Designstudie des Java-Projekts WebAssign

1. Allgemeines

WebAssign ist ein WWW-basiertes Übungssystem, das sowohl Studierende als auch Lehrende unterstützt.
Insbesondere werden Aufgabenerstellung, Entgegennahme der Lösungen, Korrektur und Verteilung über das Internet vorgenommen. Dabei werden die Nutzer in verschiedene Gruppen eingeordnet, die jeweils über spezifische Zugangsrechte und Möglichkeiten der Datenmanipulation verfügen:

- Studierende
- Korrektoren
- Betreuer

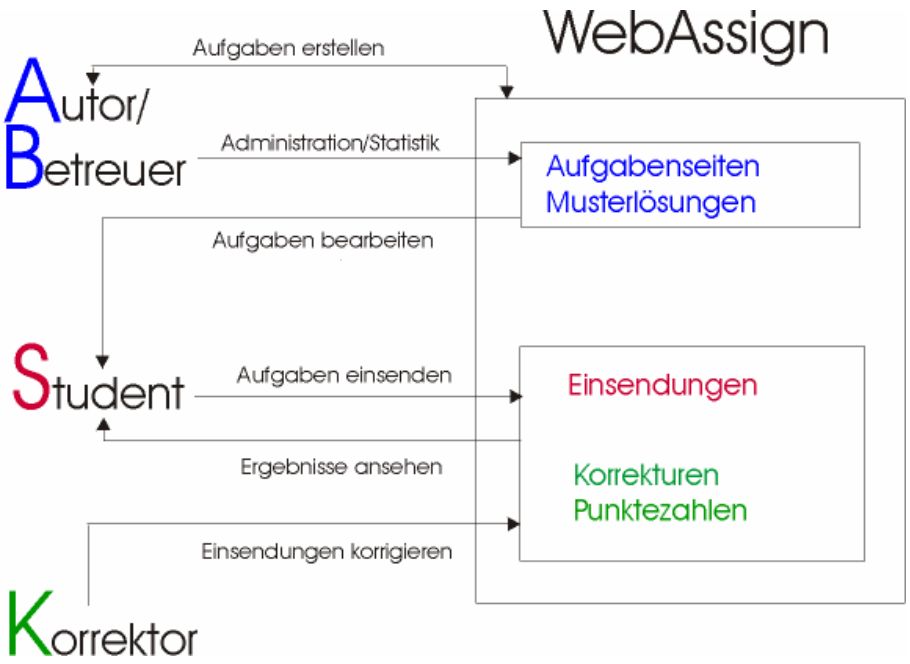


Abb. 1.1: Hauptfunktionalitäten der Benutzergruppen

Da WebAssign ein web-basiertes System ist, wird für den Betrieb ein Web-Server (http) benötigt, welcher in der Lage sein muß Java-Servlets über eine Servlet-Engine zu verarbeiten. Zur Speicherung der Daten wird ein relationales Datenbanksystem verwendet, welches Treiber für JDBC-Verbindungen anbietet.

2. Produktübersicht

WebAssign unterstützt alle Phasen des Übungsbetriebs, von der Erstellung der Aufgaben, über ihre Lösung, bis hin zur Korrektur. Deshalb ist es erforderlich die Benutzer in verschiedene Gruppen einzuteilen und sie mit individuellen Rechten zu versehen. WebAssign unterscheidet dabei zwischen Betreuern, Studierenden und Korrektoren. Das Anmelden auf dem Server erfolgt mittels der Eingabe von Benutzerkennung und Passwort, was für jeden Anwender eine personalisierten Zugang ermöglicht.

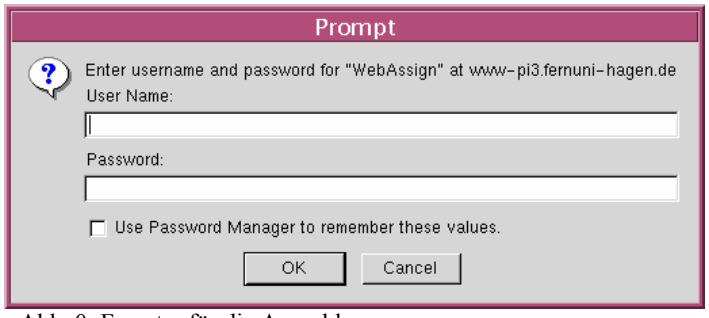


Abb. 0: Fernster für die Anmeldung

Nach der Authentifizierung darf ein Benutzer nur soweit Informationen sehen und verändern, wie dies für seine Aufgaben erforderlich ist. Insbesondere die persönlich konfigurierten Seiten, z.B. die persönlichen Ergebnisse eines Studierenden oder die zu korrigierenden Einsendungen eines Korrektors, sind nur für die entsprechende Person zugänglich.

Abb.1 zeigt die wichtigsten Seiten, die die einzelnen Benutzergruppen erreichen könne:

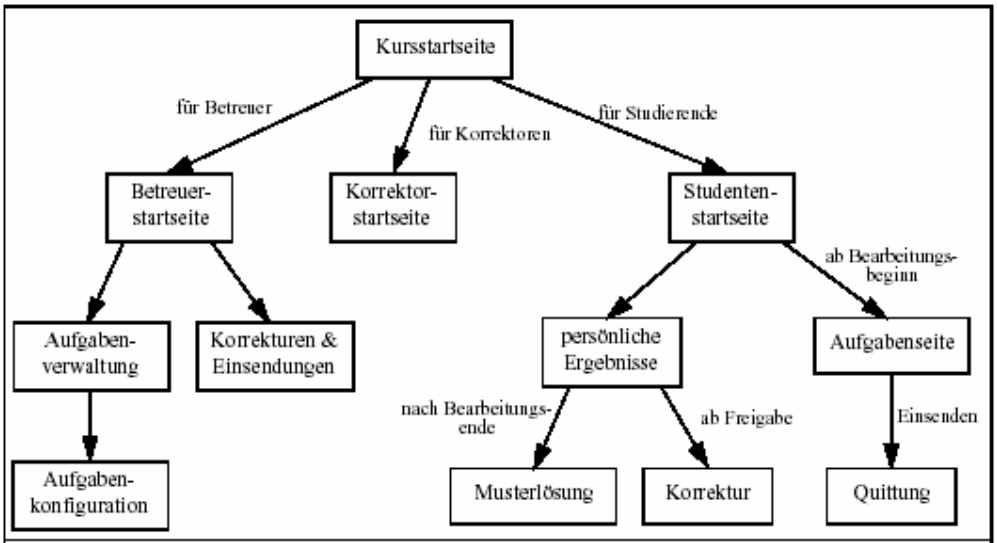


Abb. 1 Struktur der wichtigsten WebAssign-Seiten

2.1 Anwendungen für Studenten

Nach dem Login wird der Student direkt auf die Studentenstartseite weitergeleitet. Von hier aus hat er zentralen Zugang auf alle für ihn relevanten Funktionen.

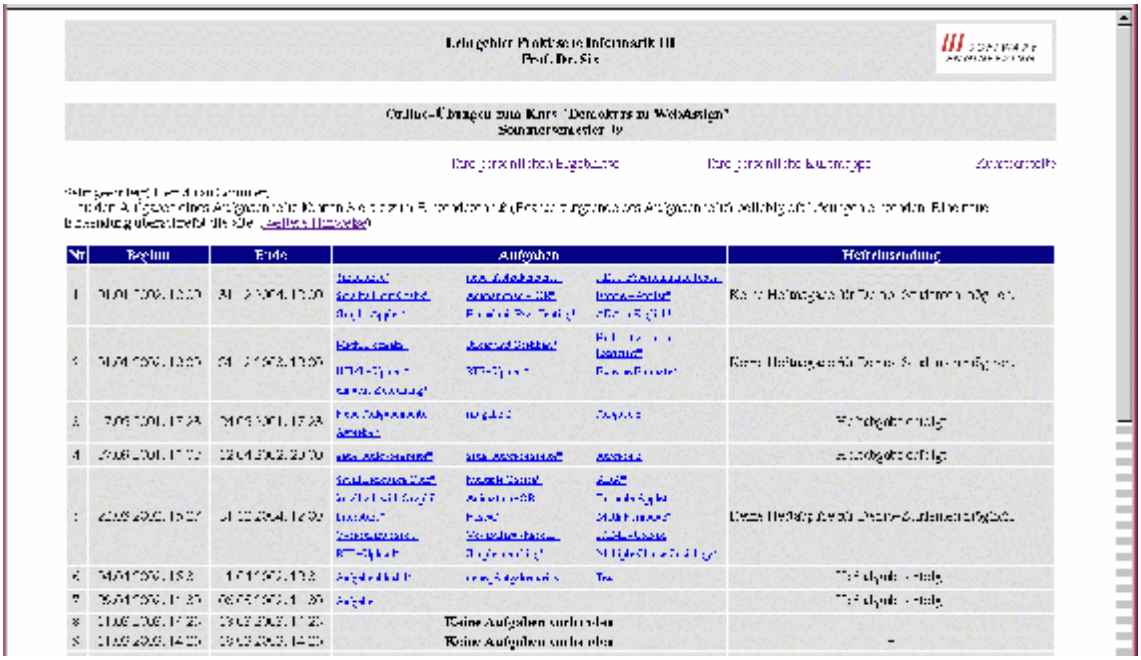


Abb. 3: Studentenstartseite

Unmittelbar angezeigt werden auf dieser Seite nach Aufgabenheften geordnet eine Übersicht über alle schon bearbeiteten, sowie noch zu bearbeitenden Aufgaben. Über die jeweiligen Aufgaben-Links gelangt man zum eigentlichen Aufgabentext, der auf einer gesonderten Seiten bearbeitet werden kann. Nach der Fertigstellung der Aufgaben bestätigt der Student die Eingaben mittels Heft schließen. Von nun an ist keine weitere Bearbeitung mehr möglich.

Von der Startseite aus gelangt der Student zu seinen persönlichen Ergebnissen und Kursmappen. In seinen Ergebnissen kann sich der Student über den Korrekturstatus seiner Aufgaben, sowie die erreichten Punkte informieren. Sofern vorhanden, befindet sich auf dieser Seite auch ein Link zur Musterlösung.

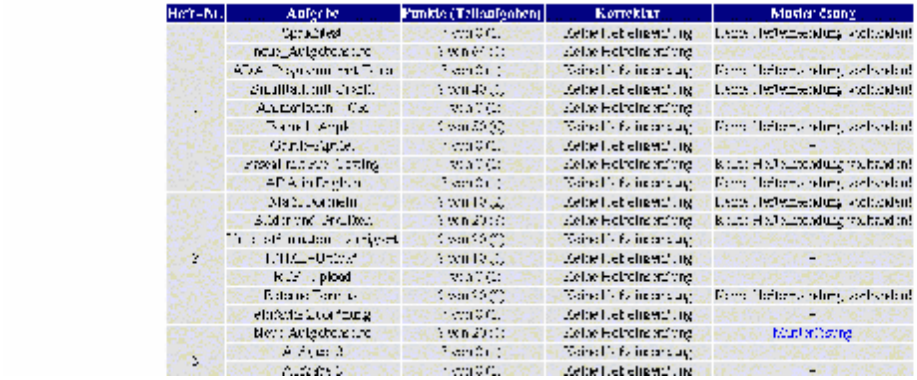


Abb. 4: Übersicht über korrigierte Aufgaben

Die Seite Kursmappe ist eine Alternative zur Aufgabenübersicht und der Ergebnisseite. Hier findet man im linken Frame eine Kursmappe. Wenn man auf den Ordner eines Aufgabenheftes klickt und anschließend auf den Ordner einer Aufgabe, kann man sich den Aufgabentext, die korrigierte Einsendung oder die Musterlösung anzeigen lassen bzw. beim Aufgabentext diesen auch bearbeiten. Die ausgewählten Seiten erscheinen jeweils im rechten Frame.

2.2 Anwendungen für Korrektoren

Für Korrektoren besteht die Wahl zwischen online- und offline-Arbeit, wobei für die offline-Korrektur ein entsprechender Server auf dem lokalen PC installiert werden muss.
Auf der Korrektorstartseite findet der Korrektor einen Überblick über die ihm zugeordneten Aufgaben. Sortiert nach Aufgabenheften wird angezeigt, wie viele Aufgaben schon bearbeitet sind oder eben noch korrigiert werden müssen.
Aus einer Tabelle kann der Korrektor nun die zu prüfenden Einsendungen öffnen , korrigieren und auf den Status korrigiert setzen.

2.3 Anwendungen für Betreuer

Analog zur Studentennavigation gibt es auch für die Betreuer eine Startseite, welche die zentrale Anlaufstelle für alle administrativen Tätigkeiten bildet.

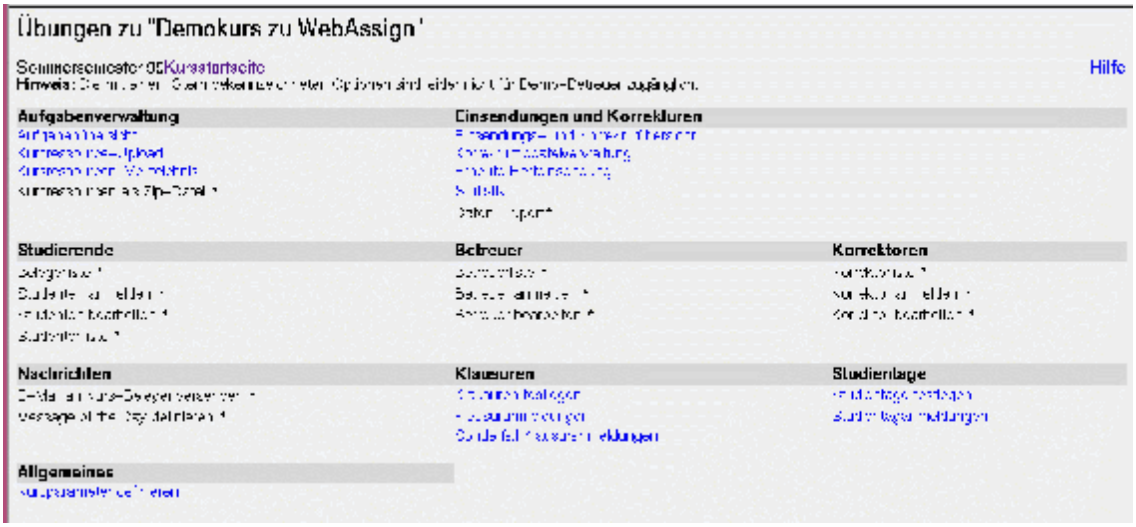


Abb. 5: Betreuerstarseite

Wie in Abb. 5 ersichtlich kann der Betreuer zwischen folgenden Funktionalitäten wählen:

- Verwaltung der Aufgaben
- Einsendungen und Korrekturen bearbeiten
- Studierende, Betreuer und Korrektoren verwalten
- Nachrichten versenden
- Klausuren editieren und
- Studientage organisieren, sowie
- die Definition von Kursmappe

Exemplarisch werden im folgenden drei der weiterführenden Links näher erläutert.

Über den Link Aufgabenübersicht gelangt der Betreuer zu einer Seite, die ihm die Verwaltung der Aufgabenhefte ermöglicht. Er kann dort neue Aufgaben erzeugen, alte löschen oder bestehen verändern.

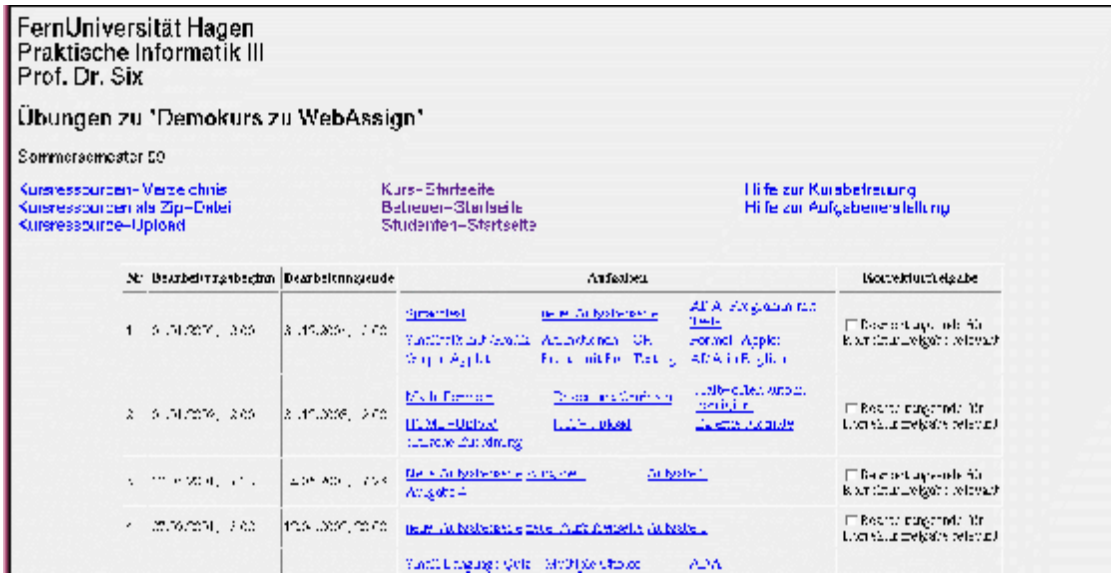


Abb. 6: Seite „Aufgabenübersicht“

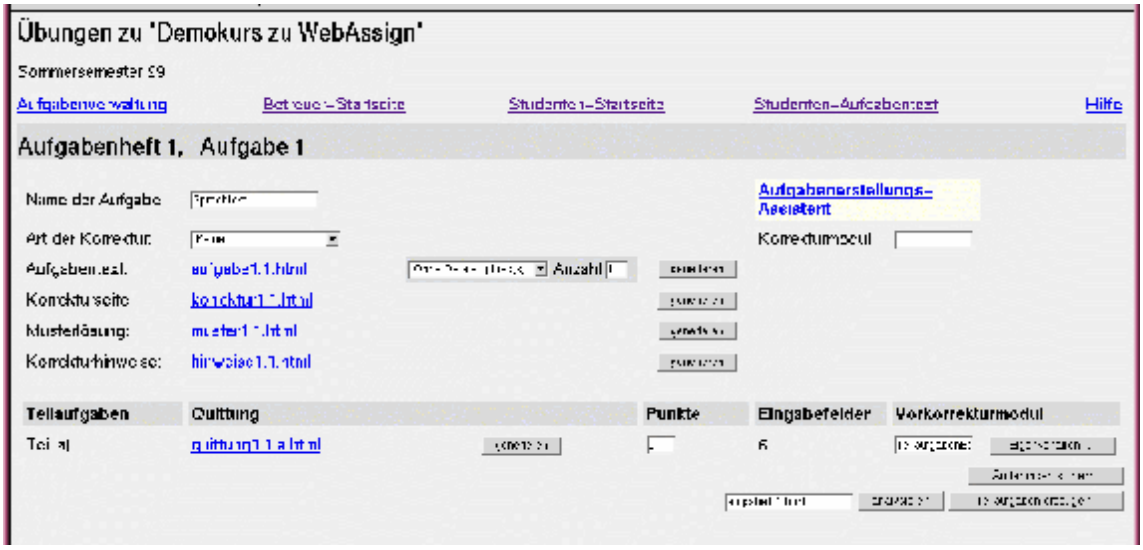


Abb. 7: Bearbeitung der Aufgaben zu „Sprachtext“

Auf der Seite Einsendungen und Korrekturen erhält der Betreuer einen Überblick über die Daten der von ihm gestellten Aufgaben. Der Tabelle kann er entnehmen:

- wie viele Aufgabenhefte existieren
- wie viele von diesen noch nicht bearbeitet/verteilt sind
- welcher Korrektor einem Aufgabenheft zugeordnet ist und
- wie der Stand der Bearbeitung ist

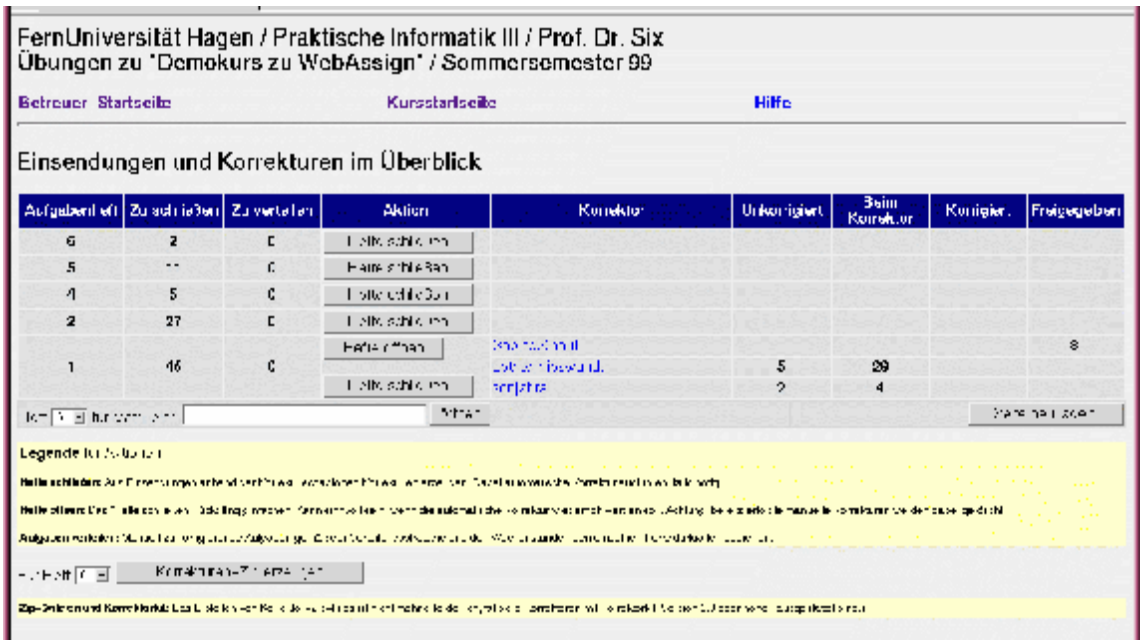


Abb. 8: Seite „Einsendungen und Korrekturen“

Der Link „Klausuren festlegen“ führt den Betreuer auf folgende Seite:

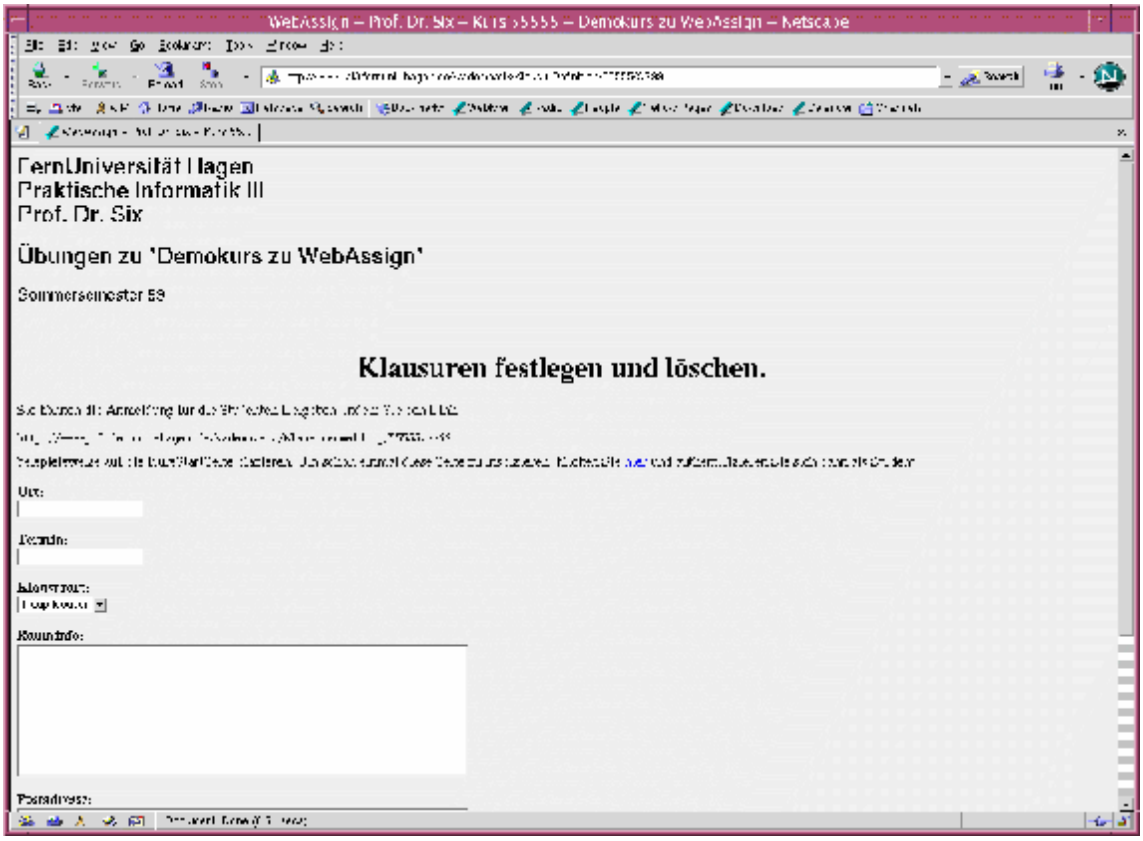


Abb. 9: Seite zum Festlegen und Löschen von Klausuren

Hier kann der Betreuer alle relevanten Daten für die Klausur angeben und die Daten anschließend veröffentlichen.

3. Grundsätzliche Design-Entscheidungen

3.1 Überblick

In der nachfolgenden Abbildung ist der wesentliche Datenfluss veranschaulicht.
Jede Anforderung an das System beginnt mit einem Request an das zentrale Servlet (vgl. 3.2.1 Servletarchitektur) „webassign“ (links oben). Die Daten des Requests werden gekapselt und an die Klasse „WebApplicationServer“ weitergeleitet. Handelt es sich bei dem Request um den Versuch sich einzuloggen, wird geprüft, ob die Daten korrekt sind. Ist der Benutzer bereits eingewählt kann der Service direkt ausgeführt werden.

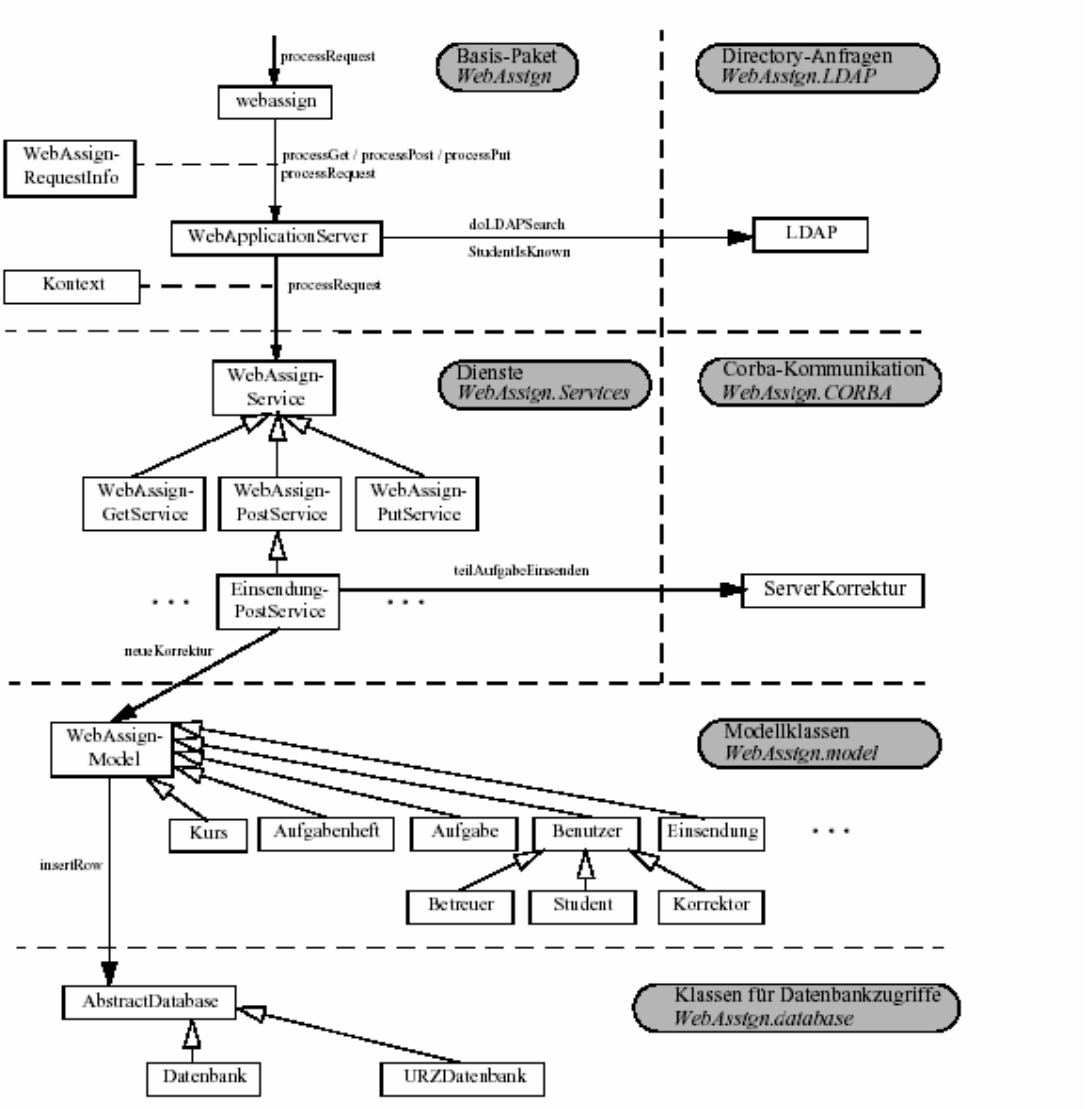


Abb. 3.1: Grundlegende Klassenstruktur von „WebAssign“

Die Basisfunktionalitäten für alle Service-Klassen werden von WebAssignService bereitgestellt. Dazu zählen:

- Aufbau des Antwortstromes
- Expansion der darin enthaltenen Variablen
- Setzen von Content-Type und –Length

Abhängig von den Anforderungen des Users werden die Services wie folgt gegliedert:

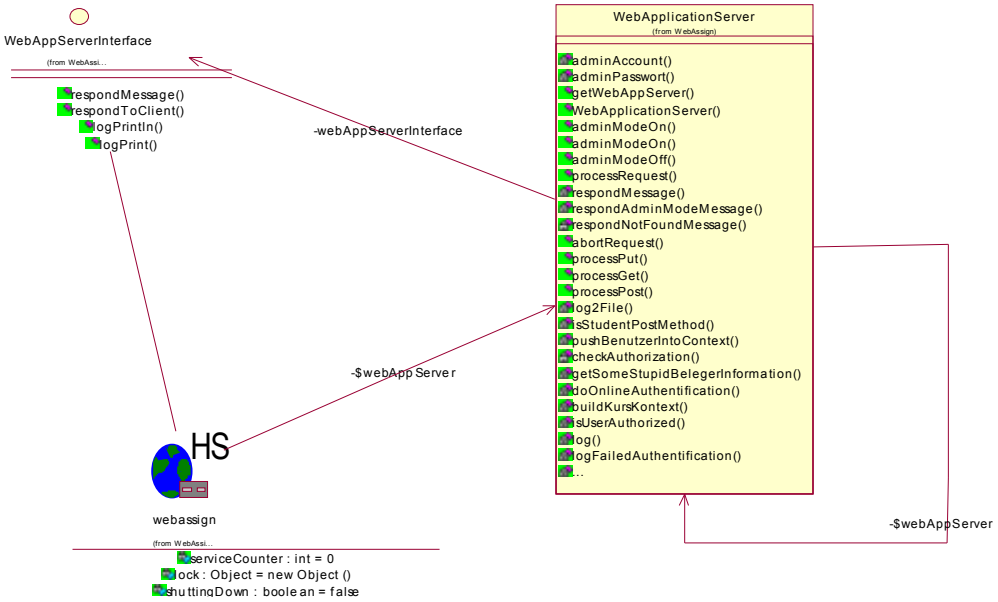
- Get-
- Post-
- Put-Service

Nähere Informationen zu den einzelnen Services befinden sich im Abschnitt 4. Packet- und Klassenstruktur, sowie in den 3.4 Grundlegende Designentscheidungen bei den Services.

3.2 ... in der Architektur

3.2.1 Servletarchitektur

Um Java-Programme über einen Web-Server aufrufen zu können, wird die Java-Servlet-Architektur verwandt, welche die Servlets auf der Server- und nicht auf der Client-Seite ausführt (im Gegensatz zu Applets).
Der vom Benutzer erzeugte Request (Daten in WebAssignRequestInfo gekapselt) wird über das von der Servlet-Engine aufgerufene Servlet „webassign“ an den WebApplicationServer geleitet, wo die URL analysiert, der aufgerufene Dienst extrahiert wird und die Zugangsdaten des Users überprüft werden. Die Attribute des gerade agierenden Benutzers befinden sich im Kontext-Objekt.
Die im folgenden benötigten Basisfunktionalitäten befinden sich für alle Klassen in WebAssignService. Untergliedert werden die Services in Get-, Post- und Put-Services.



WebAssignView

3.2.2 ... in Bezug auf Schnittstellen

Der WebAssign-Server kann sowohl als Dienstgeber als auch als Dienstnehmer fungieren. Wird der Server als Dienstnehmer eingesetzt, kommuniziert er mit sog. EinsendungsServern, Korrekturmoduln in z.B. Java. Somit ist es möglich den Korrekturbetrieb an die individuellen Wünsche und gegebene Voraussetzungen der Korrektoren anzupassen. Da diese individuellen Programme normalerweise nicht auf dem Host installiert sind, schickt der Server eingegangene Lösungen an den EinsendungsServer und erhält die fertige automatische Korrektur zurück.
Ein Beispiel für die Funktion als Dienstgeber ist das Offline-Korrektorkit. Hierbei fordert der WebAssignServer neue Korrekturen für den jeweiligen Korrektor an oder lädt korrigierte Einsendungen herauf.

3.3 ... bei der Datenhaltung

Das Programm ist so konzipiert, dass es mit den gängigen Datenbanken kooperiert. Dies ermöglicht es dem Betreiber das System optimal an seine Bedürfnisse und Kenntnisse anzupassen.

Es werden nur solche Datentypen verwendet, die mit verschiedenen Datenbanksystemen (Informix, MySQL) und DB-Anbindungen (JDBC, proprietäre) unverändert eingesetzt werden können. Deshalb werden z.B. boole'sche Felder durch den Typ INTEGER (0=false, 1=true) und Datumsfelder durch CHAR(11) realisiert.

3.4 ... bei den Diensten

WebAssign bietet verschiedene Funktionalitäten an, genannt Dienste. WebAssign-Dienste unterscheiden sich hinsichtlich der Zielbenutzergruppe und Aufrufart. Ein Dienst kann für Studenten, Betreuer, Korrektoren oder für den WebAssign-Administrator vorgesehen sein. Bei der Aufrufart wird zwischen GET-, POST- und PUT-Services unterschieden.
GET-Services liefern eine vom WebAssign-System generierte HTML-Seite zurück. Ihre Benutzung erfolgt entweder direkt über die URL-Zeile des Browsers (z.B. Anfordern einer Kursstartseite) oder als Link innerhalb einer anderen Seite (z.B. Verweis auf die persönliche Ergebnisseite eines Studierenden). Anhand der gewählten URL identifiziert WebAssign den Service und lädt eine Schablone, d.h. ein Grundgerüst für die zu generierende Seite. Anhand der Authorisierungsinformation, die der Benutzer eingegeben hat und die der Browser bei jedem Request mitliefert, bestimmt WebAssign einen Kontext und expandiert dementsprechend die in der Schablone enthaltenen Variablen. Das Ergebnis wird an den Browser zurückgliefert.
POST-Services sind dafür verantwortlich, an den Server gesandte Formulardaten anzunehmen, weiterzuverarbeiten und schließlich eine Bestätigungsseite zurückzuliefern, die nach dem selben Prinzip ensteht wie bei GET-Services, allerdings in der Regel ohne Verwendung einer Schablone. Der Benutzer trägt die URL eines POST-Services in das *Action*-Attribut eines HTML-*Form*-Tags ein.
PUT-Services dienen zur Weiterverabeitung von ganzen Dateien, die ins WebAssign-System hochgeladen werden. Ihr Aufruf erfolgt wie bei den POST-Services mit dem Unterschied, dass das Formular Felder des Types *File* enthält und das *ENCTYPE*-Attribut.
Details dazu, wie die zu den Services gehörenden URLs gebildet werden und wie diese in eine HTML-Seite eingebunden werden können, finden Sie im Benutzerhandbuch. In Tabelle 2 auf

3.5 .. bei den Schnittstellen

Ein wichtiger Bestandteil des Programms sind seine Schnittstellen auf der Applikationsebene. Auf dieser arbeitet der WebAssign-Server mit anderen Programmmoduln zusammen. Dabei fungiert es entweder als Dienstnehmer oder als Dienstgeber.
Ein Beispiel für die Funktion als Dienstnehmer ist die Kommunikation mit EinsendungsServern, Korrokturmoduln.
Als Dienstgeber fungiert der Server für das Offline-Korrektorkit, welches auf dem privaten Rechner eines Korrektors installiert ist (vgl. 2.2 Anwendungen für Betreuer).
Sämtliche Daten werden in einer relationalen Datenbank gespeichert, welche mittels JDBC angesprochen wird.
Fernen bietet WebAssign eine Schnittstelle zu CORBA

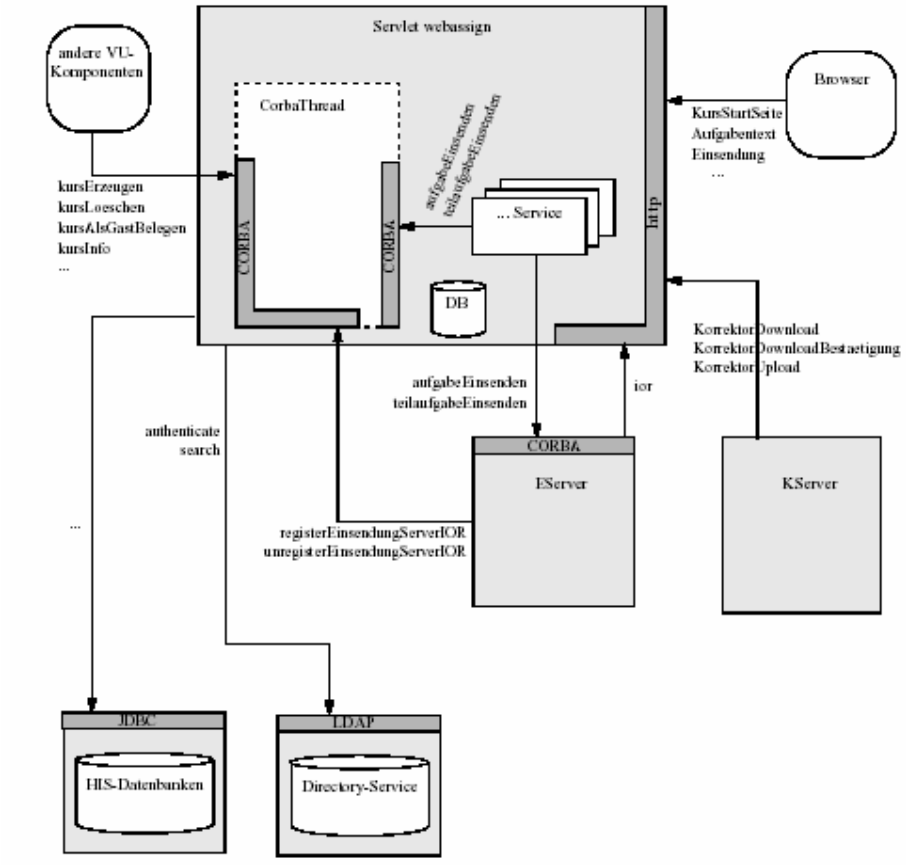


Abb. 3.2: Schnittstellen des WebAssign-Systems

4. Paket- und Klassenstruktur

- **WebAssign**:

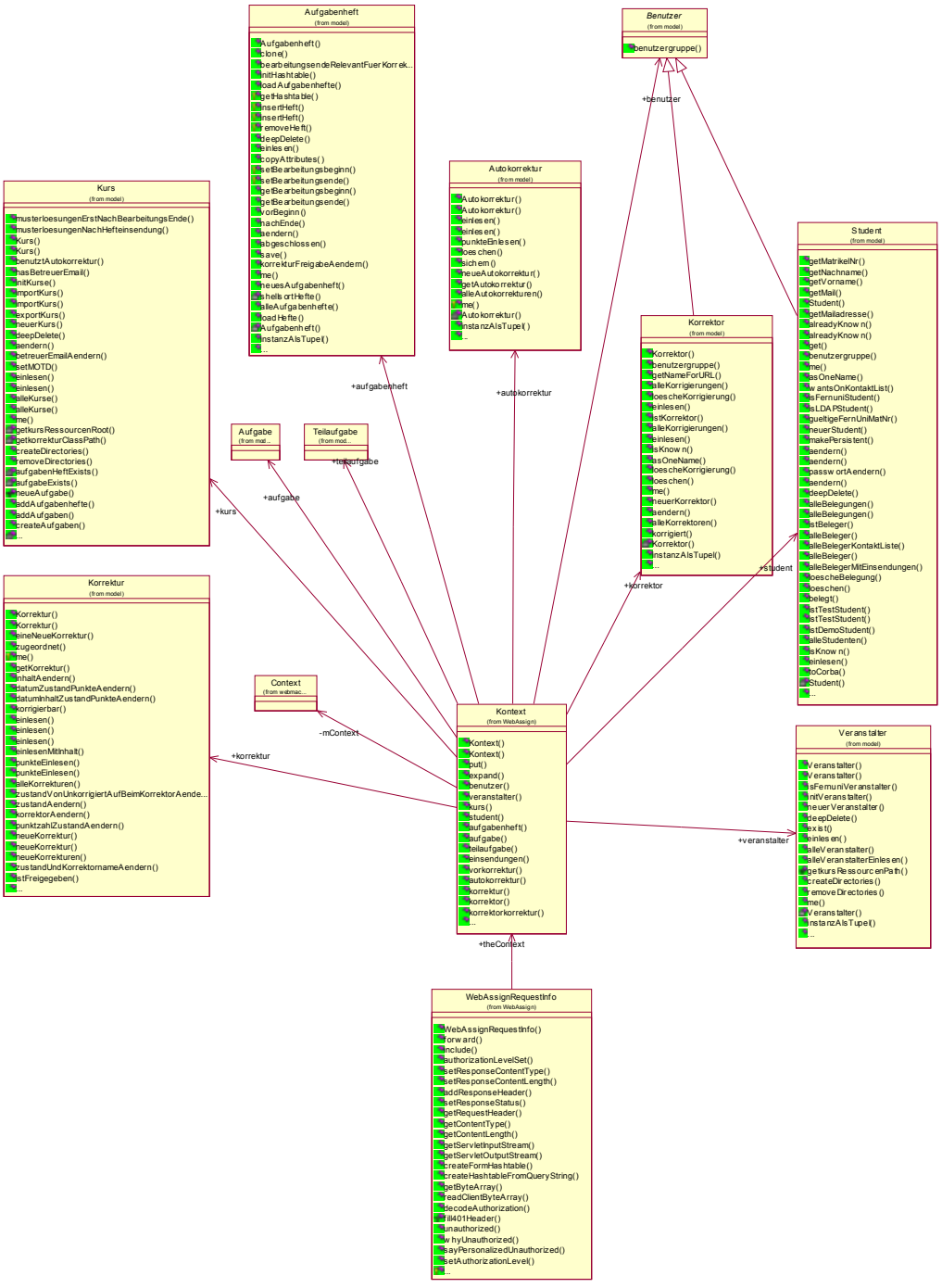
Im Package WebAssign sind die Grundfunktionalitäten des Programms untergebracht. So stellt es Klassen bereit, die das zentrale Servlet implementieren, welches sich um die Durchführung und die Weiterleitung von nutzerseitigen Serviceanfragen kümmert (mywebassign**)**. Der gewünschte Service wird in einer weiteren Klasse (WebApplicationServer) mittels der URL ermittelt und aufgerufen. Die Klasse Kontext speichert alle wichtigen Informationen aus der Datenbank, die auf einer html-Seite benötigt werden, in einem Objekt, das u.a. wie folgt strukturieret sein kann:

Kontextname	**bekannte Daten**
Benutzer	Benutzer
Veranstalter	Benutzer, Veranstalter
Kurs	Benutzer, Veranstalter, Kurs
Student	Benutzer, Veranstalter, Student
Aufgabenheft	Benutzer, Veranstalter, Kurs, Aufgabenheft
Aufgabe	Benutzer, Veranstalter, Kurs, Aufgabenheft, Aufgabe
Teilaufgabe	Benutzer, Veranstalter, Kurs, Aufgabenheft, Aufgabe, Teilaufgabe

KontextView:
Die Klasse *WebAssignRequestInfo* sorgt für die Darstellung/ Bereitstellung der Informationen über die Webpage bzw. Programmfenster. Um die Daten im richtigen Zusammenhang wiederzugeben, nutzt diese die Klasse *Kontext*, sie weiß, je nach dem welcher Benutzer angemeldet ist, was wichtig ist. *Kontext* arbeitet wie ein Vermittler/ Sammler, es erfragt notwendige Informationen und Services über die angeschlossene Klassen, wie *Student*, *Kurs* und *Veranstalter*, die ihrerseits alle Infos aus der Datenbank holen die einen auf sie zugeschnittene Sicht besitzt. Je nach Zusammenhang übergibt *Kontext* diese Inhalte an *WebAssignRequestInfo* für die Webpage bzw. das Programmfenster.

Außerdem führt die Klasse WebAssignAdminThread administrative Tätigkeiten parallel zu laufenden Betrieb von WebAssign aus. Dies umfasst sowohl die Garbage-Collection, wie auch das Prüfen der Forumseite.

-> Graphik siehe nächste Seite


Aufgabenheft
(from model)
Aufgabenheft()
clone()
bearbeitungsendeRelevantFuerKorrek...
initHashtable()
loadAufgabenhefte()
getHashtable()
insertHeft()
insertHeft()
removeHeft()
deepDelete()
einlesen()
copyAttributes()
setBearbeitungsbeginn()
setBearbeitungsende()
getBearbeitungsbeginn()
getBearbeitungsende()
vorBeginn()
nachEnde()
aendern()
istGeschlossen()
save()
korrekturFreigabeAendern()
name()
neuesAufgabenheft()
sheilsortHefte()
alleAufgabenhefte()
loadHefte()
Aufgabenheft()
instanzAlsTupel()
...
Benutzer
(from model)
benutzergruppe()
+benutzer
Autokorrektur
(from model)
Autokorrektur()
Autokorrektur()
einlesen()
einlesen()
punkteEinlesen()
loeschen()
sichern()
neueAutokorrektur()
getAutokorrektur()
alleAutokorrekturen()
name()
Autokorrektur()
instanzAlsTupel()
...
Kurs
(from model)
musterloesungenErstNachBearbeitungsEnde()
musterloesungenNachHefteinsendung()
Kurs()
Kurs()
benutzteAutokorrektur()
hasBetreuerEmail()
initKurse()
importKurs()
importKurs()
exportKurs()
neuerKurs()
deepDelete()
aendern()
betreuerEmailAendern()
setMOTD()
einlesen()
einlesen()
alleKurse()
alleKurse()
name()
getkursRessourcenRoot()
getkorrekturClassPath()
createDirectories()
removeDirectories()
aufgabenHeftExists()
aufgabeExists()
neueAufgabe()
addAufgabenhefte()
addAufgaben()
createAufgaben()
...
Aufgabe
(from mat...
Teilaufgabe
(from mod...
+teilaufgabe
+aufgabenheft
+autokorrektur
+aufgabe
+kurs
Korrektor
(from model)
Korrektor()
benutzergruppe()
getNameForURL()
alleKorrigierungen()
loescheKorrigierung()
einlesen()
istKorrektor()
alleKorrigierungen()
aendern()
einlesen()
isKnown()
loescheKorrigierung()
loeschen()
name()
neuerKorrektor()
aendern()
alleKorrektoren()
korrigiert()
Korrektor()
instanzAlsTupel()
...
+korrektor
Student
(from model)
getMatrikelNr()
getNachname()
getVorname()
getMail()
Student()
getMailadresse()
alreadyKnown()
alreadyKnown()
get()
benutzergruppe()
name()
isKnOneName()
wantsOnKontaktList()
isFernuniStudent()
isLDAPStudent()
gueltigeFernUniMatNr()
neuerStudent()
makePersistent()
aendern()
aendern()
passwortAendern()
aendern()
deepDelete()
alleBelegungen()
alleBelegungen()
istBeleger()
alleBeleger()
alleBelegerKontaktListe()
alleBeleger()
alleBelegerMitEinsendungen()
loescheBelegung()
loeschen()
belegt()
istTestStudent()
istTestStudent()
istDemoStudent()
alleStudenten()
isKnown()
einlesen()
toCorba()
Student()
...
+student
Korrektur
(from model)
Korrektor()
Korrektor()
eineNeueKorrektur()
zugeordnet()
name()
getKorrektur()
inhaltAendern()
statumZustandPunkteAendern()
statumInhaltZustandPunkteAendern()
korrigierbar()
einlesen()
einlesen()
einlesen()
einlesenMitInhalt()
punkteEinlesen()
punkteEinlesen()
alleKorrekturen()
zustandVonUnkorrigiertAufBeimKorrektorAende...
zustandAendern()
korrektorAendern()
punktzahlZustandAendern()
neueKorrektur()
neueKorrektur()
neueKorrekturen()
zustandUndKorrektornameAendern()
istFreigegeben()
...
+korrektur
Context
(from webmac...
-mContext
Kontext
(from WebAssign)
Kontext()
Kontext()
out()
expand()
benutzer()
veranstalter()
kurs()
student()
aufgabenheft()
aufgabe()
teilaufgabe()
einsendungen()
vorkorrektur()
autokorrektur()
korrektur()
korrektor()
korrektorkorrektur()
...
+theContext
Veranstalter
(from model)
Veranstalter()
Veranstalter()
isFernuniVeranstalter()
initVeranstalter()
neuerVeranstalter()
deepDelete()
exist()
einlesen()
alleVeranstalter()
alleVeranstalterEinlesen()
getkursRessourcenPath()
createDirectories()
removeDirectories()
name()
Veranstalter()
instanzAlsTupel()
...
+veranstalter
WebAssignRequestInfo
(from WebAssign)
WebAssignRequestInfo()
forward()
include()
authorizationLevelSet()
setResponseContentType()
setResponseContentLength()
addResponseHeader()
setResponseStatus()
getRequestHeader()
getContentType()
getContentLength()
getServletInputStream()
getServletOutputStream()
createFormHashtable()
createHashtableFromQueryString()
getByteArray()
readClientByteArray()
decodeAuthorization()
fill401Header()
unauthorized()
whyUnauthorized()
sayPersonalizedUnauthorized()
setAuthorizationLevel()
...

- **WebAssign.corba**: Stellt Klassenrümpfe für einen Grossteil der Funktionen bereit

Im Package WebAssign.corba sind alle Klassenrümpfe zusammengefasst. Diese dienen den Implementierern als Richtlinien, welche Funktionalitäten die jeweiligen ausgearbeiteten Klassen später bereitstellen sollen. Für die meisten Klassen die Klassen Klasse *Holder, *Helper, sowie gelegentlich *POA, *POATie und ein Interface vorgesehen. Das folgende Beispiel zeigt die grundlegenden Funktionen für die Bearbeitung von Aufgaben:

AufgabenHandlerHelper:
Stellt die Rümpfe für folgende grundlegende Funktionalitäten bereit:
insert: Einfügen eines Streams
extract: Extrahieren eines Streams
type:
read: Auslesen eines Streams
write: Transformiert ein AufgabenHandler-Objekt in einen Stream
narrow: Gesicherter Typecast eines beliebigen Objekts in einen AufgabenHandler
get_id: Ausgabe der ID
get_type: Rückgabe des zugehörigen TypeCodes

AufgabenHandlerHolder:
Enthält Verweise auf die zugehörigen Helper-Klassen

AufgabenHandlerPOA:
Transformation eines Objekt in einen AufgabenHandler sowie Test, ob es sich um das Objekt einer bestimmten Klasse handelt.

AufgabenHandlerPOATie:
Extention der Klasse AufgabenHandlerPOA

- **WebAssign.database**:

Das Package WebAssign.database beinhaltet alle Klassen die zum Arbeiten mit der Datenbank benötigt werden. Grundlegend ist hierbei die Klasse AbstractDatabase, die eine abstrakte Oberklasse für die Verbindungen zu verschiedenen Datenbanken bildet. Unterstützt werden die Datenbanken HIS, Informix, Mysql und Oracle, sowie die Datenbank des URZ. Für diese Datenbanken sind in den entsprechenden Klassen Methoden für die Arbeit mit diesen definiert.

- **WebAssign.korrektur**:

Alle nötigen Klassen zur Korrektur der Aufgaben sind in diesem Package zusammengefasst. Unterschieden werden dabei die Korrektur von Aufgaben und Teilaufgaben, die beide von einer Oberklasse Korrekturmodul erben. Gemeinsame Funktionen sind: Auslesen der Aufgaben-/Kursnummer, des Namens des Veranstalters, sowie die Speicherung korrigierter Aufgaben.

- **WebAssign.kserver**:

Das Package kserver enthält alle Klassen, die für die technischen Möglichkeiten der Korrektur nötig sind. So stellt es Klassen für die Verkapselung von Daten bereit:
- Config
- Einsendung(en)
- KorrektorDownload/UploadRequest
- KursInfo

Außerdem gibt es Klassen zur Verwaltung mehrer kursspezifischer Konfigurationen, die Utilities für den Dateizugriff bereitstellen und den Status eines laufenden Datenaustauschs protokollieren.
Zentrale Klassen sind:
- MainServerRequest, einer Oberklasse für alle Requests an den großen MainServer,
- Server, der Hauptklasse des Korrekturservers und
- Slave, dem zentralen Request-Handler für den Korrekturserver.

- **WebAssign.ldap**

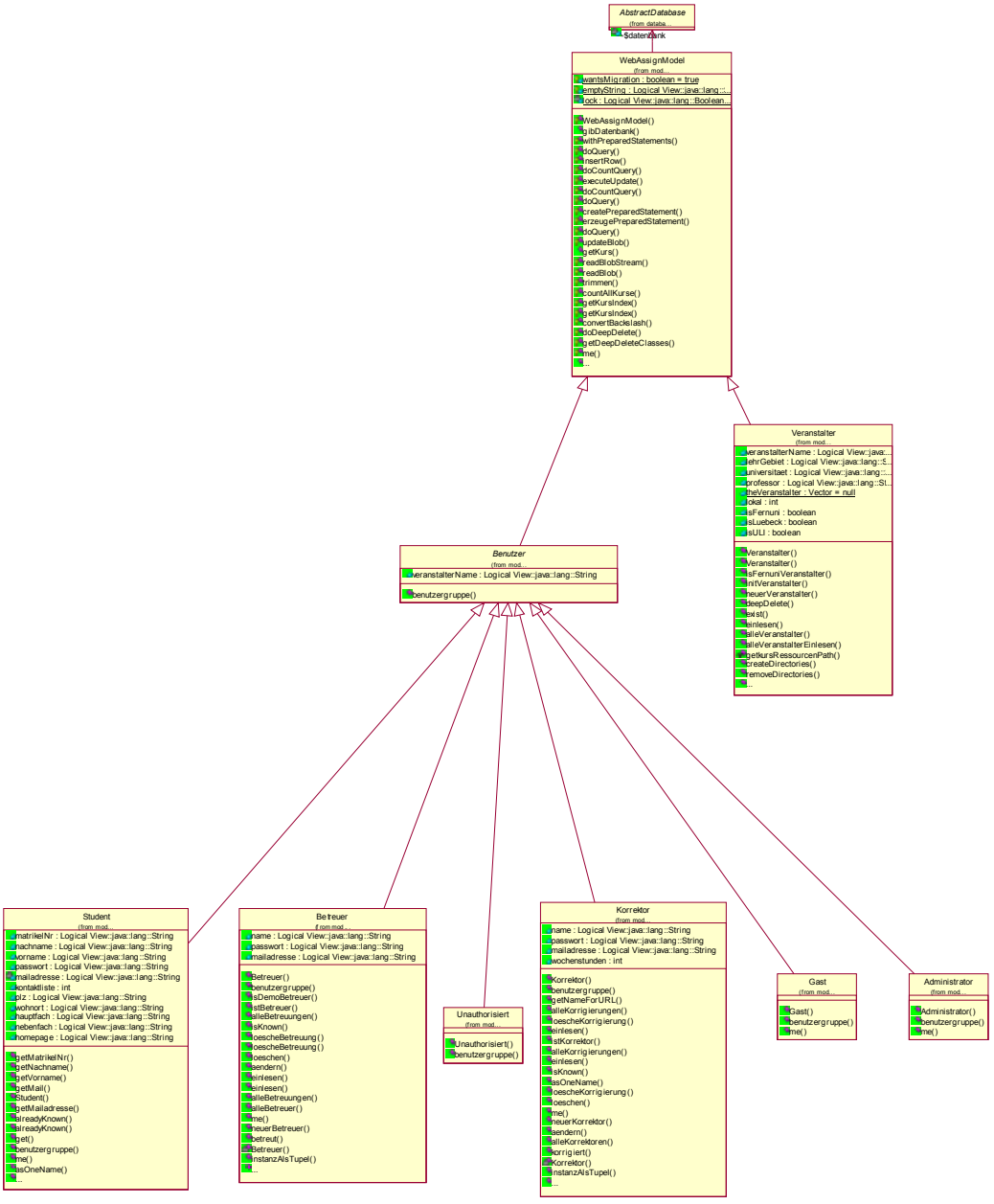
ldap dient der Auswertung von Studenten und Objekten.

- **WebAssign.migrate**:

Da WebAssign die Möglichkeit bietet, verschiedenste Datenbanken zu nutzen werden in diesem Package Klassen zu Überführung von Daten von einer Datenbank in eine andere gegeben.

- **WebAssign.model**:

In diesem Package sind die Entitätsklassen für alle verwendeten Objekte enthalten. Sie definieren die Informationen, die für die jeweiligen Benutzer nötig sind und geben Methoden zu deren Manipulation. Beispielhaft werden die Daten von Benuter und Korrektur aufgeführt:
Für Benutzer können folgende Informationen gespeichert werden: Nachname, Vorname, PLZ, Wohnort, MatrikelNr, Hauptfach, Nebenfach, Passwort, Homepage, Kontaktliste
Für die Korrektur: AufgabenheftNr, AufgabenNr, Datum, Inhalt, Korrektorname, KursNr, MatrikelNr, Punktezahl, Veranstaltername, VersionsNr, Zustand



BenutzerView:
Der *Student*, *Betreuer*, *Korrektor*, *Unauthorisiert*, *Gast* und *Administrator* erben alle von der Klasse *Benutzer*. Dabei erhalten der *Student*, der *Betreuer* und der *Korrektor* weitere Attribute, wie zum Bsp. „*passwort*“ und „*mailadresse*“.
Der *Benutzer* und der *Veranstalter* wiederum erben die Eigenschaften von *WebAssignModel*, welches mit *AbstractDatabase* den Datenbankzugriff realisiert.
Diese Erbfolge ermöglicht ein leichtes Verwalten der Zugriffsrechte auf die Datenbank. *Benutzer* wie *Verwalter* haben dann die gleichen Freiheiten bzw. Einschränkungen. Da der *Verwalter* andere Anliegen hat als ein *Benutzer* wird er abgespaltet und in einer extra Klasse organisiert.

- **WebAssign.services**:

Das Package Services beinhaltet alle Klassen, die für Anwendungen (Services) im Uebungsverwaltungssystem zuständig sind. Unterschieden wird dabei in drei wesentliche Gruppen:

- get
- post
- put

Das Package **services.get** stellt Klassen zur verfügen, die eine html-Seite zurückliefern. (vgl. 3.4 Designentscheidungen bei den Diensten), wie z.B.:
für den Administrator:

- Veranstalter erzeugen und loeschen
- Kurs erzeugen, loeschen, kopieren, veraendern
- Betreuer erzeugen
- Konfiguration anzeigen

für den Korrektor

- Online-Korrekturen-Uebrsicht anzeigen
- Einem Korrektor zugeordnete Einsendungen packen und zum Download bereitstellen.

für den Benutzer:

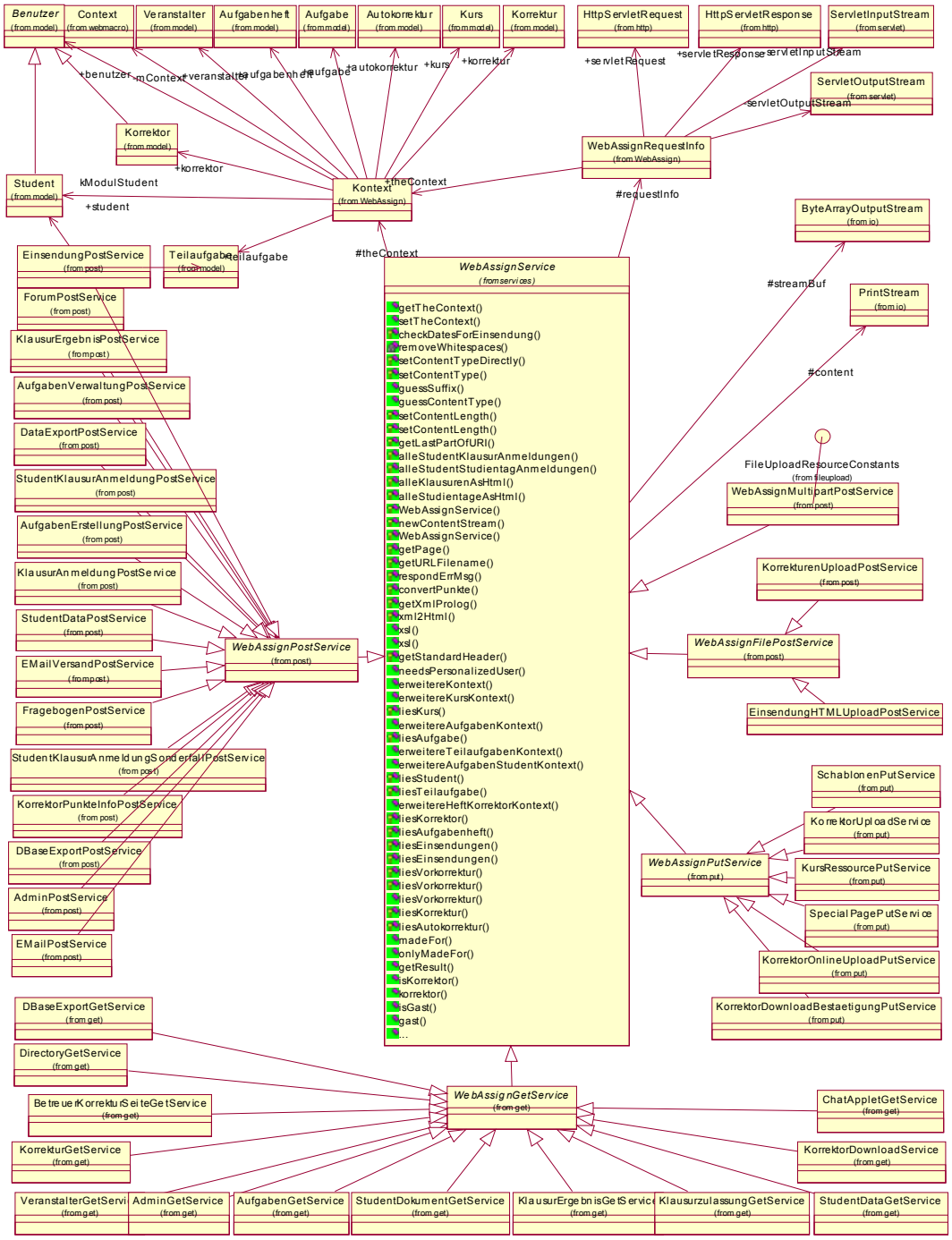
- Kursparameter anzeigen
- Auswahlmaske zur Ansicht spezieller studentischer Einsendungen anzeigen
- Aufgabentext, Musterloesung, Korekturhinweise, Korrekturschablone oder Quittung ausgeben.

Das Package **services.post** sind dafür verantwortlich, an den Server gesandte Formulardaten anzunehmen, weiterzuverarbeiten und schließlich eine Bestätigungsseite zurückzuliefern:

- Hochladen einer Schablone
- E-mail an alle Kursbeleger versenden.
- Anmeldung zu Klausur und Studientag durchfuehren.
- Gezippte Korrekturen ins System hochladen.
- "Message of the day" aendern.
- Anmeldung / Kursbelegung in WebAssign loeschen.

Services.put-Klassen dienen der Weiterverabeitung von ganzen Dateien, die ins WebAssign-System hochgeladen werden:

- Online-Korrekturen in die DB eintragen.
- Eine neue "Rohseite" ins System hochladen.
- Eine Portalseite ins system hochladen



ServiceView:
WebAssignService ist Basisklasse aller Serviceklassen in WebAssign, sie enthält alle grundlegenden Methoden und Funktionalitäten. Die Services werden in drei Gruppen spezialisiert: *WebAssignPostService*, *WebAssignPutService* und *WebAssignGetService*, sie alle erben von *WebAssignService* und erweitern die Funktionalität entsprechend. Die Servicearten werden über die *Kontext*Klasse gesteuert/zugeschnitten. An alle Unterklassen wird zum Beispiel eine Methode vererbt, die den Zugriff auf *Kontext* bereitstellt, diese liefert den derzeitigen Zusammenhang zurück. Mit dieser

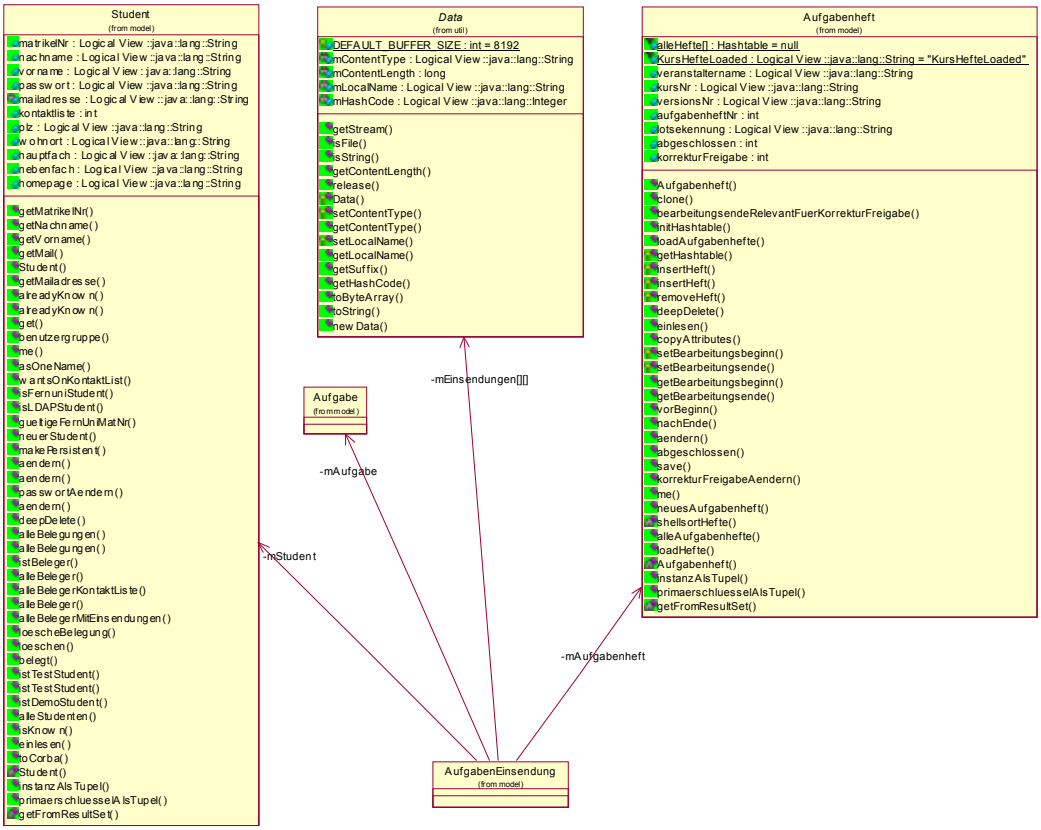
Info wissen die ServiceKlassen welche Methoden anspringen können und welche nicht. (Filtereigenschaft)

- **WebAssign.util**:

In diesem Package werden einige nützliche Zusatzfunktionen (Rundmail versenden, Button einbauen) bereitgestellt, die universell einsetzbar sind.

- **WebAssign.work**:

Die Klasse AufgabenSeite steht für eine Aufgabenseite mit den dazugehörenden Eigenschaften. Die Verwaltung der verschiedenen Aufgabentypen und deren Aufgabennamen erfolgt in der Klasse AufgabenTypen.



WebAssignView:

Die Klasse *AufgabenEinsendung* steuert alle Einsendungen einer Aufgabe und bietet viele Funktionen darauf an. Diese Klasse nutzt die Datenkommnikationsmethoden von der Klasse *Data*. Danach erkennt die Klasse um was für eine Einsendung sich es handelt und macht sich die Methoden dieser Objekte nutzbar. (Bezug zu *Aufgabenheft*, *Aufgabe*) Methoden wären zum Beispiel Korrektur.