

Dokumentation der Fopra-Verwaltung

Inhaltsverzeichnis

| | |
|---|----|
| Aufgabenstellung und Problemanalyse..... | 2 |
| Basissystem..... | 2 |
| Problemanalyse und Vorgehensstrategie..... | 3 |
| Architektur und verwendete Technologien..... | 3 |
| Das Datenmodell..... | 6 |
| Besonderheiten des Datenmodells..... | 7 |
| Struts-spezifische Klassen..... | 7 |
| Die Action-Klassen..... | 8 |
| Die Klasse Utils..... | 8 |
| Die Forms-Klassen..... | 8 |
| Die Filter-Klassen..... | 8 |
| Sonstige Hilfsklassen..... | 9 |
| Webseiten..... | 9 |
| WEB-INF..... | 10 |
| Bibliotheken..... | 10 |
| Einrichten der Entwicklungsumgebung..... | 10 |
| Konfiguration..... | 11 |
| Spezielle Konfiguration (Konfigurationsdatei config.xml)..... | 12 |
| Links..... | 15 |

Aufgabenstellung und Problemanalyse

In diesem Fortgeschritten-Praktikum ging es um den Entwurf und die Entwicklung eines webbasierten Verwaltungssystems für Fortgeschritten-Praktika des Fachbereich Mathematik und Informatik der Universität Marburg. Das Ziel war, den Aufwand für die, momentan manuell durchgeführte, Organisation der Praktika durch eine IT-Systemunterstützung zu minimieren.

Basissystem

Mit der Entwicklung des Systems wurde bereits im Sommersemester 2007 als Übung begleitend zu der Veranstaltung „Entwurf großer Software Systeme“ betreut durch Prof. Dr. Hesse und Andrej Bachmann begonnen. Hier stand dasselbe praktische Ziel im Vordergrund, jedoch sollten auch die in der Vorlesung vorgestellten Konzepte zur Realisierung von größeren Softwareprojekten zur Anwendung kommen. Dies schloss die präzise Orientierung an dem Vorgehensmodell OpenUP¹ ein, auf dessen Grundlage der Entwicklungsprozess organisiert wurde.

Ausschlaggebend für den Projektverlauf war damit einerseits die Einteilung in evolutionäre Iterationsphasen, die dem Projekt eine feste zeitliche Struktur auferlegten bzw. die termingerechte Verfügbarkeit von den zu erstellenden Fragmenten sicherstellten. Auf der anderen Seite wurde bereits in der Anfangsphase eine umfangreiche Dokumentation der Anforderungen und Aufgaben erstellt, die ermöglichte, die Ziele des Projekts klar zu definieren und die Fähigkeiten des Entwicklerteams den Interessen des Auftraggebers (Andrej Bachmann) gegenüber zu stellen und sie mit diesen abzustimmen.

In einer Art an der industriellen Wirklichkeit orientiertem „Rollenspiel“ übernahmen die Teammitglieder die Aufgabenbereiche von Analysten, Architekten, Projekt Managern, Testern und Entwicklern, wie sie durch OpenUP vorgesehen sind. Infolgedessen wurde innerhalb des Projektes durch das Management die zeitgerechte Einhaltung des Projektplans herbeigeführt, während Analysten die Anforderungsanalyse betrieben und sich Analysten und Architekten auf die technische Realisierung des Systems konzentrieren konnten.

Gegen Ende des Semesters standen bereits die grundlegenden Mechanismen des Systems als Webapplikation bereit, für einzelne Anwendungsfälle war ein Durchstich bis in die Datenbankschicht realisiert. Die eigentliche Aufgabenstellung, nämlich die Implementierung aller in der Elobarationsphase ausgearbeiteten Anwendungsfälle, war damit allerdings noch lange nicht erfüllt. Für die einzelnen Anwendungsfälle existierte zwar ein rudimentäres Web-Layout, das an das Design der Fachbereichswebseiten angelehnt war, jedoch fehlte es selbst dort noch an visueller Aufbereitung, Übersichtlichkeit und applikationslogischer Fehlerbehandlung und Validierung.

In diesem Zusammenhang entschieden wir uns gemeinsam mit den Auftraggebern, diesen vorrangig mit der Implementierung und Konstruktion

¹OpenUP Vorgehensmodell: <http://www.epfwiki.net/wikis/openup/>

befassten Teil der Systementwicklung als Fortgeschrittenen-Praktikum fortzusetzen, um das Projekt zu einem gelungenen Abschluss zu bringen.

Problemanalyse und Vorgehensstrategie

Aufgrund der bereits im Vorhinein entstandenen Dokumentation, existierte bereits zu Beginn des Fortgeschritten-Praktikums eine komplett verschriftlichte Anforderungsanalyse des Projektes. Das Vorgehen orientierte sich demnach stark an der Implementierung und dem Design der fehlenden Anwendungsfälle und der weiteren Ausarbeitung des bestehenden Codes und der Webseiten. Bei Unklarheiten über konkret gewünschte Ausprägungen des Systems, fand meist eine direkte Rücksprache mit dem Auftragssteller (Andrej Bachmann) statt. Außerdem wurden regelmäßige Treffen vereinbart, an denen neu hinzugekommene Funktionalität präsentiert und Rücksprache über eventuell gewünschte Änderungen gehalten werden konnte.

Die komplette Dokumentation der Anwendungsfälle im *Microsoft Word* – Format ist in der Distribution des Projektes enthalten, und kann dort eingesehen werden.

In der letzten Phase des Fortgeschritten-Praktikums, die Ende September 2007 begann, ging es dann hauptsächlich um die Stabilität, Benutzbarkeit und Sicherheit des Systems. Zu Testzwecken wurde die Applikation mehrfach in der jeweils aktuellen Version auf einen Webserver des Fachbereichs überspielt und dort vom Auftraggeber und von den Entwicklern intensiv geprüft. In einem weiteren Treffen wurde dann über kleinere Änderungen an der Applikation gesprochen, und die Bereinigung von den gefundenen Fehlern geplant. Anschließend wurden die besprochenen Änderungen durch die Entwickler in das System integriert. Im Folgenden wurden immer wieder kleinere Änderungen und Verbesserungen vorgenommen, die sich durch die Benutzung des Systems durch den Auftraggeber ergaben.

Architektur und verwendete Technologien

Die in der Fopra-Verwaltung eingesetzten Technologien, die unter anderem die Verwaltung der Persistenz der verwendeten Objekte, das Erstellen des Datenmodells oder die Entwicklung einer Weboberfläche unterstützen oder übernehmen sind:

- ✓ Als Programmiersprache wurde **Java** (Version 6.0) verwendet
- ✓ **EMF** (2.3.x)/**Teneo** (0.8.0) für das Generieren des Datenmodells inklusive einer Datenzugriffsschicht mit **Hibernate** (3.2.5)
- ✓ **MySQL** (ab Version 5.0) als Vorgabe für die Datenbankbindung (mit entsprechendem Treiber und durch Anpassen der Konfiguration kann jede beliebige Datenbank mit JDBC-Treiber angebunden werden)

- ✓ **Struts** (Version 1.3.8) ist ein Framework für Webanwendungen, die dem Model-View-Controller-Architekturmuster folgen, die Seiten selbst benutzen **JSP**
- ✓ Ein geeigneter Server, der die benötigten Java-Web-Standards erfüllt, ist **Apache Tomcat** (ab Version 5.5)
- ✓ Weiterhin wurden Java-EE/SE-Technologien benutzt, darunter: **JNDI** und **JavaMail** (1.4.1)
- ✓ **iText** (2.0.6) für den PDF-Export

In Abbildung 1 soll der Zusammenhang der Komponenten noch einmal verdeutlicht werden.



Abbildung 1: Die Architektur (Zusammenspiel der Komponenten)

Das Datenmodell



Abbildung 2: Das ecore-Datenmodell

Das Datenmodell wurde, wie oben bereits angemerkt, mit Hilfe diverser, am Markt etablierter Tools erstellt. Diese Tools, nämlich EMF (zur

Generierung von Modellcode) in Kombination mit Teneo (für die Persistenz der Objekte), stellen auch die Datenbankzugriffsschicht her. Das mit EMF erstellte Datenmodell-Diagramm (Ecore-Diagramm) ist in Abbildung 2 dargestellt.

Besonderheiten des Datenmodells

EMF erlaubt in der aktuellen Version keine bidirektionalen Assoziationen. Dies hatte zur Folge, dass die Beziehungen zwischen der Klasse *Fopra* und *Betreuer*, sowie der Klasse *FopraListe* und *Fopra* zweifach existieren – in jede Richtung einmal. Dies führt wiederum dazu, dass zwei identische Tabellen im Datenbankschema angelegt werden, die die Daten für die wechselseitige Referenzierung (jedoch in umgekehrter Reihenfolge) beinhalten.

Dadurch tut sich ein Problem auf: Abgesehen von der Tatsache, dass es schwierig bis unmöglich ist, kaskadierendes Löschverhalten einzustellen, müssen ja Einträge aus beiden Tabellen gelöscht werden. Die Auswirkungen hiervon werden beim Löschen von Fopras deutlich: zuerst müssen die Betreuer des zu löschenden Fopras abgetrennt werden, sodann das zu löschende Fopra von den Betreuern. Erst jetzt können Betreuer und Fopras gelöscht werden (möglicherweise hätte eine Abtrennung gereicht; nichtsdestotrotz bleibt ein Update-Vorgang nach zumindest einer Abtrennung, der zusätzlich zum eigentlichen Löschen durchgeführt werden muss).

Ähnlich verhält es sich mit den Fopralisten. Allerdings dürfen keine Fopras gelöscht werden, die sich in einer Fopraliste befinden.

FopraStatus ist eine Enumeration (Aufzählung), wie sie seit Java 5 unterstützt wird. Sie dient dazu den Status eines Fopras angebar zu machen. Die Bedeutung der einzelnen Status kann der Webseite zur Statusänderung eines Fopras entnommen werden.

Die Klasse *FopraManagement* ist eine Containerklasse, in der alle anderen Klassen (zumindest über Vererbungen) enthalten sein müssen. Dies ist ein Charakteristikum der EMF-Spezifikation, die eine Datenspeicherung grundsätzlich im XML-Format vorsieht.

Der nicht-löschbare Standard-Administrator wird durch seinen Benutzernamen ausgezeichnet: er heißt stets „admin“ (*Benutzername*). Um zu gewährleisten, dass es einen solchen gibt, muss nach dem Einrichten der Software (Web-/Datenbankserver) und dem Aufspielen der Webseite die Action /setup.do aufgerufen werden.

Struts-spezifische Klassen

Die Klassen, die im Package *web* zu finden sind, sind, bis auf die Ausnahme der Klasse *Utils*, Klassen, die von Struts verwendet werden. Die direkt im Package befindlichen Klassen mit dem Suffix *Action* sind die Controller-Klassen, die Struts benutzt, wann immer ein Webformular abgesendet wird. Im Unterpaket *forms* finden sich die *ActionForm*-Klassen, die als Container für Formulareingaben dienen. Die Verknüpfung der beiden Genannten mit den entsprechenden Webseiten findet in der Konfigurationsdatei von Struts, der *struts-config.xml*, statt.

Weiterhin gibt es ein Unterpaket *beans*, das eine Klasse zur Kapselung der für die Anzeige von Fortgeschrittenenpraktika notwendigen Daten enthält. Das Unterpaket *exceptionhandlers* beinhaltet eine Klasse, die der allgemeinen Abhandlung von unerwarteten Fehlern (Exceptions) dient. Im Unterpaket *filters* befinden sich noch zwei Klassen, die als Filter für Anfragen (Requests) fungieren.

Die Action-Klassen

Die Klassen, die für Struts als sogenannte Actions dienen, haben im MVC-Architekturmuster die Aufgabe des Controllers. Hier werden die Eingaben, die auf der Weboberfläche gemacht wurden ggf. überprüft und in die Datenbank übernommen.

An einigen Stellen wurde, um Redundanz zu vermeiden dieselbe Action-Klasse für mehrere Actions benutzt (dies kann man in der *struts-config.xml* einstellen).

Die Klasse Utils

Die Klasse *Utils* ist eine, wie der Name schon sagt, Hilfsklasse, die diverse Funktionen wie das Senden von E-Mails, das Auslesen von Konfigurationsdaten aus der Konfigurationsdatei, das Überprüfen von E-Mail-Adressen und auch die Suche im LDAP-Verzeichnis bereitstellt. Da es sich um eine reine Hilfsklasse mit Hilfsfunktionen handelt, ist sie nicht instanzierbar. Alle Methoden der Klasse sind mit Javadoc dokumentiert, sodass ihre Anwendung schnell klar werden dürfte.

Die Forms-Klassen

Die Klassen im Package *web.forms* repräsentieren allesamt Eingaben, die in Formulare auf der Webseite gemacht werden. Letztlich sind es nur Datenkapseln mit Getter- und Setter-Methoden und ggf. zwei speziellen Methoden, *validate* und *reset*. Die Methode *validate* dient der Validierung der eingegebenen Daten, die Methode *reset* setzt die Daten auf (in der Methode definierbare) Standardwerte zurück.

Auch hier wurde, um Redundanz zu vermeiden, einige Male dieselbe Forms-Klasse für verschiedene Formulare verwendet. Dies ist insbesondere in Fällen so, in denen das Webformular potentiell viele Checkboxen, die durch einen Long-Wert ausgezeichnet sind, besitzt.

Die Filter-Klassen

Die Klasse *UTF8Filter* im Package *web.filters* dient dazu, die Anfragen vom Benutzer als UTF-8-codiert auszuzeichnen. Mit dieser Klasse wird ein bekannter Bug von Apache Tomcat umgangen. Die Klasse *RestrictedAccessFilter* prüft die „Absenderadresse“ der Anfrage daraufhin, ob sie innerhalb

des Universitäts-Netzwerkes (hier in Marburg: UMRnet) liegt. Falls das nicht der Fall ist, wird ein HTTP/403-Fehler zurückgegeben und eine speziell angepasste Fehlerseite angezeigt.

Sonstige Hilfsklassen

Im Paket *fopramgmt.util* finden sich noch einige Hilfsklassen, darunter auch eine Klasse mit einer Methode zum Exportieren einer Liste aus Fopras in ein PDF-Dokument.

- **HibernateManager**, **HibernateUtil** stellen Hilfsfunktionen zum Datenzugriff bereit. **HibernateUtil** wird nicht mehr verwendet.
- **PDFexport** ist für den Export von Listen von Fopras nach PDF zuständig
- **TeneoSample** soll die Verwendung von Teneo verdeutlichen
- **SemesterList** erstellt dynamisch Strings mit den Namen der Semester seit WS 2005/06
- **Test** ist eine Testklasse

Webseiten

Fast alle Webseiten der Fopra-Verwaltung benutzen die Tiles-Technologie von Struts, d.h. wiederkehrende und allen Dateien gemeinsame Teile der Webseiten werden in separate Dateien ausgelagert und über eine Definitionsdatei (tiles-defs.xml) bei der Anzeige wieder zusammengesetzt.

Die Semantik der Ordnerstruktur dürfte sich aus den Namen der Ordner ergeben. In **jsp** liegen die JSP-Seiten, in **css** die Stylesheet-spezifischen Elemente, die wegen des HTTPS-Zugriffs auf die Seite, wie sie jetzt im Betrieb ist, in das Projekt aufzunehmen waren (anstelle des Verweises auf die Fachbereichsseite), im **img** die Grafiken, die von der Seite verwendet werden. In **js** liegen von den css-Dateien referenzierte Javascript-Dateien. Im Ordner **jscalendar** findet sich ein in Javascript implementierter, dynamischer Kalender.

Webseiten, die keiner vorherigen Überprüfungsphase (d.h. eine Action vor der Anzeige, um bspw. für die Darstellung der Seite relevante Daten aus der Datenbank zu laden) unterliegen, wurden mit einem direkten Forward versehen, d.h. nach außen sieht der Aufruf einer bestimmten Webseite genauso aus, wie der einer Action. Beispiel hierfür ist die Seite „fopraHinzufuegen“ (nach außen zugreifbar durch *fopraHinzufuegen.do*) mit einem direkten Forward auf eine Tiles-Seitendefinition (*page.fopraHinzufuegen*),

WEB-INF

Im Verzeichnis WEB-INF befinden sich sämtliche Konfigurationsdateien. Insbesondere sind dies:

- **web.xml** – die Standardkonfigurationsdatei einer JavaEE-Webseite
- **struts-config.xml** – die Konfigurationsdatei von Struts
- **tiles-defs.xml** – die Definitionsdatei für Struts-Tiles
- **struts-bean.tld, struts-html.tld, struts-logic.tld, struts-nested.tld, struts-tiles.tld** – Definitionsdateien für Tag-Bibliotheken (Taglibs), die in den JSP-Seiten Verwendung finden
- **validation.xml, validator-rules.xml** – Regeln und Konfiguration für die Validierung (wurde im Projekt nicht benutzt)
- **errorpages/403.jsp** – die Fehlerseite, die bei einem HTTP/403-Fehler angezeigt wird

Bibliotheken

Die Bibliotheken, die für das Projekt benötigt werden, finden sich in Unterverzeichnis *lib* von WEB-INF. Sofern möglich, wurden weitestgehend die bei der Erstellung der Dokumentation aktuellsten Versionen der Bibliotheken eingesetzt, um reibungslosen Betrieb mit fehlerbereinigten Versionen zu gewährleisten.

Einrichten der Entwicklungsumgebung

Als Entwicklungsumgebung wurde Eclipse gewählt, da es die Entwicklung eines Webprojektes geeignet unterstützen kann. Eine Reihe von Plugins sind allerdings vonnöten, dies sind

- Teneo und EMF
- GMF für das Editieren des Ecore-Diagramms (Abhängigkeiten!)
- WTP (insbesondere JST) und alle Abhängigkeiten
- Amateras StrutsIDE und alle Abhängigkeiten (darunter GEF)

- optional TPTP, Subclipse zur Quellcodeverwaltung

Diese Plugins des Eclipse Projects können alle über die Discovery Site eingespielt werden. Am Besten lädt man sich Teneo, Amateras StrutsIDE und dessen Abhängigkeiten herunter, installiert die Plugins, wählt danach im Update-Managers (unter „Search new features to install“) alle möglichen Quellen aus, wartet, bis alle Download-Seiten abgerufen wurden und wählt als allererstes „Select required“. Dies sollte, sofern das Repository aktuell und komplett ist, alle Abhängigkeiten der Plugins mitinstallieren.

Überdies ist ein aktueller **Apache Tomcat** sowie ein **MySQL-Datenbankserver** angeraten; theoretisch kann auch mit anderen Produkten entwickelt werden, Erfahrungsberichte hierzu liegen aber nicht vor (die Umstellung des Datenbankservers dürfte weniger Schwierigkeiten machen als die des Webservers). Um nun integriert testen zu können, muss eine sogenannte Server Runtime richtig eingerichtet sein und das Projekt so konfiguriert werden, dass es damit läuft. Hierzu konsultiere man die Dokumentation der WTP. Zur Entwicklung wurden Versionen 6.0.13/6.0.14 von Tomcat und 5.0/5.1 von MySQL benutzt.

Die Schritte der Konfiguration, die im nächsten Kapitel erläutert werden, sind ebenfalls zu beherzigen.

Konfiguration

Die Konfiguration der Anwendung geschieht an verschiedenen Stellen:

1. Zur allgemeinen (zusätzlichen) Konfiguration der Webseite gibt es die Datei **WEB-INF/web.xml**. In ihr kann der Zugriff auf die Seite geregelt, weitere Filter eingerichtet, spezielle Fehlerseiten angegeben usw. werden. Hierzu konsultiere man die Tomcat-Dokumentation.
2. In **WEB-INF/classes/hibernate.properties** kann die Datenbankanbindung konfiguriert werden. Weitere Informationen dazu finden sich in der Hibernate-Dokumentation.
3. Die Datei **WEB-INF/classes/log4j.properties** regelt die Protokollierung (Logging) des Webservers über log4j. Mehr Informationen hierzu gibt es in der log4j-Dokumentation.
4. Ressourcen-Strings können, beispielsweise zur Lokalisierung (aber nur für Meldungen, die Webseiten selbst enthalten auch deutschen Text), in **WEB-INF/classes/MessageResources.properties** angepasst werden.

In der Regel wird es genügen, die Dateien unter 2. und 3. zu modifizieren. Prinzipiell muss nichts angepasst werden, wenn man den Datenbankserver mit den Einstellungen einrichtet, wie sie von *hibernate.properties* vorgegeben werden.

Spezielle Konfiguration (Konfigurationsdatei config.xml)

Die Fopraverwaltung verfügt außerdem noch über eine eigene XML-Konfigurationsdatei (die Java-Properties enthält), die angepasst werden und sich an einem bestimmten Ort befinden muss. Für die Feststellung des Ortes der Konfigurationsdatei **config.xml** gibt es zwei Möglichkeiten:

I. Im Benutzerverzeichnis (Java-Eigenschaft **user.dir**):

Hier muss ein Unterverzeichnis **fopramgmt**, darin wiederum **config** vorhanden sein. Hierin lege man die Konfigurationsdatei. Also **fopramgmt/config** unter **user.dir**.

II. In einem anderen Verzeichnis:

Dazu muss die Umgebungsvariable **fopramgmt_config_path** auf den Pfad gesetzt sein, der die Konfigurationsdatei enthält. Der Name der Konfigurationsdatei selbst darf nicht enthalten sein.

Es wird zuerst unter der Umgebungsvariablen (II.) nachgeschaut; ist diese nicht gesetzt, wird ein Fallback zum Benutzerverzeichnis (I.) durchgeführt. Die Verzeichnisse müssen nicht existieren. Sie werden gegebenenfalls angelegt und eine Standardkonfigurationsdatei wird (nach Bedarf der zu ladenden Einstellungen) erstellt. Dies ist somit eine Alternative zum a-priori-Erstellen oder Anpassen der Konfigurationsdatei. Hierbei gilt es zu beachten, dass die Standardwerte der einzelnen Einstellungen nur dann geschrieben werden, wenn die Einstellungen auch benötigt werden. In der folgenden Tabelle werden die Einstellungen einzeln erläutert.

| Einstellung | Bedeutung | Wird benötigt | V ² | Standardwert |
|----------------------|---|---|----------------|--|
| mail.body.adminack | <i>E-Mail-Text bei Administratorbestätigung</i> | <i>Fopra hinzufügen</i> | 1,2 | <code>\n{%foprainfo%}\n\nAckUrl={%url%}</code> |
| mail.body.announce | <i>E-Mail-Text bei Vorstellungsankündigung</i> | <i>Vorstellung ankündigen</i> | 2,3 | <code>\n{%fopratermine%}\n\n{%url%}</code> |
| mail.body.deadline | <i>E-Mail-Text bei Stichtags-Ankündigung</i> | <i>Stichtag ankündigen</i> | 2,4 | <code>\nDeadline={%stichtag%}\n\n{%url%}</code> |
| mail.body.bearkey | <i>E-Mail-Text beim Versenden des Bearbeiter-Schlüssels</i> | <i>Fopra hinzufügen, Bearb.-Schlüssel versenden</i> | 1,2,5 | <code>\n{%foprainfo%}\n\nBear={%bearbeiterschluessel%}\n\n{%url%}</code> |
| mail.body.betrkey | <i>E-Mail-Text beim Versenden des Betreuer- (und Bearbeiter-)Schlüssels</i> | <i>Fopra hinzufügen</i> | 1,2,5,6 | <code>\n{%foprainfo%}\n\nBetr={%betreuerschluessel%}\n\nBear={%bearbeiterschluessel%}\n\n{%url%}</code> |
| mail.body.regconfirm | <i>E-Mail-Text für Bestätigung einer Anmeldung zu einem Fopra</i> | <i>Zu Fopra anmelden</i> | 1,2,7,8,9 | <code>\n{%foprainfo%}\n\nUrl={%url%}\n\nBearb={%bearbeitervorname%}\n\n{%bearbeiternachname%}\n\nMail={%bearbeiteremail%}\n</code> |
| mail.body.userack | <i>Text für die Bestätigung des Eintragenden</i> | <i>Fopra hinzufügen</i> | 1,2 | <code>\n{%foprainfo%}\n\nAckUrl={%url%}</code> |
| resources.basepath | <i>Basispfad für Fopra-Ressourcen</i> | <i>Datei hoch-, herunterladen, löschen</i> | - | <code>user.dir+/fopramgmt/data</code> |
| ldap.host | <i>LDAP-Host für Mitarbeiter-Verifikation</i> | <i>Fopra hinzufügen</i> | - | <code>ldap://ldaphost.mathematik.uni-marburg.de:389</code> |
| ldap.basedn | <i>Basis-Verzeichnisname im LDAP</i> | <i>Fopra hinzufügen</i> | - | <code>ou=People,dc=mathematik,dc=uni-marburg,dc=de</code> |
| mail.smtp.host | <i>SMTP-Hostname für E-Mail-Versand</i> | <i>Alle E-Mail-Aktionen³</i> | - | <code>mailhost.mathematik.uni-marburg.de</code> |
| mail.sender | <i>E-Mail-Absenderadresse</i> | <i>Alle E-Mail-Aktionen³</i> | - | <code>admin@mathematik.uni-marburg.de</code> |
| mail.contenttype | <i>E-Mail MIME-Typ und evtl. Codierung</i> | <i>Alle E-Mail-Aktionen³</i> | - | <code>text/plain; charset=UTF-8</code> |

2 Variablen: 1=foprainfo, 2=url, 3=fopratermine, 4=stichtag, 5=bearbeiterschluessel, 6=betreuerschluessel, 7=bearbeitervorname, 8=bearbeiternachname, 9=bearbeiteremail

3 Dies sind alle Aktionen, die Einstellungen mit dem Präfix *mail.body.* benötigen.

Ein Beispiel für eine Konfigurationsdatei ist in **WEB-INF/classes/config.xml**.

Links

- > Amateras StrutsIDE:
http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=StrutsIDE
- > Apache Tomcat: <http://tomcat.apache.org/>
- > Eclipse: <http://www.eclipse.org/>
- > EMF, GEF, GMF (bei Tools Project), Teneo (bei EMFT), TPTP, WTP:
<http://www.eclipse.org/projects/>
- > Hibernate: <http://www.hibernate.org/>
- > iText: <http://www.lowagie.com/iText/>
- > Java EE: <http://java.sun.com/javaee/index.jsp>
- > Java: <http://java.sun.com/index.jsp>
- > JavaMail: <http://java.sun.com/products/javamail/>
- > Java Server Pages (JSP): <http://java.sun.com/products/jsp/>
- > JNDI: <http://java.sun.com/products/jndi/>
- > jscalendar: <http://dynarch.com/mishoo/calendar.epl>
- > log4j: <http://logging.apache.org/log4j/>
- > MySQL Community Server: <http://dev.mysql.com/>
- > Struts 1: <http://struts.apache.org/1.x/index.html>