Mitglieder: Sebastian Kohl (SK), Mirko Schmidt (MS), Peter Matjeschk (PM), Carsten Lüdecke (CL), Michael Thiele (MT), Sven Fischer (SF), Ibrahim Osman (IO), Maik Müller (MM)

Entwurfsbeschreibung
Cooperator

Inhaltsverzeichnis

Mitglieder: Sebastian Kohl (SK), Mirko Schmidt (MS), Peter Matjeschk (PM), Carsten Lüdecke (CL), Michael Thiele (MT), Sven Fischer (SF), Ibrahim Osman (IO), Maik Müller (MM)

1 Allgemeines

Mit dem Softwareprodukt, soll eine gemeinsame Verwaltung mehrerer Lehrveranstaltungen und Prüfungen ermöglicht werden.
Die wichtigsten Funktionen einer solchen Applikation sind dabei die Einschreibung für Lehrveranstaltungen und Prüfungen. Beide Arten der Einschreibung sind an bestimmte Bedingungen, wie z.B. Zulassungsvorausetzungen geknüpft. Zulassungsvorausetzungen stehen in Form eines Webservices der entsprechenden UebManager-Instanzen oder als XML Datei zur Verfügung. Die Ergebnisse der Einschreibung (Listen) werden durch die Plattform über eine Schnittstelle zur Verfügung gestellt werden.
Diese Software wird als webbasiertes Informationssystem entwickelt, d.h. die Bedienung der Software findet über einen Browser statt. Die Applikation selbst befindet sich auf einem Server, welcher einen Servlet fähigen Container, in diesem Fall Apache Tomcat Server verwendet.
Als Basistechnologie wird das Open Source-Framework Struts zum Erstellen von Web-Applikationen in Java verwenden**.**

2 Produktübersicht

Die Software bietet die Möglichkeit sich sowohl über das Intranet als auch über das Internet einzuloggen. Jeder eingerichtete Benutzer besitzt einen Benutzernamen und ein Passwort. Die Benutzer der Applikation werden in drei verschiedene Rollen unterteilt, die Rolle Admin, Dozent und Student. Jede Rolle hat andere Rechte und Funktionen. Der Dozent ist verantwortlich, bzw. hat die Möglichkeit Veranstaltungen bereitzustellen, Studenten können sich für diese einschreiben und der Admin ist in der Lage das gesamte System zu beeinflussen und zu kontrollieren.

3 Grundsätzliche Struktur- und Entwurfsprinzipien des Systems

Dem System liegt das MVC-Konzept zugrunde, welches die Datenverwaltung (Modell), die (graphische) Benutzeroberfläche (View) und Programmlogik (Control) soweit als möglich trennt.
Realisiert wird das MVC- Konzept durch die Verwendung des Struts-Framework.

Model
Das Datenmodell dieser Software wird auf Basis von XML erstellt. Zur einfachen Handhabung von XML-Dateien wird das JDOM-Framework verwendet und von diesem ausgehend Manager-Klassen aufgesetzt, die alle von einer groben Verwaltungs-Klasse abgeleitet werden.

View
Als Schnittstelle zwischen Applikation(Server) und dem Benutzer (Client) fungieren aus JSP-Seiten dynamisch generierte HTML-Seiten, welche im Browser des Benutzers angezeigt werden. Das Layout der Seiten wird per CSS und Struts-Tiles konfiguriert.

Control
Die Applikationslogik wird wie der Rest der Software in Java geschrieben und läuft auf einem Apache Tomcat Server.

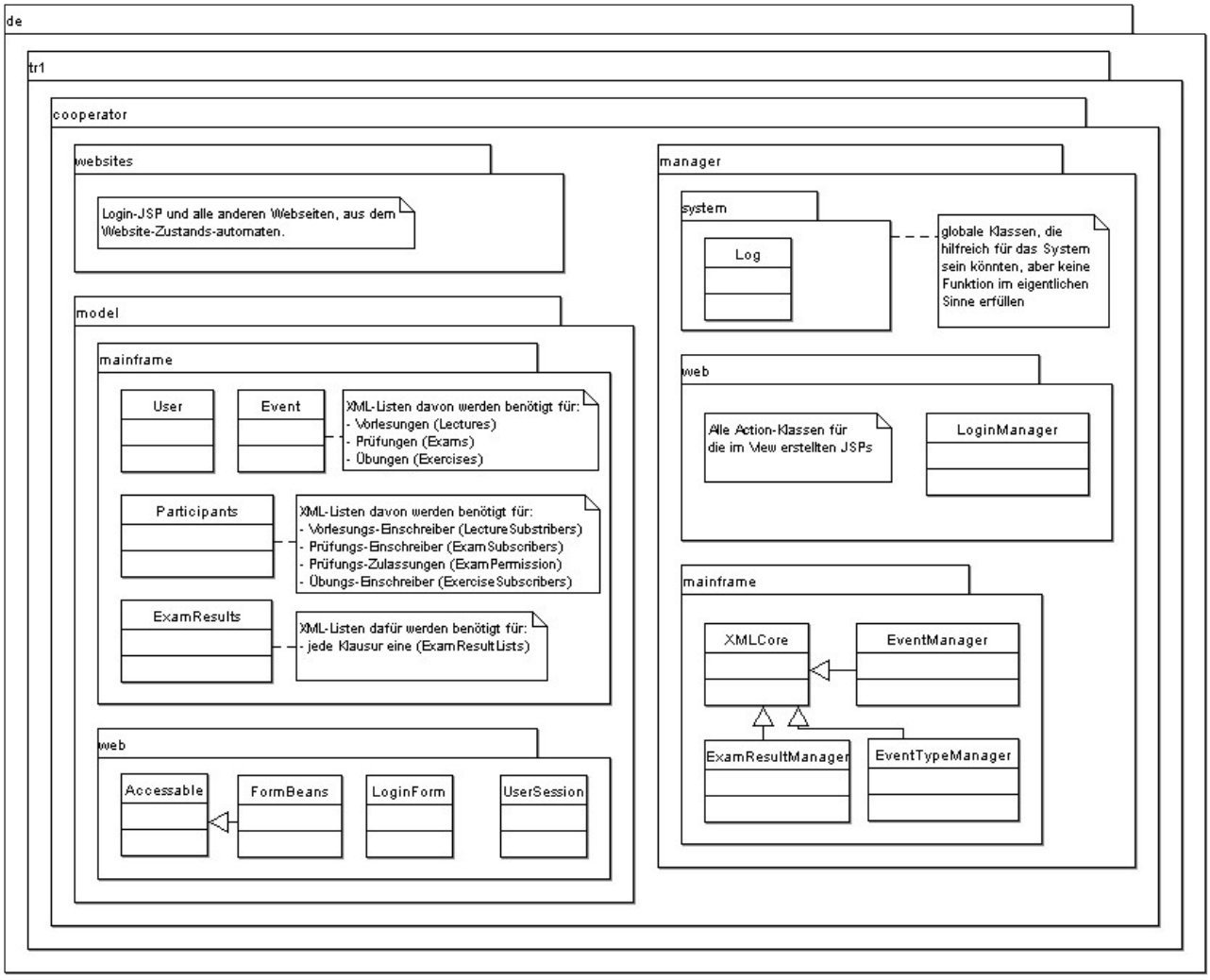
4 Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 Statisches Modell

Die Klassen der Software werden in hauptsächlich drei Packages aufgeteilt. Diese drei Packages entsprechen Model, View und Control aus dem MVC-Konzept und heißen *model*, *websites* und *manager*. Die Package Aufteilung wurde so gewählt um die Umsetzung des MVC-Konzeptes auch im Programmcode verfolgen zu können. Mit der Arbeit am Programmcode ist es durchaus möglich, dass weitere Packages hinzugefügt werden oder die Package Struktur sich grundsätzlich ändert.

Die einzelnen Pakete *manager* und *model* unterteilen sich wiederum in weitere Pakete, die nach Aufgabenbereichen aufgeschlüsselt wurden. So ist z.B. das Paket *manager.system* nur für System-Aufgaben zuständig, die aber nichts mit der eigentlich sichtbaren Arbeitsweise zu tun haben und nur System-Funktionen, bzw. Klassen bereitstellt, wie die Klasse Log, die stets genutzt wird um im Hintergrund Aktivitäten und Veränderungen für den Entwickler oder Administrator aufzuzeichnen.
Das Paket *manager.web* ist zuständig für die Bereitstellung der Action-Klassen, die für die Struts-generierten Webseiten und den Kontrollfluss notwendig sind. Zusätzlich bietet dieses Paket weitere Klassen an, die im Umgang mit Web-spezifischen Daten notwendig sind, wie etwa der *LoginManager*, der sich um die Verwaltung von eingeloggten Usern und Sessions beschäftigt.
Das wichtigste Unterpaket in *manager* ist *manager.mainframe*, da dies die Hauptverwaltung der Anwendung bereitstellt. Dort zu finden sind also alle Manager, die für die Datenhaltung der XML-Datenbank zuständig sind. Diese Klassen setzen alle auf einem Core auf, welches die Grund-Aufgaben bereitstellt, wie Laden und Aufarbeiten von XML-Dateien, Speichern dieser und Suchen in den XML-Strukturen. Alle Manager sind dabei als Singletons designt, die Großteile der Daten permanent im Speicher halten, damit die vielen Server-Anfragen durch die Webseiten möglichst zeitsparend erfolgen kann, da es sonst immer notwendig wäre die entsprechenden Dateien zunächst zu öffnen und aufzuarbeiten. Durch die Singletons wird aber nur eine einzelne persistente Instanz erzwungen, die als einzige Zugriffe gewährt. Das Prinzip Singletons gilt ebenfalls für die Klasse *LoginManager* und *Log*.
Das Paket *model* untergliedert sich in die zwei Unterpakete *model.mainframe* und *model.web*. Das Paket *model.web* enthält lediglich Beans, die zum Datenaustausch zwischen den Struts-Webseiten, bzw. den JSP-Dateien untereinander und in Kombination mit den ihren zugehörigen Action-Klassen aus *manager.web* oder aber für anderweitig Web-basierende Daten benötigt werden.
Das Paket *mode.mainframe* ist wiederum das wichtigste Paket, was die Applikation und ihre Daten anbelangt. In diesem Paket sind alle Daten enthalten, mit der die Anwendung hauptsächlich arbeitet und die es zu verwalten heißt. Ein kurzer Kommentar zur internen Speicherung ist im groben Diagramm ersichtlich.

Grobes Diagramm:



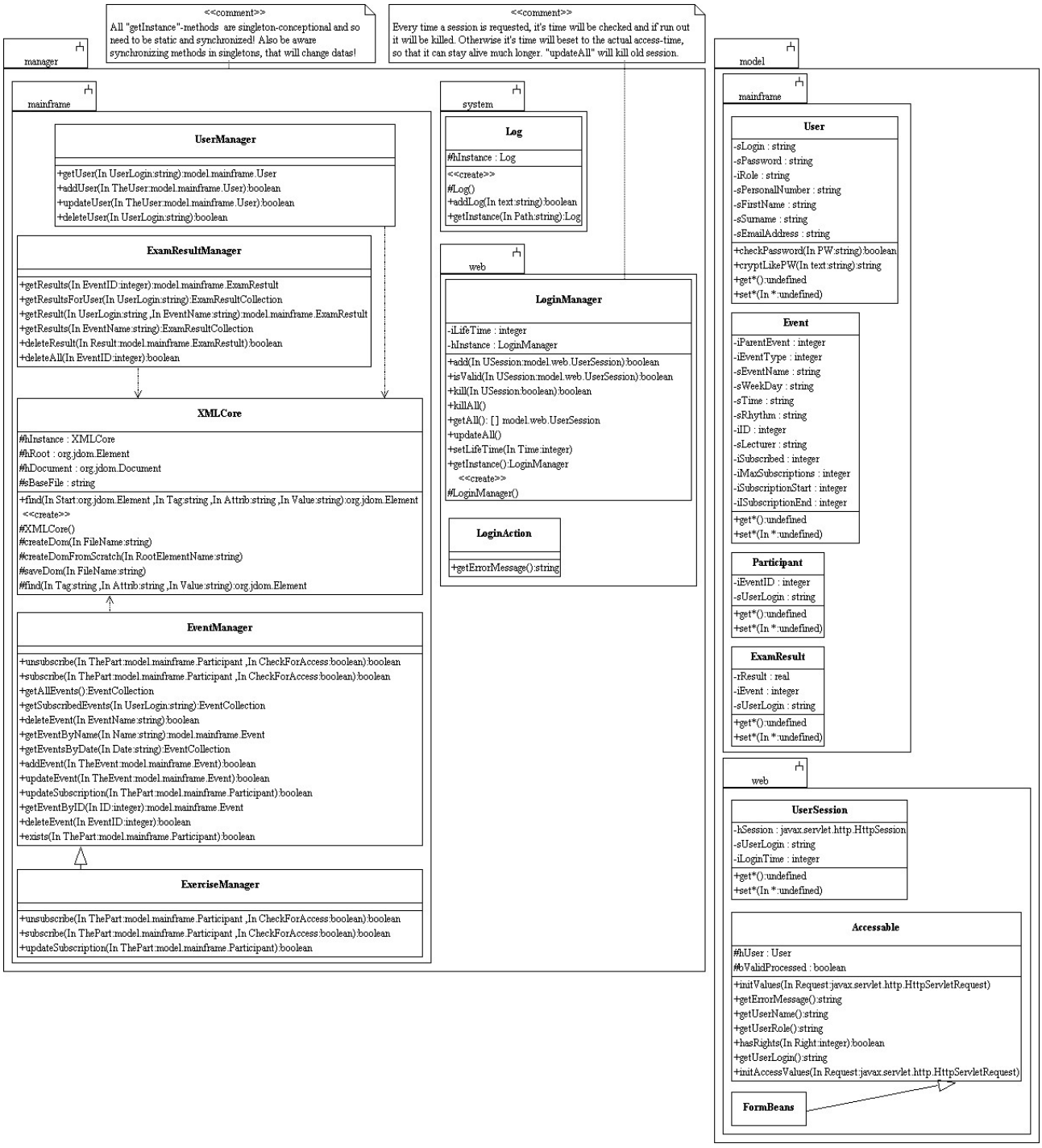
Das letzte Paket *websites* ist im eigentlichen Sinne gar kein Paket oder zumindest kein Java-Paket, da es sich hierbei nur um die Menge der JSP-Dateien handelt, die die auf den Managern und dem Modell beruhende Struktur und Verwaltung für den Benutzer des Systems als HTML-Seiten generiert. Interfaces zum Browser und somit die grafische Schnittstelle zum Benutzer ist einzig und allein hier zu finden.

Offensichtlich sind die Paketnamen mehrfach ähnlich, was den Bezug der Pakete zueinander ersichtlich machen soll. Die Pakete sind also nicht nur strikt nach Aufgaben und Zugehörigkeiten getrennt, sondern auch nach ihnen benannt worden.

Eine detaillierte Klassenstruktur mit grundlegenden Methoden oder Attributen ist folgende.

Mitglieder: Sebastian Kohl (SK), Mirko Schmidt (MS), Peter Matjeschk (PM), Carsten Lüdecke (CL), Michael Thiele (MT), Sven Fischer (SF), Ibrahim Osman (IO), Maik Müller (MM)

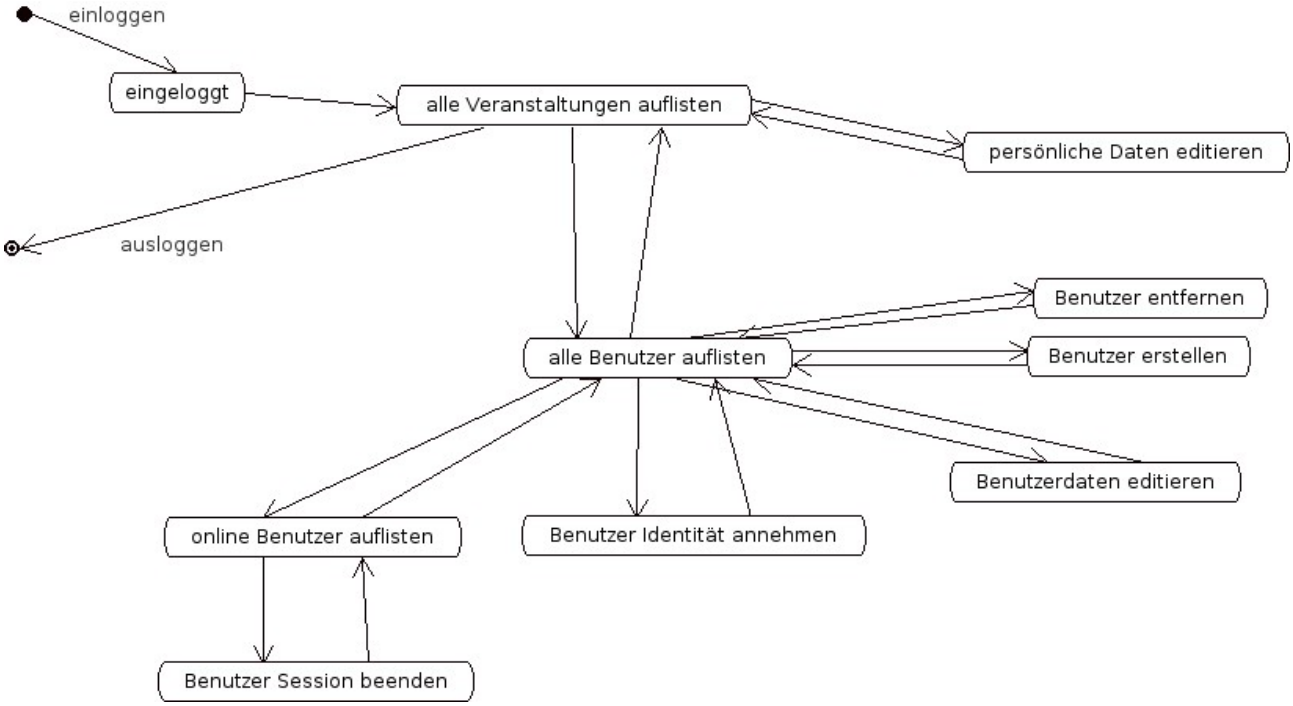
Detailliertes Diagramm:



4.2 Dynamisches Modell

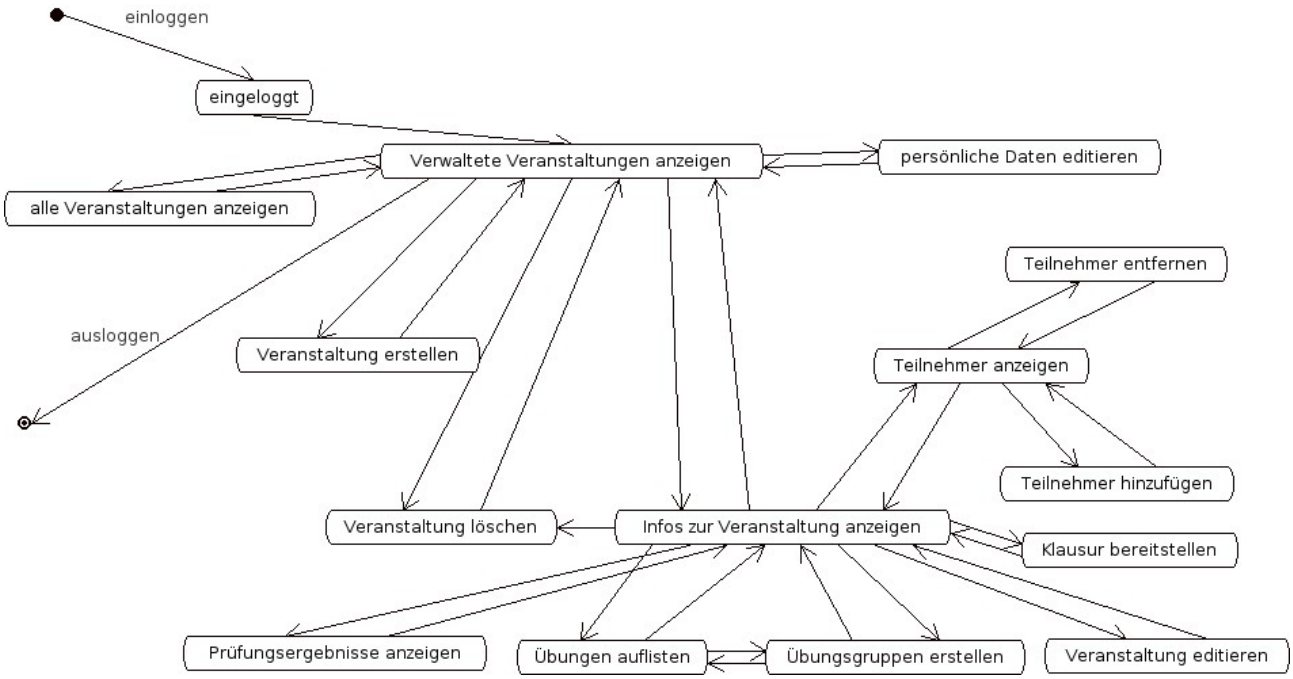
In den folgenden drei Zustands-Diagrammen werden einige Funktionen der einzelnen Rollen von Benutzern der Applikation dargestellt. Wobei die Zustands-Diagramme eher als Kontrollfluss Diagramme der Html-Seiten aufgefasst werden können. Die Zustandsübergänge stellen die "Klicks" auf Links dar, und sind deshalb um eine Übersichtlichkeit zu behalten nicht beschriftet.

Admin:

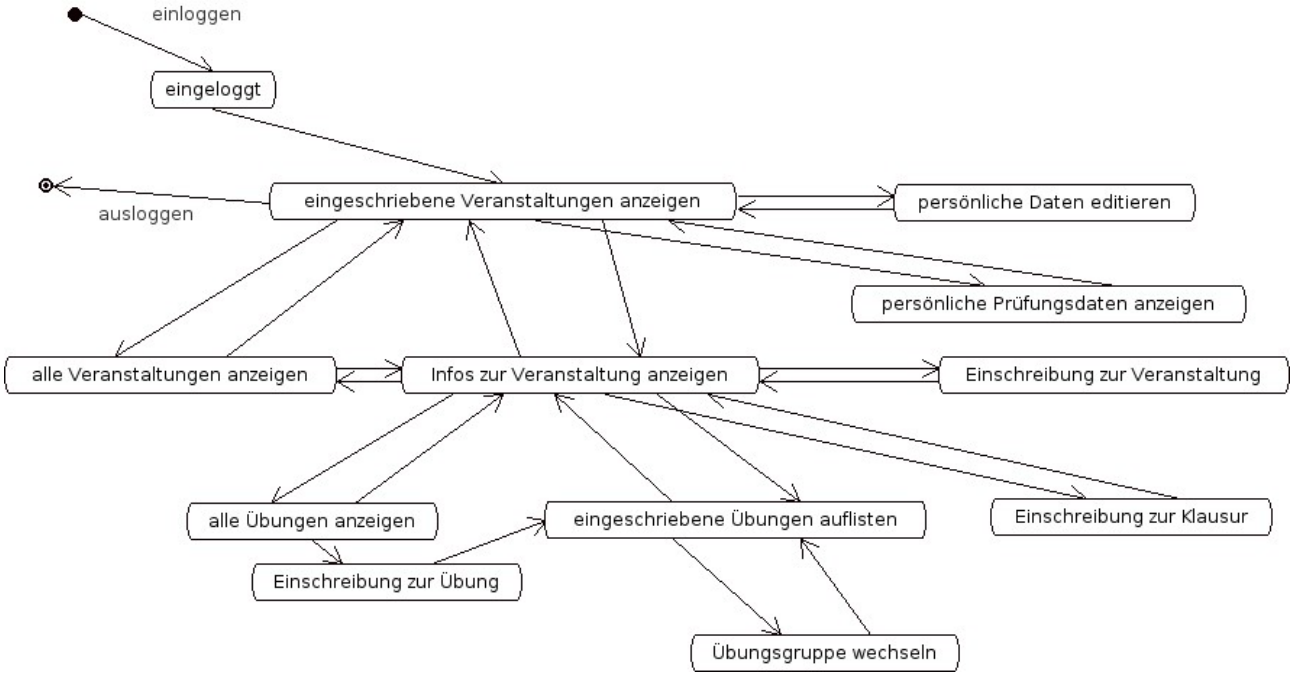


Der Admin hat zusätzlich zu den hier aufgeführten Möglichkeiten natürlich noch alle Möglichkeiten von Dozent und Student, da er sich jederzeit als ein beliebiger Nutzer ausgeben und dessen Identität annehmen kann. Das vereinfacht die Verwaltung des Administrators sehr stark, da die bereits vorhandenen Methoden benutzt werden können und keine zusätzlichen Methoden mehr benötigt werden. Natürlich gilt dies nur für den Rahmen dessen, was Dozent und Student selber verwalten dürfen. Darüber hinaus besitzt der Admin aber dennoch die gleichen Möglichkeiten, wie als eingeloggter Dozent oder Student gleichzeitig und erhält darüber hinaus noch weitere Optionen, wie etwa dass beim Ändern von Veranstaltungsdaten der lesende Dozent geändert werden könnte, Rollen von Benutzern beim Editieren von Benutzdaten oder eine Einschreibung für einen Benutzer dennoch möglich ist, obwohl gewisse Voraussetzungen nicht erfüllt sein könnten.

Dozent:



Student:



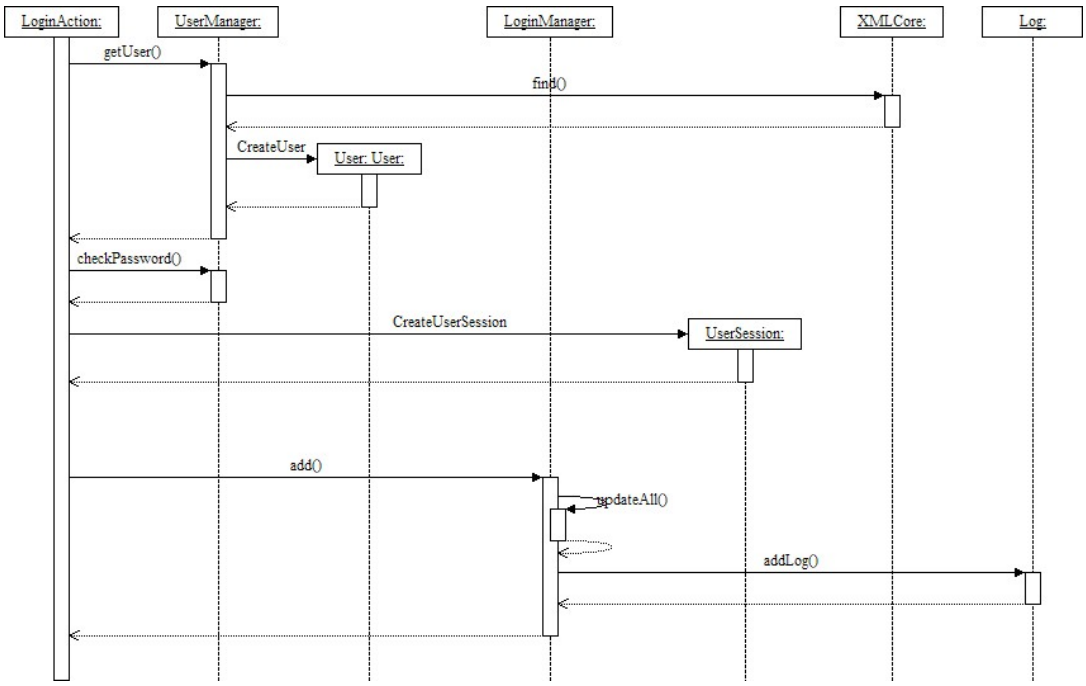
In den folgenden Sequenz-Diagrammen werden die Abläufe der Anwendungsfälle Login, Einschreibung und Bereitstellung grob beschrieben.

Mitglieder: Sebastian Kohl (SK), Mirko Schmidt (MS), Peter Matjeschk (PM), Carsten Lüdecke (CL), Michael Thiele (MT), Sven Fischer (SF), Ibrahim Osman (IO), Maik Müller (MM)

Sequenzdiagramme:

Login:

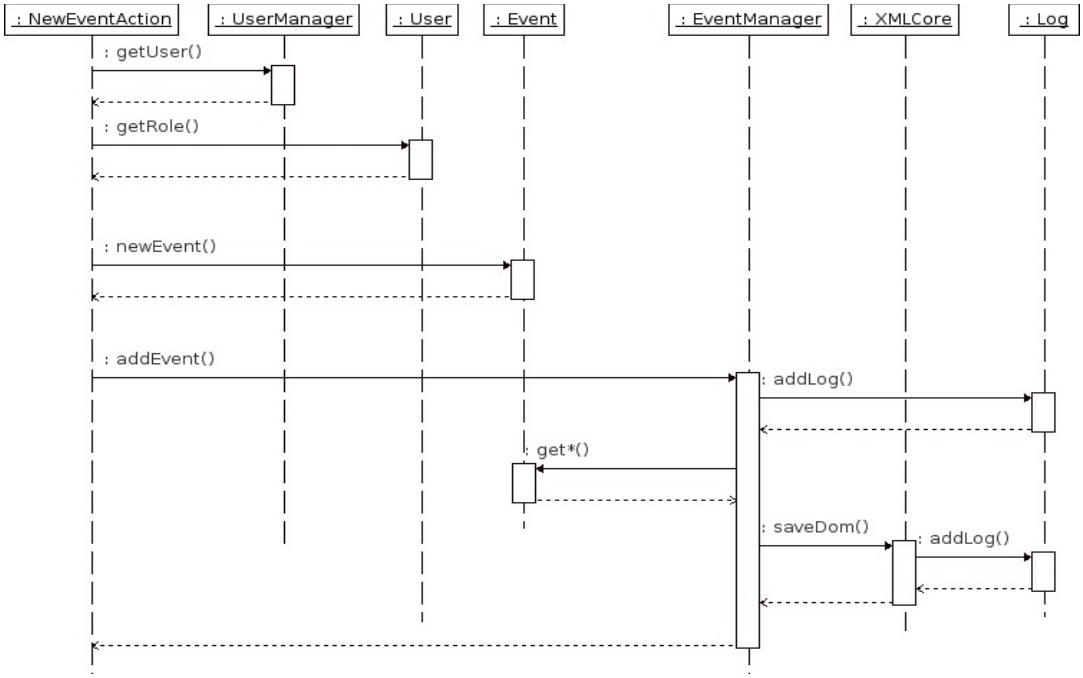
In diesem Sequenzdiagramm wird der Ablauf des Login dargestellt. Durch das betätigen des entsprechenden Button auf der Loginseite wird die Aktion gestartet. Zu Beginn fordert die LoginAction vom UserManager den User mit dem eingegebenen Benutzername an. Der UserManager hohlt sich diesen Benutzer aus der XMLCore und erstellt daraufhin eine Instanz der Klasse Nutzer mit den Daten aus der XMLCore. Dieser User wird der LoginAction zurückgegeben und das eingegebene Passwort wird mit dem Passwort des Users verglichen. Wenn alle Eingaben korrekt waren wird für den User eine Session angelegt. Nun aktualisiert der LoginManager alle vorhandenen Sessions und fügt die eben erstellt noch hinzu. Zu guter letzt wird der erfolgreiche Login in der Log-Datei gepeichert.



Mitglieder: Sebastian Kohl (SK), Mirko Schmidt (MS), Peter Matjeschk (PM), Carsten Lüdecke (CL), Michael Thiele (MT), Sven Fischer (SF), Ibrahim Osman (IO), Maik Müller (MM)

Bereitstellung:

In diesem Sequenz Diagramm wird der Ablauf des Anwendungsfall "eine Veranstaltung erstellen" dargestellt. Diese Sequenz wird von einer Instanz der Klasse NewEventAction angestossen.
Dieses Objekt holt sich zunächst vom UserManager ein User Objekt des Benutzers. Anhand des User Objektes wird dann die Rolle des Benutzers geprüft. Danach wird ein neues Event Objekt erstellt welches dem EventManager übergeben wird. Dieser loggt den Vorgang in eine Datei, und holt per get-Methoden alle erforderlichen Parameter des Event-Objektes(wegen der Übersicht im Diagramm get*()).
Danach wird das Event in den XML-Dateien abgespeichert und der Vorgang wird wieder geloggt.



Einschreibung:

Ein Student kann sich in die Teilnehmerliste einer beliebigen Veranstaltung eintrage, sofern er zu dieser zugelassen ist und die Veranstaltung noch nicht vollständig belegt ist. Es gibt drei Veranstaltungstypen: Vorlesung, Übung und Prüfung.
Will er sich einschreiben, so gelangt er von der Webseite, die nähere Informationen zur Veranstaltung zur Verfügung stellt, über einen Link zum Anmeldeformular für diese Veranstaltung, welche durch die *subscribe.jsp* verwirklicht wird. Nach Ausfüllen und Abschicken des Formulars werden von der *SubscribeAction.class* vier Bedingungen geprüft:

1. Ist der User bereits in die Teilnehmerliste eingetragen
2. Erfolgt die Einschreibung innerhalb des vorgegebenen Einschreibungszeitraums
3. Ist die Maximale Belegung für die Veranstaltung schon erreicht
4. Erfüllt der Student die Zulassungsvorraussetzungen zu dieser Veranstaltung

Ist eine dieser Bedingungen nicht erfüllt, wird der Einschreibungsvorgang abgebrochen und der User informiert. Sind alle Bedingungen erfüllt, so wird der Student in die Einschreibungsliste eingetragen, diese gespeichert und das *Logfile* aktualisiert. Der User wird über den Erfolg benachrichtigt.