird und dadurch die gewünschten Weiterentwicklungsrichtungen herauskristallisiert werden.

Für die Anwender der WEB 2.0 Portale ist es sehr wichtig, möglichst alle Anforderungen realisieren zu können, und zwar so, wie er sich das vorstellt, wobei manchmal die Bedeutung der Realisierbarkeitskomplexität in den Hintergrund gerückt wird. Aus diesem Grund soll es möglich sein das Framework auf die neuen Anforderungen ohne großen Aufwand anpassen zu können.

Das Framework soll eine maximal mögliche leichtgewichtige Architektur bieten, wobei, falls nötig, die projektbezogenen Erweiterungen erstellt werden können. Es soll möglich sein, die zu implementierenden Anforderungen nicht nur durch ein vorgeschriebenes Szenario zu realisieren. Bei Bedarf muss eine adäquatere Weise der Umsetzung bedacht werden.

3.2.1.2. Entkoppelung der Konfiguration

Die Devise muss sein: „Alles muss konfigurierbar sein“.

Das zu entwickelnde Framework soll für die darauf basierenden Applikationen eine modulare Architektur zur Verfügung stellen – so, dass die erstellten Module oder ihre Teile wieder verwendet werden können. Um den Raum des Einsatzgebietes einzelner Module und Komponenten zu verbreiten, ist es nötig, eine Möglichkeit zu haben, diese konfigurieren zu können.

Die Konfiguration soll über eine überschaubare und bequemere Weise als die Konfiguration in Struts oder JSF realisieren zu können. Für viele Softwareentwickler wird die in diesen Technologien verwendete Konfiguration-Methodik zur Hölle.

Andererseits muss die Konfiguration entkoppelt sein, da – wie schon betont wurde – dies ein sehr wichtiger Punkt für die Erweiterbarkeit des Systems ist.

Spring-MVC verfügt im Vergleich zu Struts, JSF und einigen anderen Frameworks über eine eigenartige Konfigurations-Methode in Form von Spring-Bean-Konfiguration. So eine Konfiguration ist bequemer. Das Problem der XML liegt im fehlenden Bezug von zu editierendem XML-Element zu damit konfigurierten Eigenschaft einer Klasse. In der XML-Spring-Konfiguration hat jeder Bean Bezug zu einer Klasse. Somit ist es gleich ersichtlich welche Parameter zur Verfügung stehen. Zu dem gibt es eine Reihe Tools, die so eine Konfiguration in Form von Modellierung unterstützen. Somit stellt ein gutes (genug detailliertes) Klassendiagramm praktisch ein komplettes Wissen zur Verfügung, was-wo-wie konfiguriert werden kann.

3.2.1.3. Trennung zwischen Model Vorbereitung und Model Änderung

In Webapplikationen können durch Benutzer initialisiertes Request drei Typen von Aktivitäten in der Lifecycle der Requestsabarbeitung durchgeführt werden: Erstens wird die Änderung des Models (möglicherweise auch in der Datenbank) ausgeführt, zweitens wird das Initialisieren oder Aktualisieren des Models (häufig mit den Daten aus der Datenbank) gemacht und drittens wird das vorbereitete Model mit Hilfe von Views dargestellt. Wobei das Vorbereiten des Models hundertprozentig von dem letzten Schritt abhängig ist. Je nachdem welche Teile des Models dargestellt werden sollen, müssen sie auch vorher vorbereitet werden.

Die Realisierung von Struts sieht das Trennen zwischen Änderung und Vorbereitung von der Model nicht vor. Hier gibt es so genannte Action, die diese beiden Aufgaben übernehmen sollen. Es kann in XML für bestimmte URL die Actions-Reihenfolge, die nacheinander ausgeführt werden muss und das anzuzeigende View konfiguriert werden.

In JSF sieht die Situation genau so aus. Der Unterschied ist nur in der Realisierung. Hier gibt es Commands, die für die Vorbereitung und Änderung des Models zuständig sind. Der JSF-Lifecycle sieht vor, dass vor dem Starten einer Action das vorher repräsentierte Model noch mal wiederhergestellt wird, was meistens ein Performance-Nachteil ist.

3.2.1.4. Portalbasierter Aufbau der Webapplikation

!!! Um Unklarheiten auszuschliessen, werden weiterhin die Views JSPs genannt.

Unter dem portalbasierten Aufbau wird verstanden, dass die gesamt angezeigten Seiten aus mehreren Teilen bestehen. Weiter werden die zusammengebauten Seiten (die üblicherweise <HTML> Tag enthalten) als *RootPerspektiven* bezeichnet, die Platzhalter, auf die Perspektiven geteilt werden, genannt *ViewAreas*, und die Teilseiten, die in diese ViewAreas geladen werden, gennant Views. Die Views können auch auf ViewAreas verteilt werden. Der zentrale Punkt hier ist, wie diese Seiten zusammengebaut werden.

Es gibt unterschiedliche Ansätze wie das realisiert werden kann.

In JSF und Struts können die JSP in andere JSP mit *include* Tag hinzugefügt werden. Auf diese Weise ist es möglich, statische Adressen den hinzugefügten JSP einzugeben. Wenn die Seiten jedoch dynamisch hinzugefügt werden sollen, in einem Fall soll Seite X hinzugefügt werden, in anderem – Y, ist es nicht so einfach realisierbar. Die beiden Frameworks legen Wert auf die Rückgabe der Action, entscheidend ist, wo der Request weitergeleitet werden soll.

Eine andere Technologie der Aufteilung der Perspektiven zu unterstützen ist Java Portlets Spezifikation[17]. Der größte Nachteil dieser Spezifikation und damit aller dieser Spezifikationen implementierter Frameworks ist, dass die einzelnen Portlets (Views) Zugriff nur auf die eigenen Portlet-Scope und Application-Scope haben. Dies bedeutet, wenn ein Portlet einige Informationen in seinen Skope aus der Datenbank geholt hat, können diese Informationen nicht in anderen Portlets verwendet werden. Deshalb müssen sie noch mal bei der Datenbank abgefragt werden. Dieser Nachteil hat auch seinen Grund, da das Ziel des Portlets ist – fachlich disjunkte Informationen anzuzeigen, so wie Integrieren dieser Portlets in andere Webportale. In beiden dieser Ziele besteht kein Bedarf für den Zugriff aus einem Portlet auf Scope anderer Portlets. Im Umfeld der WEB 2.0-Applikationen wäre diese Eigenschaft oft von grossem Nachteil.

3.2.1.5. Form-Komponente

Das Framework soll über einen unterstützenden Mechanismus verfügen, der es ermöglicht, eine Applikation zu erstellen, in der die Webseiten-Inhalte von dem Benutzer gepflegt werden können.

Das am häufigsten verwendete Element, um dem Benutzer eine Möglichkeit zu geben die Daten einzugeben, ist die Form. Daher gehört die Unterstützung der Form-Komponente zu den unabdingbaren Voraussetzungen für das Framework. Ausserdem muss die Konfiguration, die Anbindung der Initialisierung dieser Komponenten an den Controller Lifecycle abdecken.

Das Framework soll in die Form eingegebenen Daten vor weiteren Verarbeitungen auf die zulässigen Eingaben überprüfen können. Die Input-Kontrolle kann sich einfach auf das Prüfen von Types beschränken (Bsp.: Integer, String), aber auch die Erlaubnis von NULL-Werten verbieten oder mit bestimmten Mustern vergleichen. Es soll auch möglich sein, fallspezifische Überprüfungen zu implementieren und an das zu kontrollierende Element anzuschliessen, um beispielsweise den eingegebenen Wert des einen Input-Form-Elements im Gesamtkontext mit den Werten der anderen zu testen. Falls die Form nicht richtig gefüllt ist, muss der Benutzer auf die fehlerhaft eingegebenen Felder hingewiesen werden. Meistens ist es auch nützlich, dem Benutzer durch den Hinweis auf falsch gefüllte Felder eine Hilfestellung zu leisten.

Die meist verwendeten Form-Elemente sind zu unterstützen: Input Felder, ComboBox, TextArea und CheckBox.

Diese Requirements sind in Struts so wie auch JSF realisiert.

---

[17] Java Portlet Specification (JSR-168) Standart für Aufbau aus Komponenten kombinierbare Weboberfläche.

In **Struts** wird unterschieden zwischen einfacher Action und ActionForm. Die ActionForm ist eine Mischung aus einer Bean Klasses, wo die an die Form gemappte Daten gekapselt werden, und Logic – wo die Initialisierung von Eigenschaften dieses Beans in der Methode *reset()* realisiert wird. In der Methode *validate()* kann das Überprüfen von eingegebenen Werten in allen Feldern durchgeführt werden. Zudem ist es möglich, die Validatoren in einer externen XML-Datei zu konfigurieren.

In **JSF** gibt es keinen Unterschied zwischen den Commands, die für die Abarbeitung der Form-Eingaben und den Commands, die für andere Zwecke wie Vorbereiten und Ändern der Daten Model Zwecke verwendet werden. Die Validatoren in JSF werden als Tag-Elemente in JSP dem zu validierenden Feld zugeordnet wie auf dem Listing 3.1.

```
<h:inputText ...>
  <f:validateLongRange minimum="0" maximum="555"/>
</h:inputText>
```

Listing 3.1. Validieren von From-Elements

Die Philosophie dieser Frameworks ist das die Entwicklung der Form zu dem View gehört. Und daher ist komplett in JSP konfigurierbar.

In dem zu entwickelndem SWA Framework müssen ViewModel durch das Anhängen der Form sowie anderer Komponenten konfiguriert werden können. Es soll möglich sein, neue Komponenten zu entwickeln und leicht in das ViewModel integrieren zu können.

3.2.2. Wünschenswerte Anforderungen

Damit das Benutzerverhalten analysiert werden kann, müssen die Abfragen dauerhaft gespeichert werden. Es muss nicht Zählpixel[18] implementiert werden, es ist völlig ausreichend, wenn das Request vor seine Bearbeitung in der Datenbank gespeichert wird.

Unter Anderem muss das Framework eine Infrastruktur zur Verfügung stellen, die in einer WEB 2.0-Applikation die unentbehrlichen Funktionsaufgaben übernimmt.

Das Framework muss über folgende System-Funktionen verfügen:

---

[18] Zählpixel ist eine kleine Grafik die in Webseiten oder HTML-E-Mails verwendet wird, die Logdatei-Aufzeichnung für die spätere Analyse ermöglicht.

E-Mail-Versand – bietet den Versand der E-Mails an bestimmte Adressen an. Der E-Mail-Inhalt muss Templatebasiert und parametrisierbar erstellt werden können;

Periodisches Ausführen bestimmter Aufgaben – Diese Aufgaben werden in dieser Arbeit als Jobs bezeichnet. Dies kann sich beispielsweise um das in bestimmten Zeitintervallen erfolgende Aktualisieren einer Datenstruktur handeln, die nicht bei jedem Request neu initialisiert werden muss. Dadurch kann das unnötige Ausführen des SQL-Statements erspart werden.

Internationalisierung – ist eine sehr wichtige Anforderung für alle WEB 2.0-Applikationen. Nach einem gewissen Erfolg werden viele Web-Portale international angeboten und in Folge dessen in andere Sprachen übersetzt.

Bearbeiten von Bildern – Alle modernen Web 2.0-Applikationen bieten dem Benutzer die Gelegenheit, ein Avatar-Bild[19] hochzuladen. Dieses Bild wird häufig in verschiedenen Grössen angezeigt. Da der Browser das Bild mit der Folge einer nicht zufrieden stellenden Qualität skaliert, soll im Framework eine dafür programmierte Bildskalierungsfunktion vorgesehen sein. Die Veränderung der Bildgröße soll durch das proportionale Einhalten dieser erreicht werden.

Dem Entwickler soll darüber hinaus eine Bildausschnittfunktion zur Verfügung stehen.

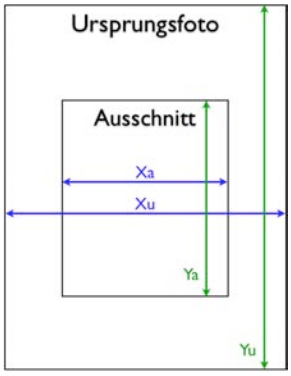

Abbildung 3.3. Schema des Bild Ausschnittes

Beim Ausschneiden, wie auf der Abbildung 3.3. dargestellt ist, wird aus der Mitte eines Bildes ein quadratischer Ausschnitt erzeugt. Es muss genau konfigurierbar sein, wie die Breite und die Höhe zustande kommen. Dabei entspricht in unserem Beispiel die Ursprungsbreite – *Xu* und die Ursprungshöhe – *Yu*. Die Ergebnis-Abmessungen sind dementsprechend *Xa* und *Ya*. Das Seitenverhältnis des Ausschnitts kann von dem des Originals abweichen. Die Grenzwerte dieser

[19] Unter dem Avatar-Bild versteht man einen grafischen Stellvertreter des Benutzers.

Abweichung müssen genau konfigurierbar sein. Das heißt, dass auch die Ausschnitte aus extremen Formaten (zum Beispiel ein sehr hoch- oder querformatiges Bild) erkennbar dargestellt werden können.

Rechte- System – Das System muss zunächst vereinfacht realisiert werden. Es soll nur geprüft werden, ob der Benutzer angemeldet ist. Dem nicht angemeldeten Benutzer wird der Zugriff auf bestimmte Ressourcen verweigert, daraufhin wird er auf eine dafür vorgesehene Webseite weitergeleitet. Die Überprüfung der Rechte in weiteren komplizierteren Fällen – dazu gehören beispielsweise das Löschen eines bestimmten Fotos und das Leisten eines Beitrags – muss fallspezifisch realisiert werden.

File Manager – Zur Aufgabe des Managers gehört das Verwalten von Dateien, die gespeichert werden müssen. Sie können Texte und Bilder enthalten. Es muss möglich sein, die Struktur des Filesystems konfigurieren zu können, sodass der File Manager unter den bestimmten Verzeichnissen die entsprechenden gleichartigen Dateien speichert.

3.2.3. Optionale Anforderungen

Die Skalierbarkeit bezeichnet das Verhalten einer Applikation im Bezug auf den Ressourcenbedarf bei wachsender Request-Anzahl. Eine „gut skalierbare“ Software kommt bei doppelter Leistung mit etwa doppeltem Resourcenbedarf aus. Um einen optimalen Systemzustand zu erreichen, kann man auf zwei unterschiedliche Möglichkeiten zugreifen: die horizontale oder die vertikale Skalierbarkeit.

Horizontale Skalierbarkeit – bezeichnet die Möglichkeit, bestimmte Funktionen auf zusätzliche Server zu verteilen;

Vertikale Skalierbarkeit – bezeichnet eine Verbesserung der Systemperformance durch den Einsatz leistungsfähiger Hardware. Zum Beispiel durch das Einfügen weiterer Prozessoren.

An der zu entwickelnden Framework wird keine Anforderung für die Unterstützung der horizontalen Skalierbarkeit gestellt, doch es wird eine Vision für das Skalieren der Webapplikation basierend auf dem SWA-Framework gemacht.

Auf der Abbildung 3.4. ist eine Verteilungsmöglichkeit der Applikation mit dem Einsatz des Server Load Balancing[20] dargestellt.

[20] Server Load Balancing (SLB) beschreibt die Wege, wie eine Last auf mehrere getrennte Server im Netzwerk verteilt werden kann.

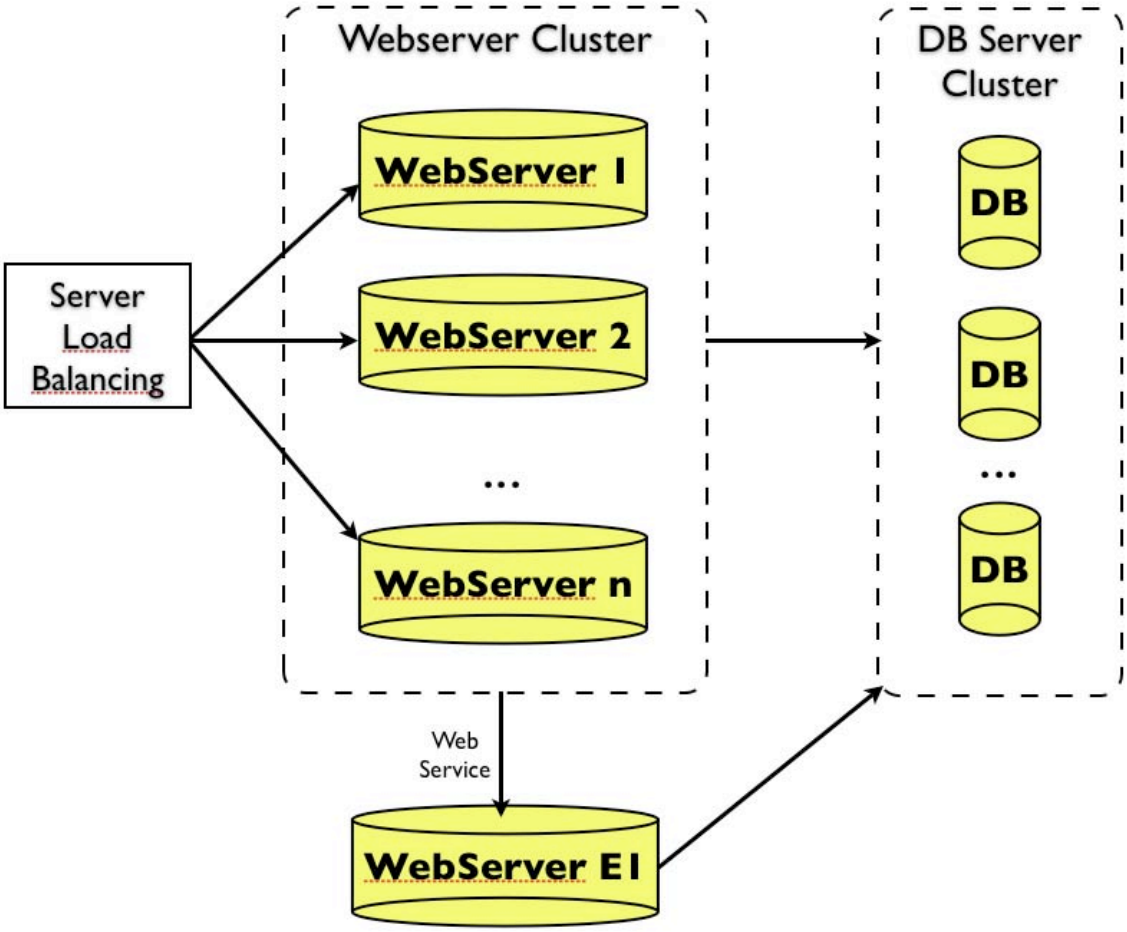

Abbildung 3.4. Vision der Skalierung

Der Server Load Balancing erlaubt, auf mehreren Exemplaren von Webservern (WebServer 1 bis n) mit der darauf installierten Webapplikation die Benutzeranfragen nach IP zu verteilen. Diese Webserver greifen auf eine verteilte Datenbank zu. Es ist an dieser Stelle Folgendes vorstellbar: es sollte möglich sein, so viele Webserver für die Abarbeitung der Klientenabfragen zur Verfügung stellen zu können, dass keine Mängel an der Performance vorhanden sein müssten. Es ist möglich die Objekte, die vom Applikationskontext abhängig sind und die dabei für die Zugriffe synchronisiert werden müssen, auf einen getrennten Webserver (auf der Abbildung WebServer E1) auszulagern und über die Web-Services den Zustand abzufragen und zu ändern.

3.3. Status Quo

Es wurde entschieden, eine eigene MVC Architektur zu entwickeln. Die Gründe, die dazu bewogen haben, sind erstmal der Wunsch über ein Framework zu verfügen, bei dem der Entwickler über komplettes Guru-Wissen zu diesem Framework verfügt, und schon allein dadurch die Weiterentwicklung des Frameworks in der angestrebten Richtung leicht realisiert werden kann. Voraussetzung ist, die Architektur muss gut überlegt sein. Andere wichtige Gründe sind, da kein mir bekanntes Framework die gestellten zwingenden Anforderungen realisiert hat. Als Hauptmerkmale dieses Frameworks sind hohe Abstraktion des Models, bequeme Handhabung, hohe Flexibilität und modular aufgebaute Architektur.

3.4. Technische Analyse

In diesem Unterkapitel werden die Techniken vorgestellt und untersucht, die den gestellten Anforderungen gerecht werden können und dafür geeignet sind. Da es festgelegt wurde den Apache Tomcat[21], Version 6, als Webserver einzusetzen, werden alle Beispiele an diesen Webserver angepasst. Die Präsentation des Apache Tomcat ist kein Gegenstand dieser Arbeit, deswegen wird darauf nicht näher eingegangen.

3.4.1. Eignung der Java Technologien für das Framework

Seit 1997 wird von Sun Java Technologie „Servlets“ und damit in Grundlagen erläuterte Model 2 Architektur angeboten. Im Mittelpunkt steht ein Servlet Container in dem alle Servlets ausgeführt werden. Gegenüber einigen anderen serverseitigen Technologien müssen einzelne Servlets bei jeder Anfrage nicht neu geladen werden.

3.4.1.1. Servlets

Servlet ist eine Java Klasse. Diese Komponente ist besonders gut zur Übernahme des Kontrollflusses zur Bearbeitung der Anfrage geeignet. Damit lassen sich die logischen Teile ausführen. Hier kann jede beliebige Anwendungslogik ausgeführt werden.

---

[21] Apache Tomcat ist ein Webserver, http://tomcat.apache.org/

Der spezielle Filter-Servlet kann so konfiguriert werden, dass er als eigenständiger Bestandteil einer Anfragen-Bearbeitung vorgeschaltet werden kann, und damit sowohl die Anfragen des Browsers als auch die Antworten verändern werden können. Außerdem können solche Filter die Anfragen auf andere nicht mit dem Request angesprochene Ressourcen umleiten.

Des Weiteren ist es möglich, die Filter in eine Kette einzuschliessen. Die große Bedeutung dessen liegt darin, dass vor der eigentlichen Bearbeitung des Requests der Kontrollfluss über eine Filterkette durchgehen muss. Diese Technologie bietet eine hervorragende Gelegenheit für die Realisierung des Front Controllers in einer Applikation an. Der Controller kann auf Module verteilt werden, die als Filter realisiert werden. Eine weitere nützliche Eigenschaft der Filter stellt die Änderung des Kontrollflusses, der in der Filter-Kette bestimmt ist, dar.

```
<!-- Filter Declaration -->
<filter>
  <filter-name>RequestFilter</filter-name>
  <filter-class>
                  com.swa.frontcontroller.RequestFilter
           </filter-class>
</filter>

<!-- Filter Mappen -->
<filter-mapping>
  <filter-name>RequestFilter</filter-name>
  <url-pattern>/app/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Listing 3.2. Anbinden von Filter-Servlet an Servlet Container

Das Listing 3.2. zeigt den Anschluss eines Filter-Servlet an den Servlet Container. Zuerst wird Klasse *RequestFilter* als ein Filter deklariert. Danach ist es möglich mit dem Parameter *url-pattern* zu konfigurieren, bei welchen Zugriffsressourcen der *RequestFilter* die Kontrolle übernehmen soll. Im dargestellten Bespiel wird der Filter für alle angesprochenen Ressourcen, die mit http://hostname/app_name/app/ anfangen, vorgeschaltet.

3.4.1.2. JavaServer Pages (JSP)

JSP Technik basiert auf JHTML[22]. Bei den Servlets müssen die Ausgaben, die im Response Stream geschrieben werden, über *out.println(„Hier folgt Inhalt der Ausgabe“)* realisiert werden, was zu einem sehr unübersichtlichen Code führt. JSP muss diesen Nachteil beseitigen.

Eine JSP wird wie ein HTML-Dokument aufgebaut. In diesem Dokument können die in Java programmierten Anweisungen, mit Hilfe der so genannten Skriptlets, eingebettet werden. Bei der Anfrage einer JSP übersetzt die JSP-Engine diese JSP in einen Java-Servlet.

Das Bemerkenswerte daran ist, dass die Übersetzung nur dann gemacht wird, wenn die JSP neu erstellt oder geändert wurde. Diese Eigenschaft von JSP ist sehr nützlich beim Testen einer Applikation, da die Applikation nicht neu gestartet und installiert werden muss. Dies erlaubt die JSPs gleich nach der Bearbeitung ohne den Server-Restart zu testen.

3.4.1.3. JavaBeans

JavaBeans ist eine Komponente, die der JavaBeans API Specification entspricht. Diese Komponenten werden oft als ein Daten Container verwendet, um die Daten in einer einfachen handhabbaren Form von einem Programmteil zum anderen zu transportieren. Alle JavaBeans Klassen müssen allgemeingütigen Eigenschaften erfüllen. Diese Klassen müssen:

– über einen Standard Constructor verfügen;

– öffentliche Zugriffsschnittstellen auf die gekapselten Eigenschaften nach get-/set-Konvention besitzen.

3.4.1.4. Tag-Library

Servlets und JSPs bieten eine Chance zur Verkürzung des Java-Codes beim Entwickeln einer Applikation. Leider sind die JSPs nicht völlig vom Java Code befreit und jedes Stück Code in diesen JSPs bringt Unübersichtlichkeit mit sich, was die Wartung und die Weiterentwicklung des Systems erheblich erschwert.

---

[22] Java HTML (JHTML) ermöglicht die mit Java entwickelte Software in Webseiten einzubinden

Tag-Libraries verschaffen einen Weg, die JSP-Seiten auf eine solche Weise zu entwickeln, dass 4gar keine oder nur noch sehr wenige in Skriptlets eingebundene Java-Anweisungen vorhanden sind. Somit ist es auch möglich, die JSPs von einem Designer bearbeiten zu lassen, der was Java angeht, ein Laie ist.

Den Aufbau einer Tag-Library kann man folgendermaßen beschreiben: Sie besteht aus der Tag-Library-Description (TLD) und den Tag-Klassen. Die Tag Klassen haben Zugriff auf den Kontext der JSP, wo der betreffende Tag aufgerufen ist. Somit ist es möglich, auch auf alle Bestandteile eines Request zuzugreifen. So implementierte Tags können in XML-Notation in die JSP-Seite eingebunden werden.

Die JSP bringt JavaServer Pages Standard Tag Library (JSTL) mit. Diese Library ermöglicht eine Realisierung folgender Funktionen in der XML-Notation:

– die Schleifenbildung;

– die Internationalisierung und Formatierung;

– direkter Zugriff auf Datenbankverarbeitung.

3.4.1.5. Resource-Dateien

Resource-Dateien sind eine weitverbreitete Methode Internationalisierung durchzuführen. Dabei kann die Übersetzung der Wörter und Phrasen auf verschiedene Dateien verteilt werden. Wie aus dem Listing 3.3. ersichtlich ist, wird die strukturelle Aufteilung auf verschiedene zusammengehörende zu internationalisierende Phrasen und Wörter auf erster Ebene gemacht, dann muss jeweils ein *properties* File erzeugt werden. Danach wird der Inhalt dieser eben erzeugten File auf die Zielsprachen übersetzt und in anderen properties Dateien gespeichert.

Mit kleinen Buchstaben wie *de* oder *ru* wird das *Locale* der Sprache konfiguriert, dem entsprechend mit grossen – das Land der *Locale*.

```
allgemein.properties
allgemein_de_DE.properties
allgemein_ru_RU.properties
forum.properties
forum_de_DE.properties
forum_ru_RU.properties
```

Listing 3.3. Strukture von *properties*-Dateien

Somit wäre es allerdings nicht möglich und dazu noch nicht einmal bequem, auf diese Weise mehrzeilige Ausdrücke zu übersetzen. Dafür wäre es besser, auf JSP-File-Ebene Internationalisierung durchzuführen. So dass für bestimmte Views eine JSP-Seite pro Sprache verwendet wird. Das wäre sehr nützlich beispielsweise für die Realisierung von allgemeinen Geschäftsbedingungen.

3.4.2. Dependency Injection im Einsatz

Da die Konfiguration und Initialisierung ein unabdingbares Teil jeder modularen Software ist, wird hier genauer auf die unterstützende Implementierung dieser Entwurfsmuster eingegangen. Dies ermöglicht die Entwicklung der Softwareteile durch das Entkoppeln der Konfiguration, um eine bessere Wiederverwendung und Wartung zu gewährleisten.

3.4.2.1. Initialisierung und Benutzung

Dependency Injection hilft während der Entwicklung einer Software das Konfigurieren von Objekten und das Initialisieren der Verwendung dieser zu trennen.

In der Konfiguration wird die Verwaltung der Objekte definiert. Es wird ebenfalls festgelegt, wie die Abhängigkeiten zwischen den Objekten aufgelöst werden müssen. Mit der Abhängigkeitsauflösung ist das Referenzieren zwischen den einzelnen Objekten gemeint. Des Weiteren müssen die Objekte den Scopes zugeordnet werden. Daraus erschließt das Framework, ob das Objekt bei jedem Request neu konfiguriert werden soll oder in einem bestimmten Kontext – für alle Requests – die gleiche Instanz einer Klasse erreichbar sein soll.

Es gibt zwei Konfigurationsarten in den Dependency Injection Frameworks:

Programmierte Konfiguration – hier wird in einer Programmiersprache beschrieben, wie die Objekte parametrisiert sein müssen. Der größte Vorteil ist eine sehr einfache Benutzung und Type-Sicherheit;

Text Datei – dies ist eine alternative pflegeleichte Möglichkeit. In dieser Datei wird mit einer hierarchisch strukturierten Sprache beschrieben, welche Objekte verwaltet werden müssen und in welcher Verbindung sie zueinander stehen.

Der Zugriff auf die initialisierten Objekte wird über eine bestimmte im Framework zur Verfügung stehende Schnittstelle gewährleistet. Somit kann das Objekt referenziert und weiter verwendet werden.

3.4.2.2. Dependency Injection mit Spring

Spring Core ist die zentrale Componente des Spring Frameworks[23], das die Aufgabe des Dependency Injection Container übernimmt. Dabei implementiert die Spring Core den Type 2 und den Type 3[24] dieses Containers.

Die Zuordnung der Beans und die Art der Konfiguration wird in den XML Deskriptoren beschrieben. Wo dieser Deskriptor zu finden ist, wird in der Einstellungsdatei *web.xml* konfiguriert. Die ist auf dem Listing 3.4. vorgeführt.

```
<web-app>
.........
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>

...........
<web-app>
```

Listing 3.4. Deklaration der *contextConfigLocation*

Ein anderer Weg, die Beans konfigurierende Dateien zu laden, ist direkt in Java möglich. Im Listing 3.5. wird dies veranschaulicht.

```
      XmlBeanFactory bf = XmlBeanFactory(
                          new ClassPathResource("fileName.xml",
                                getClass()));
```

Listing 3.5. Konfiguration der BeanFactory in Java

Der XML Deskriptor zum Initialisieren der Beans kann auf verschiedene Dateien aufgeteilt und im deklarierten XML Deskriptor eingebunden werden. Im Beispiel (Listing 3.6.) ist dargestellt wie solche Dateien angeschlossen werden können.

---

[23] Spring Framework ist eine Sammlung voneinander entkoppelter Framework-Komponenten der amerikanischen Firma SpringSource, Inc. Siehe http://www.springframework.org/

[24] siehe Abschnitt 2.3.1 (Dependency Injection)

```
<beans xmlns="http://www.springframework.org/schema/beans"
...
      <import resource="forms.xml" />
      <import resource="actions.xml" />
      <import resource="perspectives.xml" />
...
<beans>
```

Listing 3.6. Importieren von XML Deskriptoren

Die einzelnen Beans müssen als Java Klassen implementiert und durch XML Deskriptor mit den Werten konfiguriert werden.

3.4.3. AJAX

Mit der Asynchronous JavaScript and XML (AJAX) ist eine Technologie gemeint, mit der der Aufbau der asynchronen Kommunikation zwischen dem Browser und dem Webserver unterstützt wird. Mit AJAX ist es möglich, in JavaSciprt die Abfragen zu erstellen und an den Webserver abzuschicken. Dabei kann die vom Webserver zurück erhaltene Antwort auf der Klientenseite evaluiert und bearbeitet werden. Dieses Vorgehen ermöglicht, Inhalte der Webseite dynamisch nachzuladen und zu modifizieren.

Zur AJAX Kernfunktionen gehören:

– neue Elemente im DOM-Baum hinzuzufügen;

– vorhandene Elemente zu verändern oder sie zu löschen.

Das Ganze wird, ohne die Seite neu zu laden, realisiert. Es ist auf dieselbe Weise sogar das Absenden der Formulareingaben und eine unmittelbare Präsentation des Response zu realisieren, möglich.

Der größte Nachteil bei der Verwendung von JavaScript als Mittel um AJAX Funktionen zu benutzen, ist die Browser-Inkompatibilität, was zu sehr grossem Aufwand führt. Katastrophal wird es, wenn dieser Aufwand manchmal nicht vorhersehbar und meistens schlecht einschätzbar ist. So eine Applikation muss viele Tests überstehen, da die Tests für die unterschiedlichen Browser mehrmals durchgeführt werden müssen.

Diese Lücke schließen eine Reihe von AJAX-Frameworks, die die Funktionsbibliotheken zur Verfügung stellen. Diese Frameworks erlauben den Cross-Browser-Einsatz ohne jeden einzelnen Browser testen zu müssen (zumindest war es das Ziel, was meistens auch erreicht wurde).

jQuery[25] ist eins von diesen Frameworks. jQuery bietet folgende komfortable Funktionen: die DOM-Navigation und die DOM-Manipulation. Es unterstützt die bequeme XPath Anotation und die CSS-Selectoren, um die DOM-Navigation gewährleisten zu können.

[25] jQuery ist einen Open Source JavaScript-Framework. Siehe http://jquery.com

Kapitel 4

4. Design

Dieses Kapitel beschreibt den grundsätzlichen Aufbau des Frameworks. Anhand der Softwarearchitektur werden die Anforderungen weiter verfeinert. Nach der Darstellung der Architektur wird auf die wichtigste Komponente genauer eingegangen.

4.1. Softwarearchitektur

Wie in der Abbildung 4.1. dargestellt, setzt sich das System aus folgenden Hauptkomponenten zusammen:

Abstract Modul Environment – beschreibt die Schnittstelle für die Implementierung der Business Logic, die an das Framework angeschlossen werden kann;

User Interface Library (UI Lib) – ist einer nach der JSP-Spezifikation entwickelte Tag Library [26];

Application Environment Manager – übernimmt die Kommunikation zu der Spring Core, um Zugriffe auf die mit XML als Bean konfigurierten Ressourcen zu gewährleisten;

Front Controller – nimmt die Klientanfrage entgegen, führt die notwendigen Operationen durch, um den Kontrollfluss nach der Konfiguration zu steuern und an bestimmte Ressourcen weiterzuleiten;

Resource and Infrastructure – stellt Schnittstellen zu den Systemfunktionen des Frameworks und Resourcen des Systems zur Verfügung;

[26] Siehe Abschnitt 3.4.1.4. (Tag Library)

View Model – enthält die vom Framework unterstützten Content-Komponenten;

Persistenzschicht – gewährleistet Input und Output von persistierbaren Objekten aus und in der Datenbank;

Implementated Environment – stellt dar, aus welchen Bestandteilen die Module zusammengestellt werden können, sowie deren Abhängigkeiten zu den Komponenten des Frameworks;

Data Model – entspricht nach dem Model 2 Entwurfsmuster der Modelkomponente, die von keinen anderen Komponenten abhängig ist. Das sind am häufigsten persistierbare Objekte.

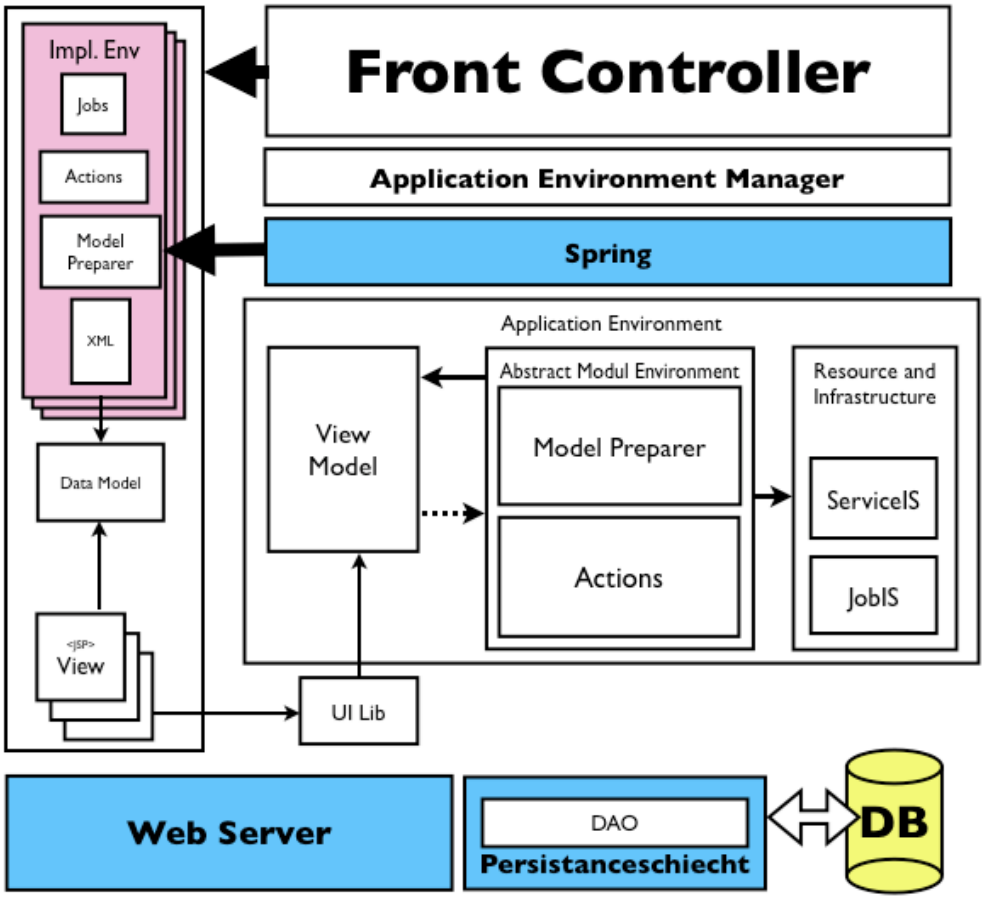

Abbildung 4.1. Framework-Softwarearchitektur

4.1.1. FrontController-Komponente

FrontController ist die Hauptkomponente des Frameworks. Hier wird die Abarbeitung des Request gestartet, anschließend wird der Kontrollfluss an die JSPs übergeben. Somit wird durch den JSPs ein Response generiert.

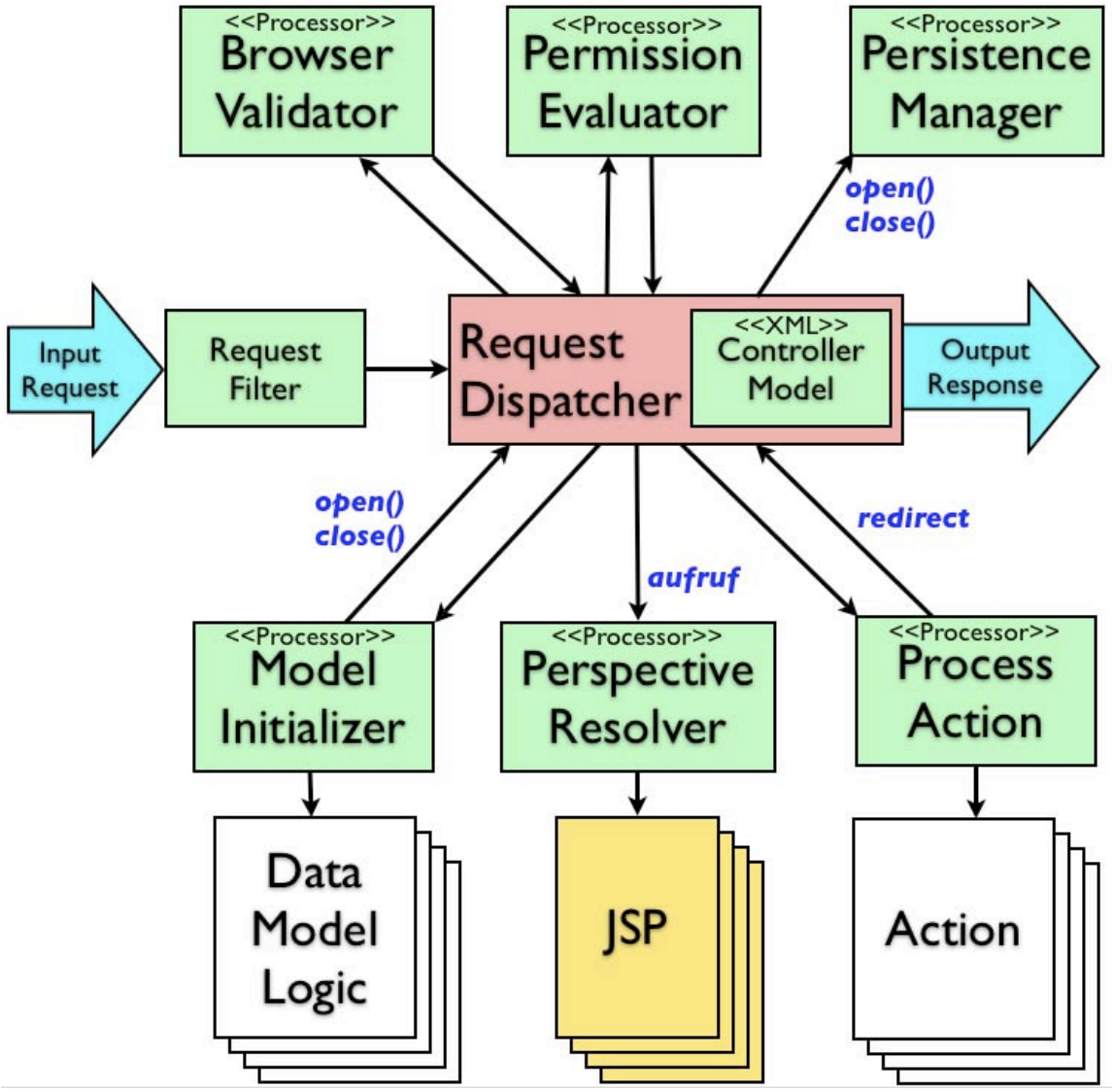

Abbildung 4.2. Lebenszyklus des FrontController

Wie auf der Abbildung 4.2. dargestellt, setzt sich der Controller aus einem *RequestFilter*, einem *RequestDispatcher* und mehreren Prozessoren zusammen. Der *RequestFilter* kümmert sich lediglich um das Abfangen der Klientanfragen und die Weitergabe des

Kontrollflusses an den *RequestDispatcher*. Weiter Unten wird auf den Lebenszyklus der Request-Abarbeitung genauer eingegangen.

4.1.1.1. Request Dispatcher

Alle Anfragen an die Applikation werden durch den *RequestDispatcher* gesteuert. Als Erstes werden hier die Requestparameter aus dem *HttpServletRequest* extrahiert und in einem *Hyperlink* Objekt strukturiert zusammengefasst.

Des Weiteren wird nacheinander die Kontrolle für weitere Bearbeitungsschritte an die Prozessoren übergeben. Welche Prozessoren und in welcher Reihenfolge die Kontrolle übergeben werden soll, ist im *ControllerModel* konfiguriert.

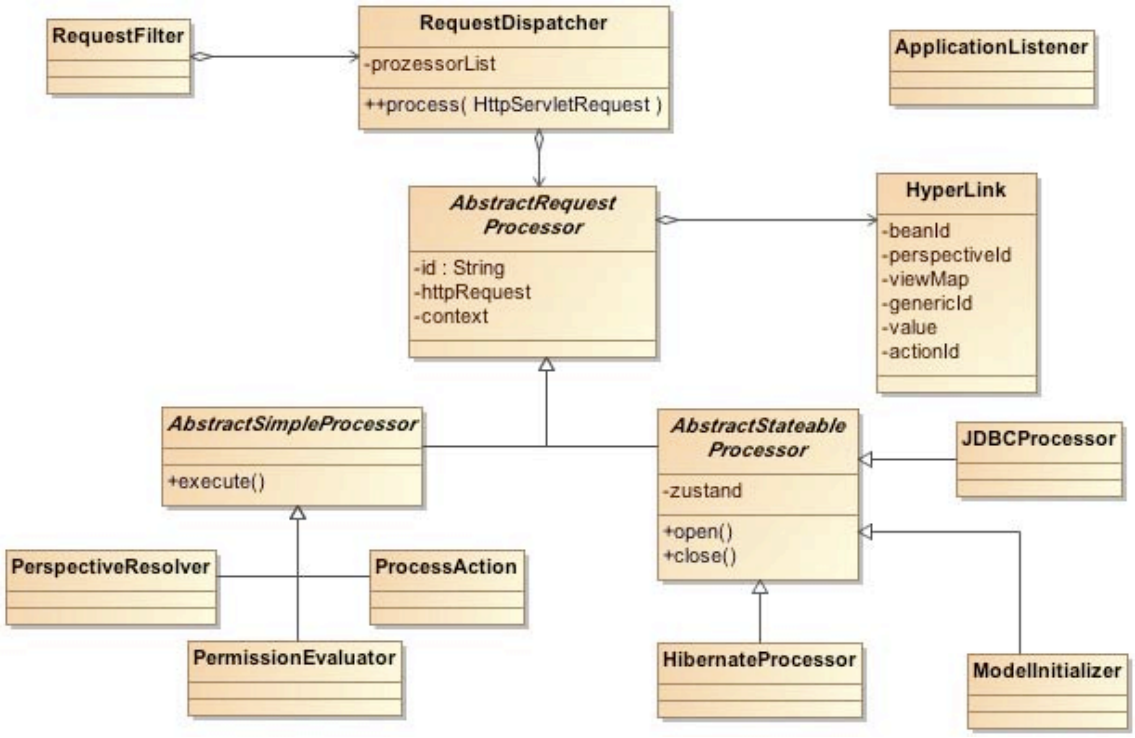

Abbildung 4.3. Klassendiagramm der *FrontController*-Komponente

Wie aus der Abbildung 4.3. ersichtlich ist, gibt es zwei Typen von unterstützten Prozessoren: *AbstractSimpleProcessor* und *AbstractStateableProcessor*. Der Unterschied besteht in der Art, wie der *RequestDispatcher* diese Prozessoren behandelt. Die *SimpleProcessor* werden nur durch *execute()* Methode aufgerufen, die *StateableProcessor* werden mit *open()* in den Zustand „geöffnet“ gesetzt, durch einen

Aufruf der Methode *close()* geschlossen und damit in den Zustand „geschlossen“ gesetzt. Nach seiner Ausführung liefert jeder Prozessor einen *AbstractResult* zurück. Es gibt zwei Typen von *AbstractResult*: *OKResult* und *RedirectResult*. Wie es bereits die Namen verraten, bedeuten sie:

OKResult – die nächsten Prozessoren in der Schlange können den Request weiter abarbeiten;

RedirectResult – bedeutet den *RequestDispatcher darauf hin*, dass weitere Bearbeitung des Requests abgebrochen werden soll. Stattdessen soll ein Redirect[27] an den Klienten zurück gesendet werden, die Zielseite für diesen Redirect steht in der *RedirectResult* Objekt. Falls während der Ausführung bei einem der Prozessoren ein Fehlverhalten aufgewiesen wird, wird an den Klienten ein Redirect mit dem Ziel einer Standart-Fehlerseite implementiert werden. Dieser Hyperlink zu dieser Fehlerseite kann im RequestDispatcher parametrisiert. Falls *RedirectResult* zurückgeliefert wird, werden die weiter in der Reihenfolge stehenden Prozessoren ignoriert und die Bearbeitung abgebrochen. Die Prozessoren, die in den Zustand „opened“ gesetzt wurden, werden geschlossen.

FrontController – ist so konzipiert, dass damit auch sehr spezielle Anforderungen der Applikation durch die Konfiguration des *RequestDispatcher* und die Erweiterung des *AbstractProcessor* realisiert werden können. Die Konfiguration des *RequestDispatcher* ist in XML durch die Spring-Beans vorgesehen.

Im Folgenden werden für die üblichen Aufgaben das Konzipieren von Prozessoren vorgeführt und der damit vorgesehene Lebenszyklus, der durch den Einsatz dieser Prozessoren realisiert wird, vorgestellt.

4.1.1.2. BrowserValidator

Die Aufgabe von diesem Prozessor ist, ein Request zu evaluieren. Wenn der Benutzer einen nicht unterstützten Browser verwendet, muss eine weitere Bearbeitung des Request abgebrochen werden, und das durch die Rückgabe des *RedirectResult* an den *RequestDispatcher*.

[27] Redirect (deutsch. Weiterleitung) realisiert im Web durch Hypertext Transfer Protocol (HTTP) die Befehle.

4.1.1.3. PersistenceManager

Nachdem geprüft wurde, ob die Ausgabe auf der Klienten-Seite repräsentiert werden kann, wird eine Schnittstelle zwischen der Persistenzschicht und der Datenbank vorbereitet. Da das Framework nicht nur für die JDBC-Schnittstelle, sondern auch für den Einsatz der Hibernate[28] konzipiert wurde, kann der Front Controller um den dafür geeigneten Persistence Manager erweitert werden. Es ist auch möglich, gleichzeitig mehrere Persistance Manager anzuschließen, unabhängig davon, ob es sinnvoll ist oder nicht.

Der Persistence Manager ist vom Type her ein *StateableProcessor,* weil die geöffnete Connection oder die gestartete Transaktion mit der Datenbank nach der Request-Abarbeitung wieder freigegeben oder geschlossen werden muss.

4.1.1.4. PermissionEvaluator

Wie bereits der Name verrät, ist die Aufgabe dieses *RequestProcessor* zu prüfen, ob der Benutzer über die erforderlichen Rechte für die angesprochenen Ressourcen verfügt. Wenn Mängel an den Rechten festgestellt werden, wird ein *RedirectResult* mit dem dafür konfigurierten Webseiten-Link an den *RequestDispatcher* zurück übergeben.

4.1.1.5. Process Action

Dieser Prozessor ist für das Ausführen der *Actions* verantwortlich. Wie schon erwähnt, existieren zwei Typen von *Actions*: *SimpleAction* und *FormAction*. Vor dem Ausführen der *FormAction* initialisiert der *ActionProcessor* das *FormModel,* dessen Bean ID in *FormAction* gesetzt ist, und füllt sie mit aktuellen Eingaben der Benutzer.

Falls nach dem Ausführen der Action kein *RedirectResult* zurück geliefert wird, wird ein Redirect auf der zuletzt angezeigten Webseite ausgeführt (wie in JSF).

4.1.1.6. ModelInitializer

Falls die Aufgabe des Request nicht das Ausführen einer *Action* war und keiner der vorher ausgeführten Prozessoren ein Fehlverhalten aufwies oder *RedirectResult* zurück geliefert hat, muss die angesprochene Ressource – eine Webseite sein.

---

[28] Hibernate ist ein Framework für O/R-Mapping der Firma Red Hat, Inc.

Die Aufgabe des *ModelInitializer* besteht darin, alle benötigten *ViewModel* und *DataModel* zu initialisieren, gegebenenfalls mit den Daten füttern und deren Instanzen im *HttpServletRequest* auslagern. Der *ModelInitializer* ist ein *StateableRequestProcessor*, denn letzten Endes muss das initialisierte Model möglicherweise ordnungsgemäß zerstört werden.

4.1.1.7. PerspectiveResolver

Dieser Prozessor macht den letzten Schritt, um die im *DataModel* gekapselten Informationen dem Benutzer zu repräsentieren. Als Erstes wird hier das *ViewModel* initialisiert. Dann wird in den *Views,* deren Attribut *localeable* auf *true* gesetzt ist, der Pfad zu den JSP-Seiten erweitert in dem die von der Benutzer gewählte *Locale* eingestellt ist.

Als Letztes wird an den in der Root-Perspective gesetzten Pfad zu der JSP-Seite der Kontrollfluss übergeben. Die Perspective kann an den nächsten in der *View* gesetzten Pfad der zugeordneten JSP-Seite weiterhin Kontrolle übergeben, weil jetzt nur noch die JSP-Seiten ausgeführt werden müssen, um ein Response zu generieren und an den Benutzer zurückzuschicken.

4.1.2. JobIS

Diese Komponente stellt eine Infrastruktur zur Verfügung, die bestimmte Aufgaben in bestimmten Zeitintervallen ausführen. Klassen, wo solche Aufgaben implementiert werden, müssen den *AbstractJob* erweitern. Wann und welcher Job zum nächsten Mal ausgeführt werden soll, ist die Aufgabe der *JobSchedulerThread*. Der *JobManager* registriert jeder Job bei der *JobScherulerThread.*

Auf der Abbildung 4.4. sind zwei solche Jobs dargestellt.

MailSenderJob – ist für den E-Mail-Versand verantwortlich;

SQLJob – ist für das regelmäßige Ausführen von SQL[29] Statements in einer Datenbank in bestimmten Zeitintervallen zuständig. Dies ist eine sehr nützliche Funktion, die es ermöglicht, beispielsweise in definierten Zeitabständen Benutzerstatistik auszuwerten. Die auszuführenden SQL Statements können in XML konfiguriert werden.

---

[29] SQL (Structured Query Language) – eine Datenbanksprache zum Abfragen, zur Definition und zur Manipulation der Daten

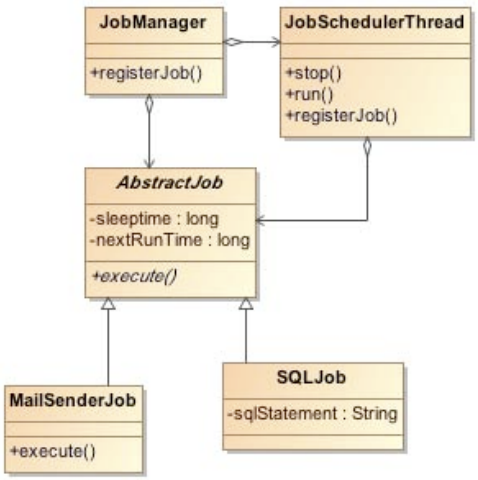

Abbildung 4.4. Klassendiagramm für *JobIS*

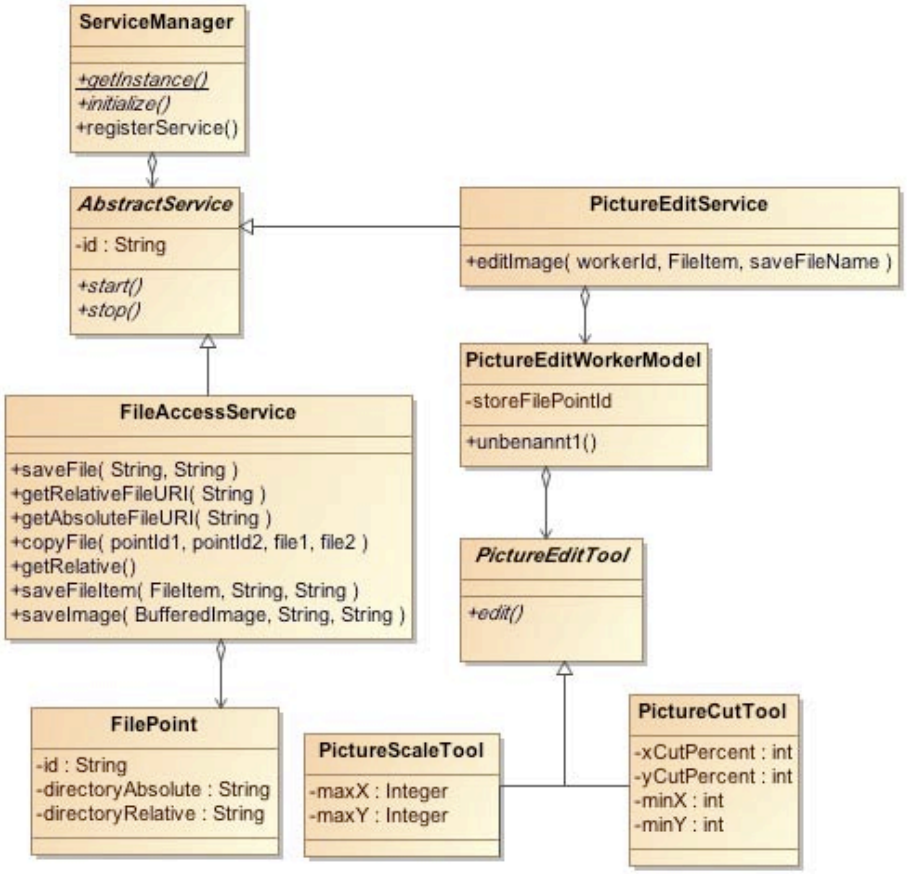

Abbildung 4.5. Klassendiagramm der *ServiceIS*

4.1.3. ServiceIS

Die Aufgabe dieser Komponente ist die entwickelten funktionellen Ressourcen anbinden zu können und dann zur Verfügung zu stellen.

Wie auf der Abbildung 4.5. veranschaulicht, müssen die entwickelten Funktionen den *AbstractService* erweitern. Diese entwickelten und in XML konfigurierten Services werden beim *ServiceManager* registriert und können dann bestimmte Routine-Aufgaben übernehmen und erledigen.

An dieser Stelle werden diese genauer beschrieben.

4.1.3.1. File Access Service

Dieser Service soll beim Speichern, Lesen und Ermitteln des Pfades von Dateien behilflich sein.

Dafür müssen die *FilePoint*s in XML konfiguriert werden. Jeder *FilePoint* verfügt über eine ID, einen absoluten Pfad und optional einen relativen Pfad. Der absolute Pfad verweist auf ein Verzeichnis, wo bestimmte Dateien abgelegt werden. Der relative Pfad wird dafür verwendet, um die URLs zu den Dateien, die unter einem zuvor konfigurierten absoluten Pfad gespeichert wurden, für den Zugriff aus einem Webbrowser ermitteln zu lassen, damit beispielsweise hochgeladene Bilder gespeichert und später angezeigt werden können.

4.1.3.2. Picture Edit Service

Dieser Service soll beim Speichern, Lesen und Ermitteln des Pfades von Dateien behilflich sein. Dafür müssen die *FilePoint*s in XML konfiguriert werden. Jeder *FilePoint* verfügt über eine ID, einen absoluten Pfad und optional einen relativen Pfad. Der absolute Pfad verweist auf ein Verzeichnis, wo bestimmte Dateien abgelegt werden. Der relative Pfad wird dafür verwendet, um die URLs zu den Dateien, die unter einem zuvor konfigurierten absoluten Pfad gespeichert wurden, für den Zugriff aus einem Webbrowser ermitteln zu lassen, damit beispielsweise hochgeladene Bilder gespeichert und später angezeigt werden können.

4.1.4. Applikation Environment Manager

Dieser Manager stellt eine Zwischenschicht zwischen dem Spring von der einen und dem Controller auf der anderen Seite.

4.1.5. Persistenzschicht Komponente

Das Ziel der Datenbank ist eine dauerhafte Speicherung von den Daten, die der Benutzer pflegt. Die Persistenzschicht spielt die Rolle einer Brücke zwischen den Daten, die in einer Datenbank abgelegt sind, und den anderen Komponenten der Webapplikation.

Die wesentliche Aufgabe dieser Schicht besteht im O/R-Mapping. Es muss für alle anderen Komponenten die strukturiert gekapselten Daten aus der Datenbank lesen und zur Verfügung stellen können, genau wie die auf diese Art zusammengefassten Daten in die Datenbank abzulegen.

Dem Entwickler steht völlig frei, ob und wenn ja, welches Framework er für diese Aufgabe einsetzen will. Es können beispielsweise mit der JDBC[30] die Daten eingeholt und manuell in Objekte zusammengefasst werden. So kann auch ein Hibernate als O/RMapper für die Automatisierung des Ablaufs eingesetzt werden.

4.2. Modul Environment

Diese Komponente definiert, aus welchen Bestandteilen die Module erstellt werden Können, sowie welche Schnittstellen die jeweiligen Bestandteile zur anderen Komponenten der Applikation haben. Es ist sozusagen ein Skelett für die Entwicklung der Module.

Weiter werden diese Bestandteile näher erläutert.

4.2.1. Data Model

DataModel besteht aus einfachen zusammengehörenden JavaBeans/POJO Klassen, in denen die zu repräsentierenden Informationen strukturiert festgehalten sind. Das Framework schreibt allerdings keine Einschränkungen für diese Klassen vor.

---

[30] Java Database Connectivity (JDBC) ist Datenbankschnittstelle für die Java-Applikationen

4.2.2. Model Preparer

Die Aufgabe der *AbstractDataModelPreparer* ist Data Model zu initialisieren. Wie auf der Abblidung 4.4. zu sehen ist, verfügen die um den *AbstractDataModelPreparer* erweiterten Klassen über Zugriffe auf Context- und Request-Scopes der Applikation, sowie auf *Hyperlink* Objekt, was für den Zugriff auf das ganze Environment der Applikation ausreichend sein soll.

Diese Komponente greift auf das DAO-Modul der Persistenzschicht zu, holt daraus Instanzen der *DataModel* Klassen und packt Sie in den Request Scope.

4.2.3. *ViewModel*

Im *ViewModel* wird konfiguriert, wie die Webseite aufgebaut ist, sowie welche Logic Routinen ausgeführt werden müssen, damit das für die Repräsentation notwendige *DataModel* initialisiert ist.

Es ist auf der Abblidung 4.6. zu sehen:

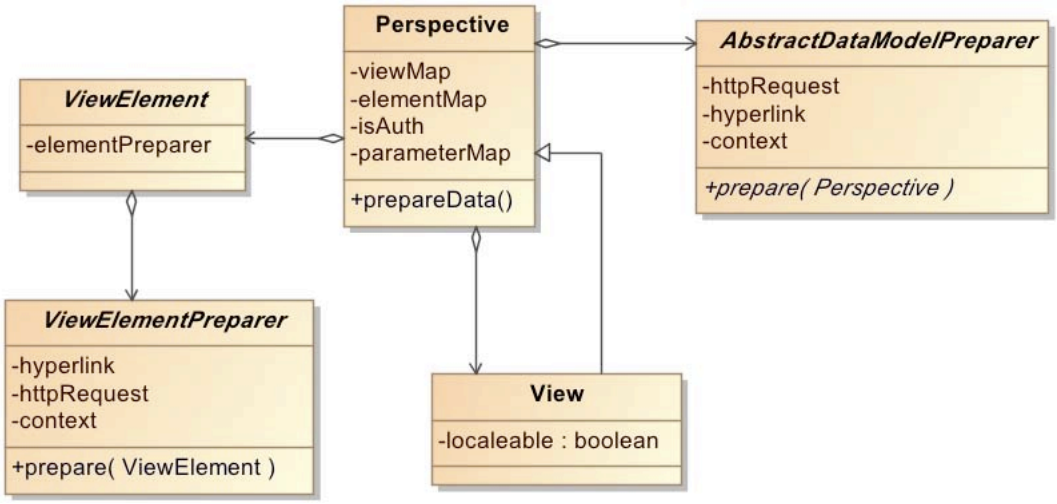

Abbildung 4.6. *ViewModel* Klassendiagramm

Das *ViewModel* setzt sich aus *Perspective* und beliebig tief ineinander geschachtelten *Views* zusammen. Zu jeder *Perspektive* kann ein *AbstractDataModelPreparer* und eine JSP-Seite registriert werden, damit der Front Controller in Kenntnis gesetzt wird, wo das Initialisieren des *DataModel* stattfindet, genauso wie, welche JSP-Seite für die Repräsentation dieses *DataModel* zuständig ist. Außerdem können an die *Perspective* implementierte *ViewElemente* angehängt werden. Durch das Implementieren solcher

ViewElemente ist es möglich, eigene Komponenten zu entwickeln und an das Projekt anzuschließen.

Im Gegensatz zu der *Perspective* kann jede *View* als *localeable* eingestellt werden. Dann müssen für jede Sprache dazugehörende Übersetzung-JSP-Seiten erstellt werden.

4.2.4. Form Model

Hier wird konzipiert, aus welchen Bestandteilen die Formkomponente besteht. Auf der Abbildung 4.6. sind alle Bestandteile der Formkomponente zu sehen und wie sie miteinander verbunden sind.

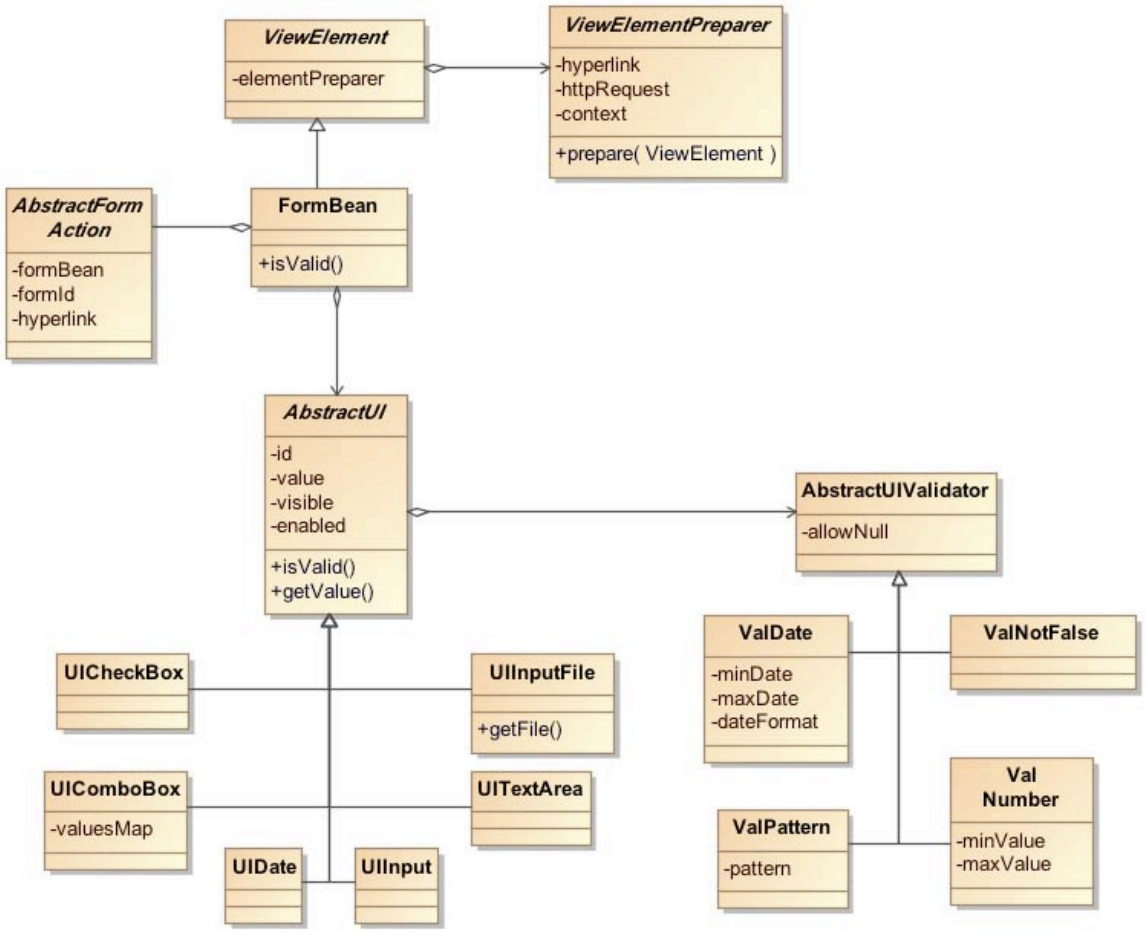

Abbildung 4.6. Klassendiagramm für das Form-Komponent

FormBean spielt die Rolle eines Containers, wo die Formular-Elemente zusammengefasst sind. Damit diese Komponente an das Framework angeschlossen werden kann, implementiert sie die abstrakte Klasse *ViewElement*.

Alle Form-Elemente erweitern die *AbstractUI* Klasse. Es kann definiert werden, mit welchen Validatoren ein *AbstractUI* Element auf seine Gültigkeit zu überprüfen ist. Die meisten der Validatoren müssen zusätzlich konfiguriert werden, um beispielsweise einen gültigen Zahlenbereich zu definieren.

Damit die zur Verfügung stehenden Form-Elemente mit Standart-Werten vorbereitet werden, vererbt das *FormBean* den Zugriff auf den *ViewElementPreparer*, der implementiert werden soll, um diese Aufgabe zu übernehmen.

4.2.5. Actions

Die Aufgabe der Actions ist meistens, die Eingaben vom Benutzer zu verarbeiten und in der Datenbank oder im Filesystem dauerhaft zu speichern oder zu verändern. Es wird zwischen *SimpleAction* und *FormAction* unterschieden. Die *FormActions* sind spezielle Actions, die die in einer Form eingegebenen Daten verarbeiten und damit auf das Modell dieser Form Zugriff haben.

4.2.6. Tag-Library

Durch den Tag-Library-Einsatz ist es bedacht, die Form-Elemente, Hyperlinks und Bilder in die JSP-Seiten zu integrieren und damit die JSPs möglichst klein und übersichtlich halten. Die Tag implementierenden Klassen haben Zugriff auf den *HttpServletRequest* sowie auf den *Context*. Somit steht für die in den Tag ausgelagerten Java-Anweisungen voller Zugriff auf das gesamte Modell der Applikation frei zur Verfügung.

Als Weiteres – die Tags müssen mit dem darzustellenden Design parametrisierbar sein. Für die Form-Elemente ist es auch möglich, den Style des Elements in dem Fall übergeben zu können, wenn in diesem Element unzulässige Werte eingegeben wurden. Es ist möglich, das Design für bestimmte Elemente mit CSS-Klassen[31] zu definieren und damit die in XML konfigurierte CSS-Klassen zu überschreiben.

[31] Cascading Style Sheets (CSS) ist eine in Webseiten oft eingesetzte deklarative Stylesheet-Sprache

4.3. Applikation Initialisierung

Das Initialisieren des Frameworks soll der *ApplicationListener* übernehmen. Es soll erst die konfigurierten Services und dann die Jobs initialisieren. Eine weitere Aufgabe des *ApplicationListener* ist das ordnungsgemäße Anhalten aller laufenden Jobs, wenn die Applikation abgeschlossen wird. Es wurde beschlossen, den *ApplicationListener* als die Implementierung der *ServletContextListener* Schnittstelle zu realisieren und durch deren Deklaration in der web.xml an das Lifecycle der Webapplikation anzuschließen.

Kapitel 5

5. Realisierung

In diesem Kapitel wird auf die Realisierung der SWA Frameworks eingegangen. Zuerst wird erläutert, welche Prototypen entwickelt wurden. Dann wird auf die Entwicklung denreinzelnen Komponenten eingegangen. Anschliessend wird erörtert wie die Tests durchgeführt wurden. Als kurzer Überblick, welche wichtigen Schritte während der Softwareentwicklung gemacht werden müssen, wird ein Testszenario ausgeführt.

5.1. Drei Prototypen

Das Ziel der Implementierung ist, anhand der drei Prototypen zu einem Ergebnis zu kommen, dessen Schnittstellen zwischen den MVC-Komponenten fest definiert sind, sodass bei der Umstellung auf die neuere Framework-Version so gut wie keine Änderungen an der auf dem Framework basierten Applikation durchgeführt werden müssen.

Vor dem weiter beschriebenen ersten Prototyp wurden einige experimentelle Prototypen entwickelt, anhand derer für die Frameworksentwicklung die Funktionsweise der eingesetzten Technologien getestet und ausprobiert wurden.

Jeder Prototyp wird getestet, indem eine auf dem Framework basierende Beispielsapplikation aufgebaut wird. Dann werden die Testergebnisse jedes Prototyp analysiert und die Ziele für den nächsten Prototyp definiert.

5.1.1. Prototyp 1

Mit dem ersten Prototyp wurde eine Basis für das MVC geschaffen. Der überwiegende Anteil der Arbeit konzentrierte sich auf die Entwicklung des *ViewModel* und auf die

Basisfunktionen des *FrontController*. Mit diesem Prototyp wurden folgende Implementierungen vorgenommen:

– Teil der *ViewModel*

Das erste Modell für die Modulentwicklung deckt das Modellieren von dem *ViewModel* und dem *DataModel* Komponenten ab;

– *ApplicationEnvironmentManager*

Der Manager initialisiert mit dem von Spring angeforderten Modell das Element und passt es dem richtigen Typ an;

– *FrontController*

In den ersten beiden Prototypen wurde der *FrontController*, als ein Filter-Chain durch die Erweiterung des Filter-Servlet[32] realisiert. Dieser Filter-Chain übernahm die in Kapitel Design beschriebene Funktionsweise von *Prozessoren*.

Der an dieser Stelle implementierte *FrontController* konnte die Requests Parameter so weit bearbeiten, dass das DataModel initialisiert werden konnte und dann die gewünschte Webseite angezeigt wurde.

5.1.2. Prototyp 2

Hierbei wurde eine weitere bedeutende Anforderung gestellt und implementiert:

– Erweiterung des *Environment:*

Das bestehende *ModulEnvironmentModel* wurde um das *FormModel* und die *Actions* erweitert. Das hier entwickelte *FormModel* wurde bereits im Kapitel Design vorgestellt. Im Unterschied zu diesem vorgestellten Design war es möglich durch den Aufruf der *getHtml()*-Methode die Form-Elemente (*AbstractUI*) direkt auf diese Weise in HTML zu rendern;

– Erweiterung der *FrontController:*

Es wurden die Unterstützung der *Form-Komponent*, so wie weitere gestellte Anforderungen an das Framework implementiert.

Das Ziel dieses Prototyps war die Erschaffung eines einsatzfähigen experimentellen Frameworks. So, nach dem dieser Prototyp entwickelt wurde, wurden die ersten beiden

[32] sieh Abschnitt 3.4.2.1. (Servlets)

funktionsreifen Web 2.0-Applikationen[33], basierend auf dem entwickelten SWA-Framework, aufgebaut.

Während der Entwicklung der beiden Webapplikationen wurden auch die unterschiedlichen Arbeitsweisen untersucht. Die größten Unterschiede bei diesen Applikationen liegen in der Struktur des *Hyperlinks*, nicht zu vergessen, dass diese auch Einfluss auf einige andere Teile der Applikation hatte.

Beispielsweise war es im Fall der ersten Webapplikation notwendig, im *Hyperlink* für die Anzeige eines Fotos auf der Seite eines Benutzers nicht nur die ID des anzuzeigenden Benutzerfotos, sondern auch die ID des Benutzers selbst abzustimmen, damit die Benutzer-spezifischen Daten angezeigt werden konnten. Dies führte zu aufwendigen Überprüfungen, ob die eingegebene Benutzer-ID tatsächlich zur anzuzeigenden Abbildung überhaupt passte. Ohne dies hätten Sicherheitslücken entstehen können, und Fotos möglicherweise einem anderen Benutzer angezeigt worden wären .

Hingegen war in der zweiten Webapplikation nur eine *genericId,* woraus die anderen IDs ableitbar waren. Somit konnte eine Foto-ID gesetzt werden, das Attribut *creatorId* in der Fototabelle zeigte dabei auf die ID den Benutzer, dem diese Foto gehörte.

5.1.3. Prototyp 3

Als Nächstes sind einige Änderungen des Designs durchgeführt worden. So wurde ein neues Konzept des *FrontControllers* entwickelt. Der Grund dafür war, dass in verschiedenen Applikationen unterschiedliche Anforderungen an den Controller-Lebenszyklus gestellt werden können. Deshalb sollte dieser Controller über umfangreiche Konfigurationsmöglichkeiten verfügen und sehr bequem mit anderen projektspezifischen Controllerfunktionen erweiterungsfähig sein. Das XML-Schema zur Beschreibung von Spring-Beans ist handlicher und umfassender als das, welches in web.xml verwendet wird. Außerdem ist es angenehmer, wenn das komplette Modell in einer Sprache deklariert und konfiguriert wird.

Die nächste durchgeführte Änderung, auf die bereits eingegangen wurde, ist die Entkoppelung des HTML-Rendering von den *AbstractUI*-Elementen in die Elemente der Tag Library, die bequemer zu benutzen sind und HTML-Templates überschaubarer gestalten lassen.

Die ersten beiden Prototypen und deren Test wurden in einem zusammengekoppelten System implementiert. Die Frameworkklassen waren dabei den beiden entwickelten Test-Webapplikationen zugefügt und damit auch projektspezifisch angepasst. Der dritte

[33] http://www.reeperbahn-community.de und http://www.deutschland-laune.de

Prototyp sollte eine festgesetzte Softwarearchitektur haben. Dadurch erfolgte hier die Aktualisierung des Design-Konzepts.

Da mit diesem Schritt alle eingeplanten Funktionen implementiert und getestet wurden, sowie ein *Refactoring* des Systems erfolgte, stehen die Schnittstellen zwischen den Komponenten fest. Im Fall der nächsten auf dem Framework basierenden Applikationen sollen die Portierungen auf neuere Versionen des Frameworks ohne große Anpassungen erfolgen.

Unter Anderem wurden mit Prototyp 3 die *ServiceIS* und das *JobIS* eingeführt.

5.2. FrontController

Hier folgen die wichtigsten Informationen über die Implementierung des *FrontController*.

RequestFilter – ist implementiert *javax.sevlet.Filter* Schnittstelle, in seiner *doFilter()* Methode holt er in der XML konfigurierten *RequestDispatcher*, und ruft bei ihm *process()* auf.

RequestDispatcher – verfügt über einen Array mit Prozessoren. Dieser Array wird in XML konfiguriert. Wenn die Methode *process()* aufgerufen wird, werden alle in der Array konfigurierte *Processoren* nacheinander ausgeführt. Der Prozessor der Unterklasse der *AbstractSimpleProzessoren* wird mit der Methode *execute() gestartet*, jedoch der *AbstractStateableProzessor* – mit *open()*. Nach dem Ausführen aller Prozessoren, werden die *Steateble* Prozessoren in umgekehrte Reihenfolge mit der *close()* Methode geschlossen.

BrowserValidator – ist so realisiert, dass es möglich ist, eine Liste von den nicht unterstützten Browsern zu konfigurieren, sowie festzustellen, welches Hyperlink für den Redirect benutzt werden soll, falls ein Browser aus dieser Liste verwendet wird. Falls die angesprochene Ressource mit dem Ressource auf das der konfigurierte Redirect-Hyperlink verweist gleich ist, wird die Abfrage durchgelassen und es wird kein Redirect ausgeführt. Welcher Browser benutzt wird, wird aus dem User-Agent-Header, der mit dem Request an den Server übermittelt wird, festgestellt.

PermissionEvaluator – hier wird einfach geprüft, ob die Instanz des User-Objekts im Session-Scope vorhanden ist oder nicht. Meldet sich der Benutzer erfolgreich an, wird sein User-Objekt initialisiert und im Session-Scope abgelegt. Bei der Abmeldung wird dieses Objekt gelöscht.

PersistenceManager – entkoppelt dem beim Webserver eingestellten Connection-Pool eine JDBC-Connection und setzt die Referenz im *HttpServletRequest*, damit alle weiteren

Module, die das Request abarbeiten, auf diese zugreifen können. Dies ist ein Steateable-Prozessor: Nachdem der Request abgearbeitet wurde, schiesst der Persistence Manager während der aufruf close()-Methode die Connection zu.

ProzessAction – hier wird einfach die angesprochene Action durch den Aufruf der *execute()*-Methode ausgeführt. Falls die Aktion keinen Redirect zurückgibt, wird ein automatischer Redirect auf die zuletzt angezeigte Webseite ausgeführt, dafür werden sie vorübergehend im Session-Scope abgespeichert. Alle zuletzt angezeigten Seiten dienen nicht dem Zweck, Actions auszuführen, sondern dem Benutzer die Informationen darzustellen.

ModelInitializer – hier werden für die angesprochenen *Perspective* und *Views* konfigurierte *AbstractDataModelPreparer* ausgeführt. Die *AbstractDataModelPreparer* greifen auf die DAOs der Persistanceschicht zu und laden erforderliche Objekte des *DataModels* in den Request-Scope. Dabei werden die *AbstractDataModelPreparer* aufsteigend, angefangen bei *Perspective,* nacheinander folgende Views zuerst in die initialisiert (rekursiver traversieren). Nachdem das Komplete *DataModel* geladen ist, wird für alle relevanten *ViewElements* die *ViewElementPreparer* aufgerufen, damit diese *ViewElements* auch initialisiert werden können.

PerspektiveResolver – hier wird die Anpassung von Views, deren Seiten Internationalisiert werden müssen, vorbereitet. Dabei wird das Ziel für den Zugriff auf die zu internationalisierenden Seite um jeweilige Locale aus der Session Attribut „userLocale“ erweitert, Beispielsweise „agb.jsp“ --> „agb_de.jsp“. Dann wird der Kontrollfluss an die in *Perspective* gesetzte JSP-Seite übergeben.

5.3. Hyperlink

Wie bereits erwähnt, hat sich die Verwendung von Hyperlink während der Entwicklung von Testapplikationen, die auf dem zweiten Prototyp basierten, verändert.

Der *Hyperlink* ist seit Prototyp 3 mit XML konfigurierbar und hat folgende Attribute:

Parameter	Beschreibung
id	dient als Suchkriterium, dass von der Spring Core verwendet wird, um den gesuchten Bean im XML-Dokument zu finden
perspectiveId	zeigt, welche *Perspective* angezeigt werden muss. Wird nicht bei den *Hyperlinks* verwendet, die für das Ausführen von *Actions* gedacht sind

Parameter	Beschreibung
viewMap	setzt fest, welche *Views* welchen *ViewAreas* zugeordnet sind
actionId	hier wird die ID der Action gesetzt und zeigt somit, welche *Action* ausgeführt werden soll, während diese das Request bearbeitet
genericId	diese Eigenschaft wurde während der Vorstellung des zweiten Prototyps bereits beschrieben
value	dieser Parameter wird beispielsweise zur Übergabe der gewählten Note während der Fotobewertung benutzt, dabei bekommt die *genericId* den Wert der ID vom bewerteten Foto

Tabelle 5.1. Hyperlink Parameter

Da der *Hyperlink* nur für die reine Datenkapselung gedacht ist, existiert eine *HyperlinkHelper*-Klasse, die die oben beschriebenen Parameter in ein String umwandelt. Diese Klasse wird als eine URL verwendet, um ein Request zu generieren. Zu den weiteren Aufgaben dieser Klasse gehört, aus dem Request der *Hyperlink* ein Objekt wiederherzustellen.

5.4. Die Form

Das *FormModel* wurde so, wie es im Kapitel Design vorgestellt wurde, implementiert und verfügt über keine besonders interessante Punkte. Es wurden einfache Klassen mit den als *private* deklarierten Variablen, die in dem Klassendiagramm stehen, erzeugt. Die get-/set-Methoden erlauben den Zugriff auf diese Variablen.

Klassen, die abstrakte *ViewElementPreparer* erweitern, müssen in der *prepare()*-Methode den übergebenen *FormBean* mit vorgelegten Daten füllen. Im Falle der UIComboBox werden die zur Auswahl stehenden Werte gesetzt.

Die Klasse *UIInputFile* verfügt über eine Methode *getFileItem()*, die ein Objekt der Klasse *FileItem* zurückliefert. Dieses File wird mit Hilfe von Commons FileUpload Bibliothek aus dem Request extrahiert.

5.5. Die Validatoren

Die *Validators* sind sehr einfach implementiert. Alle Validatoren können konfiguriert werden, ob sie „NULL“ als Wert akzeptieren oder nicht. Als „NULL“ wird ein leeres oder nur mit Leerzeichen gefühltes String interpretiert.

Erwähnenswert ist noch hier *ValPattern*. Dieser Validator wird mit einem Pattern konfiguriert, um nach dem gesetzten Muster in Form-Feld eingegebenen Wert zu überprüfen. Die Überprüfung ist mit dem Aufruf der Standartmethode *matches()* realisiert. Somit muss der gesetzte *pattern* dem in Java festgelegten regular-expression construct[34] entsprechen.

5.6. Die Tag Library

Alle implementierten Tag-Klassen haben den Zugriff auf *HttpServletRequest* und somit auch auf den Kontext der Applikation. Weiter unten werden alle implementierten Tags genauer beschrieben.

5.6.1. Die Formelements rendernde Tags

TForm ist das Haupt-Tag-Element. Damit alle nachfolgenden Elemente dieser Form zugeordnet werden können, ist die *TForm* wie der Body-Tag realisiert. Außer weiter beschriebenen Tags kann auch der *THyperlink* dem Body der *TForm* hinzugefügt werden. Dieser *THyperlink* zeigt der *Tform,* welcher URL verwendet werden soll. Das passiert nachdem der Benutzer einen Button anklickt, um die Eingaben zu bestätigen.

Damit die Form-Elemente in HTML gerendert werden können, haben diese Elemente als Eingangsparameter des String-Typs die Element-ID, die das Design bestimmende Design CSS Class und die CSS Class, für eine mögliche fehlerhafte Eingabe definiert.

Welcher „default“-Wert oder welche Werte für die Auswahl zur Verfügung gestellt werden sollen, ist die Aufgabe des *ViewElementPreparers*. Er füllt die Attribute der *FormBean* mit nötigen Werten.

Die Tag-Elemente können auf diesen *FormBean* und damit auch auf den mit diesem Tag zu rendernden *AbstractUI*-Element zugreifen. Die AbstractUI stellt nicht nur den „default“-

[34] s.u. http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html#sum

Wert zur Verfügung, sondern auch, beispielsweise in Falle einer *UIComboBox,* auch die für die Auswahl verfügbaren Werte.

Auf das Implementieren der Unterstützung des <button> mit einem Tag-Element wurde verzichtet, weil er für das bestehende HTML-Konzept völlig ausreichend ist.

5.6.2. Der THyperlink

Dieser Tag wird benutzt, um <a> HTML-Elemente zu rendern. Das *Hyperlink*-Objekt wird in XML mit den IDs für die Perspective, die Views und Actions konfiguriert. Der THyperlink holt über die ID-Instanz den Hyperlink, der in XML konfiguriert ist, über den *ApplicationEnvironmentManager* und setzt die *genericId* und *value* ein. Die zuletzt beschriebenen beiden Parameter sind optional. Die *Value* wird zur Vergabe der Note während der Foto-Bewertung verwendet, dabei bekommt die *genericId* einen Wert für die ID des zu bewerteten Fotos. Außerdem ist es möglich, an diese Stelle die CSS Class zu übergeben.

Für die visuelle Anzeige des Links ist dieser Tag wie ein Body Tag aufgebaut, der es erlaubt, weitere HTML-Text-Elemente zu enthalten.

5.6.3. Der TImgPath

Dieser Tag wird zum Ermitteln des Links für die Bildanzeige verwendet. Er verfügt über folgende Parameter:

– filePointId zeigt an, welcher *FilePoint* angesprochen werden soll;

– filename zeigt an, für welche im *FilePoint*-Verzeichnis definierte Datei eine URL ermittelt werden soll.

5.6.4. Die TIncludeView

Ist ein ganz einfacher Tag. Als Parameter wird hier die *ViewArea* übergeben, so erfasst der Tag die Lage der anzuzeigenden *View*. Nachdem die *View* aus der *Perspective* ermittelt wurde, wird an die in dieser *View* konfigurierte JSP-Seite die Kontrolle weitergegeben.

5.7. Die Modul Infrastructure

Wie auf der Abbildung 4.4. dargestellt, stellt das SWA Framework zwei Manager zur Verfügung: den *JobManager* und den *ServiceManager*. Beide Manager sind wie ein Singleton implementiert. Hier wird auf die Implementierung dieser beiden Komponenten genauer eingegangen.

5.7.1. JobIS

Diese Komponente ist für die periodische Ausführung bestimmter Aufgaben zuständig. Die erweiterten *AbstractJob* Klassen werden als Jobs bezeichnet. Der *AbstractJob* verfügt über die Attribute id, sleeptime (zeigt, mit welcher Amplitude dieser Job ausgeführt wird) und nextRunTime (mit der Information, wann dieser Job das nächste Mal ausgeführt werden soll). Zudem verfügt jeder Job über einen Zugriff auf den Applikationskontext. Durch den Aufruf der Methode *register(AbstractJob job)* können diese Jobs bei dem *JobManager* registriert werden. Weiterhin registriert der *JobManager* diese Jobs beim *JobSchedulerThread*.

Durch den Aufruf der Methode *register(AbstractJob job)* können diese Jobs bei dem *JobManager* registriert werden. Weiterhin registriert der *JobManager* diese Jobs beim *JobSchedulerThread*. Der *JobSchedulerThread* ist wie ein Thread implementiert. Die *run()* Methode geht alle registrierten Jobs nacheinander durch und führt den zuerst gefundenen aus. Die Voraussetzung ist, dass der Job entweder noch nicht ausgeführt wurde oder dass der nächste auszuführende Zeitpunkt (*nextRunTime* Attribut in *AbstractJob*) bereits vergangen ist. Nach dem Ausführen dieses Jobs wird seine *nextRunTime* erneuert. Dann übergeht der Scheduler zum weiteren Job, der ausgeführt werden soll. Wenn es keinen Job gibt, der ausgeführt werden soll, wird die Methode *sleep()* aufgerufen und der Thread bleibt bis zu einem bestimmten (ausgerechneten) Zeitpunkt inaktiv, bis der nächste auszuführende Job ausgeführt werden soll. Dann beginnt die Suche der Jobs erneut an. Es werden mit dem Framework zwei Jobs geliefert: der *MailSenderJob* und der *SQLJob*.

5.7.1.1. Der MailSenderJob

Wenn der *MailSenderJob* mit der *execute()* aktiviert wird, versucht er in dem, als *FilePoint* konfigurierten Verzeichnis, Dateien (E-Mail-Request) zu finden. Falls Eines vorhanden ist, wird dieser E-Mail-Request mit dem Spring zu einem Objekt der Klasse *EmailPageScopeBean* initialisiert. Die Parameter dieser Klasse sind in der Tabelle 5.2. aufgelistet.

Parameter	Beschreibung
`String htmlTemplatePath`	Der Pfad zum Template-File, der verwendet werden soll für die E-Mail-Generierung im HTML-Format.
`String plainTemplatePath`	Der Pfad des Template-File im Plain-Text-Format.
`String from`	erscheint in der E-Mail beim Empfänger.
`String[] to`	E-Mail-Adressen des Empfängers.
`Map<String, String> attachements`	Pfade der Files, die als Anhänge mit der E-Mail abgeschickt werden sollen.
`String subject`	Der Betreff der E-Mail für jeden Empfänger.
`Map<String, Object> properties`	Parameter für die Verwendung im Template.

Abbildung 5.2. Parameter des *EmailPageScopeBean*

Wenn der E-Mail-Job neue Dateien im Verzeichnis findet, initialisiert er sie mit Hilfe von Spring als ein *EmailPageScopeBean*. Dann werden beide Templates mit Velocity zu Strings transformiert. Anschließend werden diese beiden E-Mail Messages an die in XML Datei eingegebene E-Mail-Adresse abgeschickt. Nachdem die E-Mail abgeschickt ist, wird der File-Request gelöscht.

Diese Operation wird für alle gelesenen E-Mails durchgeführt. Falls keine gefunden Wurde, wird die *execute()* Methode abgeschlossen und das nächste Mal gestartet, wenn der Job wieder dran ist.

5.7.1.2. Der SQLJob

Dies ist ein sehr einfach zu implementierender Job. Zusätzlich zu den vererbten Attributen dieser *AbstractJob*-Klasse, besitzt dieser Job einen *sqlStatement*-String. Wie der Name verrät, wird hier das auszuführende SQL-Statement in XML parametisiert. Die *execute()* Methode baut eine JDBC Connection auf und führt das konfigurierte SQL-Statement aus.

5.7.2. Der ServiceIS

Diese Komponente stellt eine Infrastruktur für die Registrierung und die Verwaltung von Services zur Verfügung. Alle Services müssen den *AbstractService* implementieren. Die Services werden beim *ServiceManager* durch den Aufruf der *register()* Methode registriert.

Jeder *AbstractService* kann die *init()* und die *destroy()* Methoden implementieren, um spezifische Vorbereitungen und Abschlussaufgaben durchzuführen. Mit dem Framework werden zwei Services geliefert: der *FileAccessService* und der *PictureEditService*.

5.7.2.1. Der FileAccessService

Dies ist ein Service, der für die Verwaltung der Verzeichnisstruktur (*FilePoint*) zuständig ist. Der *FilePoint* an sich ist ein Verzeichnis, in dem die Dateien abgelegt werden können. Daher werden in jedem *FilePoint* seine *id* und der absolute Pfad auf der Festplatte für die Systemzugriffe (Löschen von Dateien, Hinzufügen von neuen Dateien), sowie die URL für den Zugriff über einen Browser auf die Dateien aus diesem Verzeichnis festgehalten.

Der *FileAccessService* wird in XML mit den verfügbaren *FilePoints* konfiguriert. Der *FileAccessService* hilft bei der Arbeit mit diesen Verzeichnissen, es verfügt über die Methoden *saveFile(filePointId, fileName)*, *deleteFile(filePointId, fileName)*, *getAbsolutPath(filePointId, fileName)*, *getRelativeURL(filePointId, fileName)* und *copyFile(filePointId1, fileName1, filePointId2, fileName2)*.

Dies ist eine gute Möglichkeit, um dynamisch hochgeladene Bilder zu speichern und später für den Zugriff über einen Browser die URL zu ermitteln.

5.7.2.2. PictureEditService

Dieser Service stellt die Werkzeuge (*PictureEditTool*) zur Verfügung, um Bilder zu skalieren und dann zu speichern.

Im *PictureEditService* werden die *PictureWorkers* als ein Wert und die Keys als eine ID dieser Worker in einer Map abgespeichert. Anhand des Key können diese *PictureWorkers* angesprochen werden. Jeder *PictureWorker* verfügt über ein Array an konfigurierten *PictureEditTools* und eine ID des *FilePoint*. Mit dem Aufruf der *edit()* Methode fangen die *PictureWorkers* an, mit den *PictureEditTools* das Bild zu bearbeiten. Am Ende wird das Ergebnisbild in dem angegebenem *FilePoint* abgelegt.

Es sind zwei Tools implementiert: das *PictureScaleTool* und das *PictureCutTool*.

PictureScaleTool – verfügt über zwei Attribute: *widthMax* und *heightMax*. Das Tool rechnet zuerst die Grenzen der proportional zu erzielenden Größe des Bildes aus. Dann wird das Bild mit dem Aufruf der *Image.scale()* Methode skaliert. Am Ende wird das Bild als ein *BufferedImage* zurückgegeben.

PictureCutTool – verfügt über zwei Attribute: die minimal zu erzielende Proportion zwischen den Größen des Bildausschnitts gegenüber dem Ursprungsbild für X und Y Achsen. Somit werden die Positionen für den Ausschnitt ermittelt. Der Ausschnitt wird erzeugt und das Ergebnisbild wird genau wie im *PictureScaleTool* als *BufferedImage* zurückgegeben.

5.8. Die Anbindung des Frameworks an den Tomcat Server

Für die Anbindung das Frameworks an den Servlet-Container müssen der *ApplicationListener*-, der *RequestFilter*- und der *contextConfigLocation*-Parameter konfiguriert werden.

Der ***ApplicationListener*** wird in der *web.xml*-Konfigurationsdateides Tomcat registriert, wie im Listing 5.1. dargestellt. Bei jedem Neustart der Applikation aktiviert und initialisiert das Framework alle benötigten Komponente.

```
<listener>
  <description>ServletContextListener</description>
    <listener-class>
          com.swa.ApplicationListener
    </listener-class>
</listener>
```

Listing 5.1. Binden des *ApplicationListener*

In XML konfiguriertes Model der Applikation muss bekannt gegeben werden[35].

Anbinden des ***RequestFilter*** kann exakt wie die Anbindung eines Servlet-Filters realisiert werden[36].

Da Spring zu den unabdingbaren Teilen des entwickelten Frameworks gehört, muss er auch gesondert angeschlossen werden. Dies wird folgendermaßen realisiert:

---

[35] wie in der Abschnitt 3.6.3.2. (Spring Core)

[36] wie in der Abschnitt 3.6.3.1. (Servlets)

```
<listener>
  <listener-class>
                org.springframework.web.context.request.RequestContextListener
          </listener-class>
</listener>

<listener>
  <listener-class>
                org.springframework.web.context.ContextLoaderListener
          </listener-class>
</listener>
```

Listing 5.2. Binden des Spring-Frameworks

5.9. Applikation Listener

Diese Klasse implementiert die *ServletContextListener* Schnittstelle. Damit werden zwei Methoden implementiert: *initialize()* und *destroy()*.

Die *initialize()* Methode ist für das Aktivieren der in XML konfigurierten bestimmten Komponenten der Applikation zuständig. Zuerst werden alle Manager nacheinander initialisiert – dafür werden die jeweiligen *getInstace()* Methoden aufgerufen. Dann werden Jobs und Services initialisiert, diese werden bei dem dazugehörigen *JobManager* und dem *ServiceManager* registriert.

Die *destroy()* Methode ruft bei allen Managern die Methode *destroy() auf* – so mit wird der *JobSchedulerThread* angehalten und die ausführung der Webapplikation erfolgreich abgeschlossen.

5.10. Test

Die Aufgabe des Tests ist die Qualitätssicherung durchzuführen. Auf die für diesen Zweck am besten geeigneten Test Cases wurde verzichtet. Dem liegt Folgendes Zugrunde: Während der Entwicklung wurden ständig Tests durchgeführt, nicht nur um die korrekte Ausführung der Funktionen zu überprüfen, sondern auch die Benutzerfreundlichkeit des Frameworks zu erproben und bessere Wege für die Realisierung zu finden. Außerdem wurden nach dem zweiten Prototype zwei große Webapplikationen entwickelt und damit alle denkbaren Testszenarios abgedeckt.

Tiefer wird auf ein Testszenario und auf seine wichtigsten Punkte eingegangen.

5.10.1. Testszenario „Entwicklung Group Seite für Reeperbahn-Community.de“

Erstens soll in der Realisierung das grafische Design der Webseiten erstellt werden. Als Ergebnis dieses Schrittes werden die anzuzeigenden Seiten anhand der konzipierten Maketen erstellt, dann werden daraus die Bilder ausgeschnitten. Anschließend werden alle Maketen in HTML umgesetzt und in JSPs gespeichert. Die Speicherung soll in der dafür in der Analyse vorgesehenen Filestruktur erfolgen, dabei ist es sehr sinnvoll so viele Designeigenschaften (Farbe, Schrift u.s.w.) wie möglich nicht in Tags zu definieren sondern sie in die CSS-Klassen auszulagern. Die CSS Klassen werden sinnvoll auf verschiedene Dateien aufgeteilt.

Dann müssen in der entwickelten Design-Datenbank durch das Ausführen von SQL Statements angelegt werden. An dieser Stelle wird auf die SWA Framework bezogenen Realisierungsteile genauer eingegangen.

5.10.1.1. Entwicklung der *DataModel*

Zuerst wird das *DataModel* gemäß dem auf der Abbildung 5.1. dargestellten Klassendiagramm als Entity implementiert und auf in der dazugehörenden entwickelten Datenbank mit Annotations abgebildet.

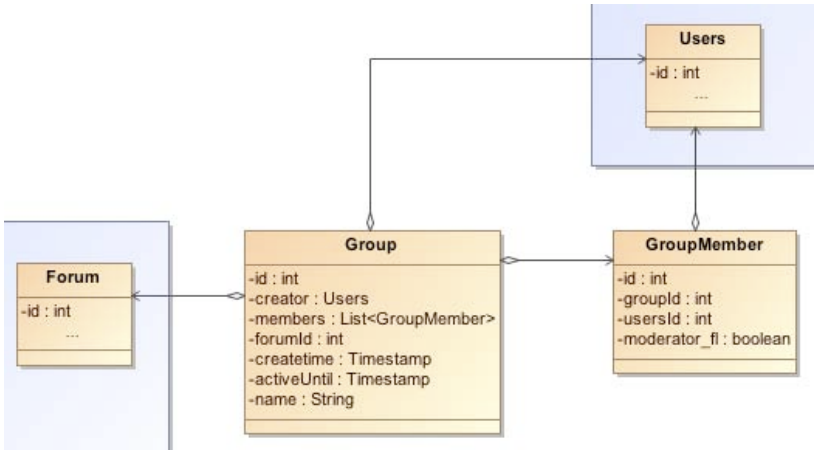

Abbildung 5.1. *DataModel* der Modul Group

Das Hibernate stellt einen *EntityManager* zur Verfügung, der die Aufgaben der DAOSchicht übernimmt. Mit dem *EntityManager* ist es möglich, die Objekte *Group*, *GroupMember* sowie die *DataModel* Elemente aus anderen Modulen (Forum oder User) in der Datenbank zu suchen und als Objekte zu laden, sowie den zugehörigen Datensatz in der Datenbank zu aktualisieren oder neu anzulegen.

5.10.1.2. *ViewModel*

An dieser Stelle wird in XML das *ViewModel* konfiguriert.

Die Struktur der *Perspective* Group ist auf der Abbildung 5.2. dargestellt.

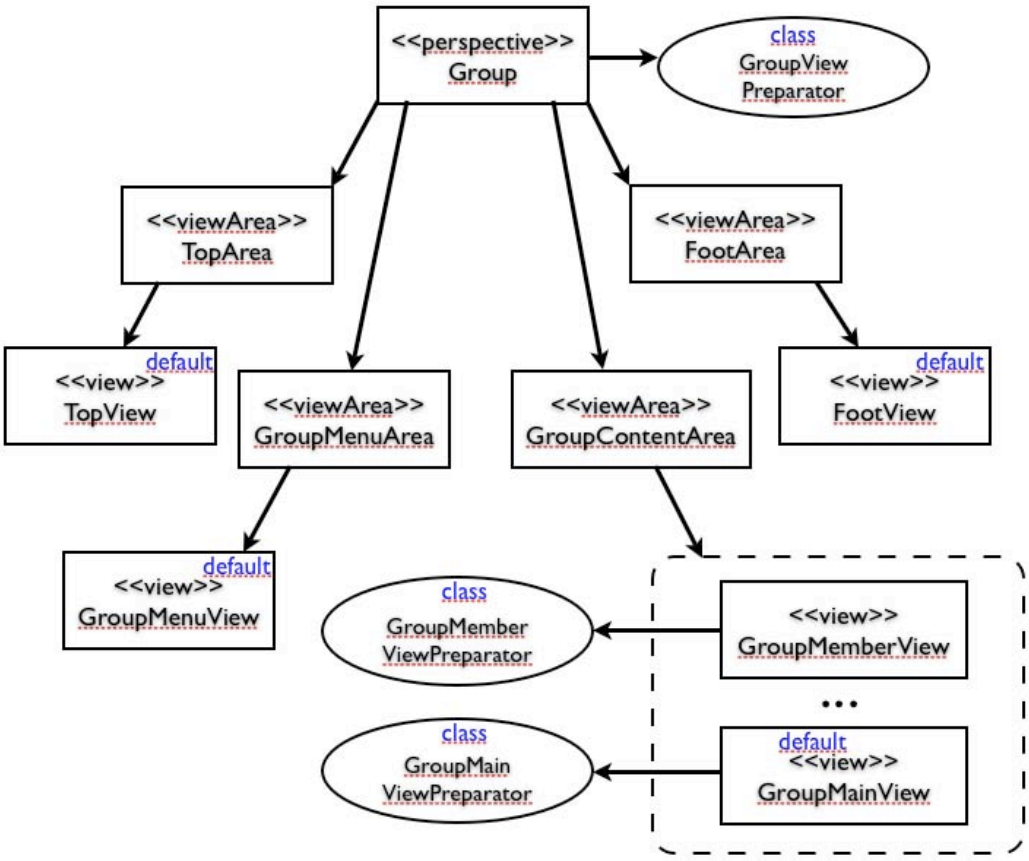

Abbildung 5.2. Struktur der Goup-*Perspective*

In der *Perspective* Group werden alle *ViewAreas* ausschließlich mit festen *Views* belegt außer der *GroupContextArea*. Hier werden, je nachdem, entweder die *GroupMemberView* oder die *GroupMainView* angezeigt. Falls im Request kein Wert für die anzuzeigende

GroupContentArea gesetzt wird, wird die *GroupMainView* angezeigt. In der *group.xml* werden diese *Perspective* und die dazugehörenden *Views* konfiguriert.

Der in der Perspective konfigurierte *DataPreparer* initialisiert die in den Views repräsentierten *DataModel* Elementen. So werden beispielsweise im *GroupViewPreparer* für die, mit dem *genericIc* Attribut des *Hyperlink* Objekts gesetzte Group ID, das *Group* Objekt aus der Datenbank geholt. Dann werden die Moderatoren und sechs Mitglieder als ein Array der *Users* Objekte aus dieser Gruppe geholt. Diese drei initialisierten Datenkapsel werden im Request mit den dazugehörenden Namen abgelegt (*group*, *groupModerators* und *groupMenuMembers*).

Genau auf diese Weise können die *GroupMemberViewPreparer* und *GroupMainViewPreparer* auch die benötigten Datenkapsel vorbereiten. So werden im *GroupMemberViewPreparer* alle Benutzer als ein Array von *Users* auch im Request Scope abgelegt.

5.10.1.3. *Action* Implementierung

Für die Modul Group werden mehrere Actions implementiert. Es wird genauer Beschrieben, welche Operationen in der Aktion für die Taste „Group beitreten“ ausgeführt werden.

Es wird ein neues Objekt *GroupMember* mit *groupId*, *moderator_fl* und *userId* gesetzt. Die *GroupId* wird aus dem *genericId* Attribut des *Hyperlink* Objekts übernommen. Die *userId* wird aus dem im Session Scope stehenden Users Objekt (dieses wird während der erfolgreicher Login Action in den Session Scope gepackt) übernommen. Der *Moderator_fl* wird stadartmäßig *false* gesetzt. Diese Action muss keinen expliziten Redirekt auf bestimmte Seite liefern, es reicht, wenn auf die zuletzt angezeigte Seite von dem FrontController ein automatischer Redirekt ausgeführt wird.

5.10.1.4. Hyperlinks Konfiguration

Damit die Hyperlinks leichter der View hinzugefügt werden können, ist es möglich, diese Hyperlinks in XML zu konfigurieren und sie dann beim ApplikationModelManager abzufragen. Für die Anzeige der Group *Perspective* mit der *GroupMainView* wird ein Hyperlink nur mit der anzuzeigende *Perspective* konfiguriert. Damit dieselbe *Perspective* aber mit *GroupMemberView* angezeigt wird, muss zusätzlich zu der *Perspective* das *ViewMap* Attribut im Hyperlink konfiguriert werden. In der *ViewMap* wird gesetzt, dass zur „*GroupContentArea*“ (Key) eine „ *GroupMainView*“ angezeigt werden soll. Beide Einträge werden als String gesetzt.

Im Falle des Hyperlinks (zum Ausführen von Actions) im Hyperlink *actionId wird im* Attribut die Id von der auszuführenden *Action* gesetzt.

5.10.1.5. Implementierung einer Action für Fotoupload

Damit die Gruppenbilder angezeigt und hochgeladen werden können, wird ein ***FilePont*** angelegt. Hier wird konfiguriert, wo das Verzeichnis für die Ablage dieser Bilder zu finden ist und welche URL für dieses Verzeichnis für den Zugriff mit einem Web-Browser verwendet werden muss. Somit ist es möglich, über *TImg* aus dem JSP diese konfigurierte URL abzufragen. Es können mehrere *FilePoints* für die Gruppenbild-Anzeige in verschiedenen Größen deklariert werden.

Zum **Skalieren** des hochgeladenen Bildes wird ein *PictureEditWorker* in XML konfiguriert und beim *FileAccessService* registriert. In diesem *PictureEditWorker* wird der *FilePoint* definiert, der auf das Zielverzeichnis, wo das Ergebnis-Bild gespeichert werden soll, zeigt. Dann werden die *PictureEditWorker* in einem Array *PictureEditTool* angelegt. In diesem Fall wird nur ein Tool gebraucht und zwar das *PictureScaleTool*. In diesem Tool wird konfiguriert, welche maximale Größen das Ergebnis-Bild erreichen darf, damit es dem genau durchdachten konzeptionellen Aspekt der Webseite entspricht.

Damit die Datei hochgeladen werden kann, muss eine **Form** erzeugt werden. Dafür wird ein *FormBean* als ein Bean in der XML angelegt. Diesem *FormBean* wird eine *id* vergeben, unter der dieser erreichbar wird. Als Nächstes wird ein Bean dem *UIInputFile* in der Liste mit den Formelementen dieser *FormBeans* angehängt. Da dieses Feld nicht mit den Daten vorbereitet werden muss, kann auf den *FormDataPreparer* verzichtet werden.

Des Weiteren wird eine Klasse-Aktion implementiert, die die Eigenschaften der *AbstractFormAction* vererbt hat und dem XML angeschlossen wird. Wenn diese Action ausgeführt wird, hat sie Zugriff auf den *FormBean* mit aktuell gesetzten Werten. Von diesem *FormBean* wird mit der Methode *getInputFile(String id) ein* Form-Element *UIInputFile* geholt. Wenn eine Datei erfolgreich hochgeladen ist, kann sie bei diesem *UIInputFile* durch Aufruf der Methode *getFileItem()* geholt werden. Dann muss diesen Datei skaliert und gespeichert werden. Dafür wird beim *ServiceManager das* registrierte *PictureEditService* abgefragt. Durch dern Aufruf der Methode *editImage()* bei diesem Service wird der Skalierungsprozess gestartet. Dabei muss die hochgeladene Datei (als FileItem), die ID des zuvor konfigurierten *PictureEditWorker* und der File-Name, unter dem das Ergebnis-Bild gespeichert ist, übergeben. Da für jede Group nur ein Bild hochgeladen werden kann, wird das Bild unter der GroupId gespeichert. Das neu hochgeladene Bild überschreibt das alte.

5.10.1.6. Views Entwicklung

Nachdem die Data- und Logic-Module der Model Komponente entwickelt und in XML konfiguriert wurden, können die Views mit HTML und CSS bereits vorbereitete JSP-Seiten mit Model-Elementen belegen. Zu dem Zeitpunkt, wenn die JSP ausgeführt wird, haben die ausgeführten erweiterten *AbstractDataModelPreparer* Klassen die benötigten *DataModel*-Elemente initialisiert und in den Request Scope abgelegt. In den Views ist es möglich, mit JSTL auf diese *DataModel*- Elemente zuzugreifen. Für die Bild-Anzeige wird der *TImg* Tag benutzt, wie es im Listing 5.3. auf der Zeile 1. (unten) angezeigt ist.

```
1. <swa:TImg filePoint="groupBigFotoId„ file="${group.id}.jpg„ />
2. <swa:TIncludeView viewArea="GroupMenuArea„ />
```

Listing 5.3. Benutzen von TImg

Auf der zweiten Zeile wird demonstriert, wie mit Hilfe des *TIncludeView* Tags die Views ineinander geschachtelt werden können. Dabei müssen an diesen Tag die viewArea-Parameter mit der ID dieser Area übergeben werden.

Auf dem nächsten Listing 5.4. abgebildet, wie die Form in einer JSP-Seite realisiert werden kann.

```
<swa:TForm id="fileUpload„ method="post„ enctype="multipart/form-data„ >
      <swa:TLinkURL id="groupFotoUpload„ genericId="${group.id}„ />
      <swa:TFile id="file_f„ css="cssClass„ cssError="cssErrorClass„ />
      <swa:TFormError id=" file_f„ cssError="cssErrorClass„ >
            Der gewählte Datei konnte nicht hochgeladen werden
            wählen Sie bitte einen anderen aus
      </swa:TFormError>
</swa:TForm>
```

Listing 5.4. Entwicklung einer Form in JSP

Im Tag *TForm* wird die id dem in XML entsprechend konfigurierten *FormBean* übergeben. Es wird ein entsprechender *enctype* gesetzt, der das Hochladen von Dateien unterstützt. Die *TLinkURL be*sagt, welche URL das Request verwendet werden soll. Mit dem *TFile* wird das speziell entworfene Input-Feld dargestellt. Im Falle wenn ein falsches oder gar kein File gewählt wurde, wird dieses Feld bei der Anzeige mit dem in *cssErrorClass* definierten Design angezeigt. *TFormError* ist ein Body-Tag und kann in seinem Body noch weitere HTML-Elemente oder Text enthalten. Diese werden dem Benutzer angezeigt, falls

einen bestimmtes Form-Element fehlerhaft ausgefüllt wurde. Das id-Parameter zeigt, in welchem Feld der Error-Status abgefragt werden soll.

5.11. Bewertung

An dieser Stelle wird die Realisierung von gestellten Anforderungen bewertet, und anschließend darauf eingegangen, wie effizient die Webapplikationen, basierend auf diesem Framework, entwickelt werden können.

Erweiterbarkeit

Diese Anforderung wurde mehrmals während der inkrementellen Entwicklung der Plattform getestet. Die neuen Funktionen lassen sich sehr leicht anschliessen. Durch die Implementierung dafür vorgesehener Schnittstellen bekommt man Zugriff auf die nötigen Ressourcen. Allerdings es ist sehr schwierig dieses Kriterium zu testen, da die Erweiterungsprobleme sich erst nach Überladung einer Software mit den zur Verfügung stehenden Funktionen erkennen lassen. Das erste Zeichen dafür ist: Es wird kein passender Platz für die anzuschliessende Komponente gefunden. Dieser Fall trat nicht auf.

Entkoppelung der Konfiguration

Die komplette Konfiguration des Frameworks, so wie auch die darauf gebauten Applikationen können in XML als Spring-Beans vorgenommen werden. Grundsätzlich ist auch alles Mögliche konfigurierbar. Neue Funktionen können in XML angeschlossen werden.

Trennung zwischen Modellvorbereitung und Modelländerung

In dem Framework gibt es *Actions* und *ModelPreparer*. Die Modellvorbereitung sollte durch Vererben von *ModelPreparern* realisierbar sein. Die *Actions* sind für die Änderung des *DataModels* und der Datenbank gedacht.

Portalbasierte Aufbau der Webapplikation

Während der Entwicklung der Webapplikation ist es möglich die einzelnen Seiten auf die Unterseiten (View) hierarchisch (wie auf der Abbildung 5.2.) zu unterteilen. Für jedes View kann eine eigene Logic entwickelt werden, die speziell für diese View eine Model vorbereitet.

Form-Komponente

Es wurde eine Form Komponente entwickelt. Diese Form muss in der XML deklariert werden, was vielleicht im ersten Blick als unnötiger Aufwand angesehen wird. Die Form-Elemente (ComboBox, InputFeld, ...) sind auch realisiert, müssen jedoch wie die Form in XML deklariert und konfiguriert werden. CSS Klassen können in JSP gesetzt werden. Diese Palette an Form-Elementen kann um weitere fehlende Komponenten wie z.B. List erweitert werden. Zu jedem Feld ist es möglich Validatoren anzuschliessen, durch die eine automatische Kontrolle der Eingaben realisiert werden kann. Es ist möglich in JSP einen Platzhalter pro Feld zu reservieren, in dem Fehlermeldungen angezeigt werden können.

E-Mail-Versand

Es ist Anforderungs-gerecht möglich, die E-Mails templatebasiert abzusenden. Die Templates können die als Map übergebenen Parameter verwenden.

Periodisches Ausführen bestimmter Aufgaben

Es ist möglich periodisch ausführbare Aufgaben zu implementieren. So wurde zum Beispiel der E-Mail Versand realisiert.

Internationalisierung

Die Internationalisierung wird durch die Ausführung des JSTL Tag realisierbar.

Bearbeiten von Bildern

Es stehen Funktionen (Skalierbarkeit und Bildausschnitt) für die Bildbearbeitung zur Verfügung. Wie schon erwähnt, sind in XML beide Funktionen konfigurierbar.

Rechte des Systems

Es ist auf minimale Anforderungen abgestimmt. Es wird einfach geprüft, ob der Benutzer angemeldet ist oder nicht, was entscheidend für die Zugriffe auf die Ressourcen ist, die nämlich Authentifizierung fordern.

File Manager

Es ist möglich in XML zu konfigurieren, in welchen Verzeichnissen welche Datei Arten zu speichern sind.

Skalierbarkeit

Diese Frage wurde nur analysiert und nicht getestet. Allerdings im Rahmen der Entwicklung Deutschland-Laune.de Projekt gibt es ein Modul, welches als synchronisierte Ressource über Web Server ansprechbar ist, und so mit getrennt auf einem anderem Webserver installiert werden kann.

Das entwickelte Framework zeichnet sich nicht durch sauberes Trennen zwischen Model und View aus, was sicherlich als ein Nachteil angesehen werden kann. Doch diese Verbindung ist recht klein und für das Einsatzgebiet wenig nachteilig.

Der übergreifende Einsatz von Dependency Injection Paradigma, was durch Spring in der Framework unterstützt wird, bietet dem Entwickler eine hervorragende Möglichkeit, viel Programmier-Arbeit mit Model Driven Software Development (MDSD)[37] zu erleichtern. Eine nicht in dieser Arbeit getestete Möglichkeit ist der Einsatz von AndroMDA[38], welches das so genannte Cartridgen für Spring und auch für Hibernate anbietet.

Durch den Einsatz dieses Tools ist es denkbar, das Konzipieren und Programmieren in einem Schritt zu erledigen. Da für das Konzipieren oft Modelle erstellt werden, um die Gedanken festzuhalten, ist es möglich einen MDA-Tool einzusetzen. Das Ergebnis kann gleichzeitig als XML-Konfiguration oder mindestens eines Teils davon verwendet werden.

[37] Model Driven Software Development (MDSD) ist eine Paradigma um aus Modellen Software generieren

[38] AndroMDA ist einen Open Source MDA Generator

Kapitel 6

6. Zusammenfassung

Im Rahmen der Arbeit wurde ein MVC-basiertes Framework, welches für die Entwicklung der Web 2.0 Anwendungen geeignet ist, angefertigt. Darauf basierend wurden zwei Webapplikationen entwickelt, die den Nutzen des Frameworks zu überprüfen hatten.

Anhand der Tests wurde festgestellt, dass das Framework die an ihn gestellten Anforderungen zum größten Teil erfüllt hat.

Für die Erstellung der zukünftigen Webapplikationen wurden aus der Arbeit folgende Erkenntnisse gewonnen: Auf der Grundlage der aus den Tests der Webapplikationen gelieferten Ergebnisse wurde der Bedarf der Bildung von generischem Layout für eine bestimmte Domäne an Web 2.0 erkannt. Dieses generische Layout kann mit Hilfe von HTML realisiert werden und kann durch CSS konfiguriert werden (Design).

Des Weiteren ist es mit Sicherheit möglich, durch den Einsatz von MDA die automatisierte Entwicklung von Webapplikationen auf diesem Framework basierend zu entwickeln.

Diese Erweiterungen würden den Zeitaufwand des Programmierens erheblich senken.

Literaturverzeichnis

[Beeger 2006] Beeger, Robert F.; Haase, Arno; Roock, Stefan; Sanitz, Sebastian: *Persistenz in Java-Systemen mit Hibernate 3*: Dpunkt-Verlag (2006)

[Bleek 2008] Bleek, Wolf-Gideon; Wolf, Henning: Agile Softwareentwicklung: Dpunkt-Verlag (2008)

[Branter 2001] Brantner, Matthias; Schmidt, Stefan; Wabnitz, Birgit und Detlef: *JavaServer Pages und Servlets*: Data Becker-Verlag (2001)

[Cross] Cross, Rob: *Making Invisible Work Visible: Using Social Network Anakysis to Support Strategic Collaboration*. – URL https://webapp.comm.virginia.edu/networkroundtable/portals/0/making_invisible_work_visible.pdf

[Fowler 2004] Fowler, Martin: *Inversion of control containers and the dependency injection pattern*. http://martinfowler.com/articles/injection.html (2004)

[Gamma 1995] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns.* Addison-Wesley (1995)

[Jayaswal 2006] Jayaswal, Bijay K.; Patton, Peter C.: *Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software*: Prentice Hall-Verlag (2006)

[Johnson Ralph 1993] Johnson, Ralph E.: *How to design Frameworks* (1993). – URL http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/how-design-frame.pdf

[Ledford 2007] Ledford, Jerri L.; Tyler, Mary E.: *Google Analytics 2.0*: Wiley-Verlag (2007)

[Masak 2005] Masak, Dieter: *Moderne Enterprise Architekturen*: Springer-Verlag (2005)

[Minter 2008] Minter, Dave: *Beginning Spring 2: From Novice to Professional*: Appress-Verlag (2008)

[Minter, Linwook 2005] Minter, Dave; Linwood, Jeff: *Pro Hibernate 3*: Appress-Verlag (2005)

[Momjian 2004] Momjian, Bruce; *PostgreSQL: Introduction and Concepts*: Addison-Wesley-Verlag (2004)

[Oates 2006] Oates, Richard; Langer, Thomas; Wille, Stefan; Lueckow, Torsten; Bachlmayr, Gerald: *Spring & Hibernate. Eine praxisbezogene Einführung*, Hanser Verlag (2006)

[O'Reilly 2005] O'Reilly, Tim: *What Is Web 2.0* – Design Patterns and Business Models for the Next Generation of Software; (2005); http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[Pomberger 2004] Pomberger, Gustav; Pree, Wolfgang: *Software Engineering*: Hanser-Verlag (2004)

[Reenskaug 1979] Reenskaug, Trygve M.H.: *Xerox PARC technical Note: Models View Controller* (1979). – URL http://folk.uio.no/trygver/1979/mvc-2/1979-12-MVC.pdf

[Richter 2007] Richter A.; Koch M.Dr.: *Social Software – Status quo und Zukunft*, Bericht 02/2007, Fakultät für Informatik, Universität München

[Sandhöfer 2008] Sandhöfer, Jürgen: „*Kein schwarzes Loch – internet facts erforscht Web 2.0"* Research & Results, 5/2008

[Schmidt 2006] Schmidt, J.: *Social Software: Onlinegestütztes Informations-, Identitäts- und beziehungsmanagement.*, Forschungsjournal Neue Soziale Bewegungen, Nr. 2/2006 S. 38-45.

[Schwarz 2006] Schwarz, Torsten; Braun, Gabriele: *Leitfaden Integrierte Kommunikation* (2006)

[Seschadri 1999] Seschadri, Govind: Server-Side Java: Understanding JavaServer Pages Model 2 Architekture. – URL http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html?page=2

[Sixtus 2005] Sixtus, M.: *W wie Wiki*, Die Zeit, 25.08.2005, Nr. 35

[Sixtus 2006] *Das Netz erfindet sich neu*, ct-magazin für Computer und Technik, hiese verlag, Nr 05/06.

[Spring] *Spring reference documentation, Version 2.5*: URL http://www.springframework.org

[Stark 2007] Stark, Thomas: *Java EE 5*: Addison-Wesley-Verlag (2007)

[Web 2.0 Begriff] Begriff Web 2.0 – URL http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[Wolff 2007] Wolff, Peter: Die Macht der Blogs – *Chancen und Risiken von Corporate Blogs und Podcasting in Unternehmen;* 2. erw. Auflage; Datakontext-Verlag

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, 18. September 2008

Ort, Datum

Unterschrift