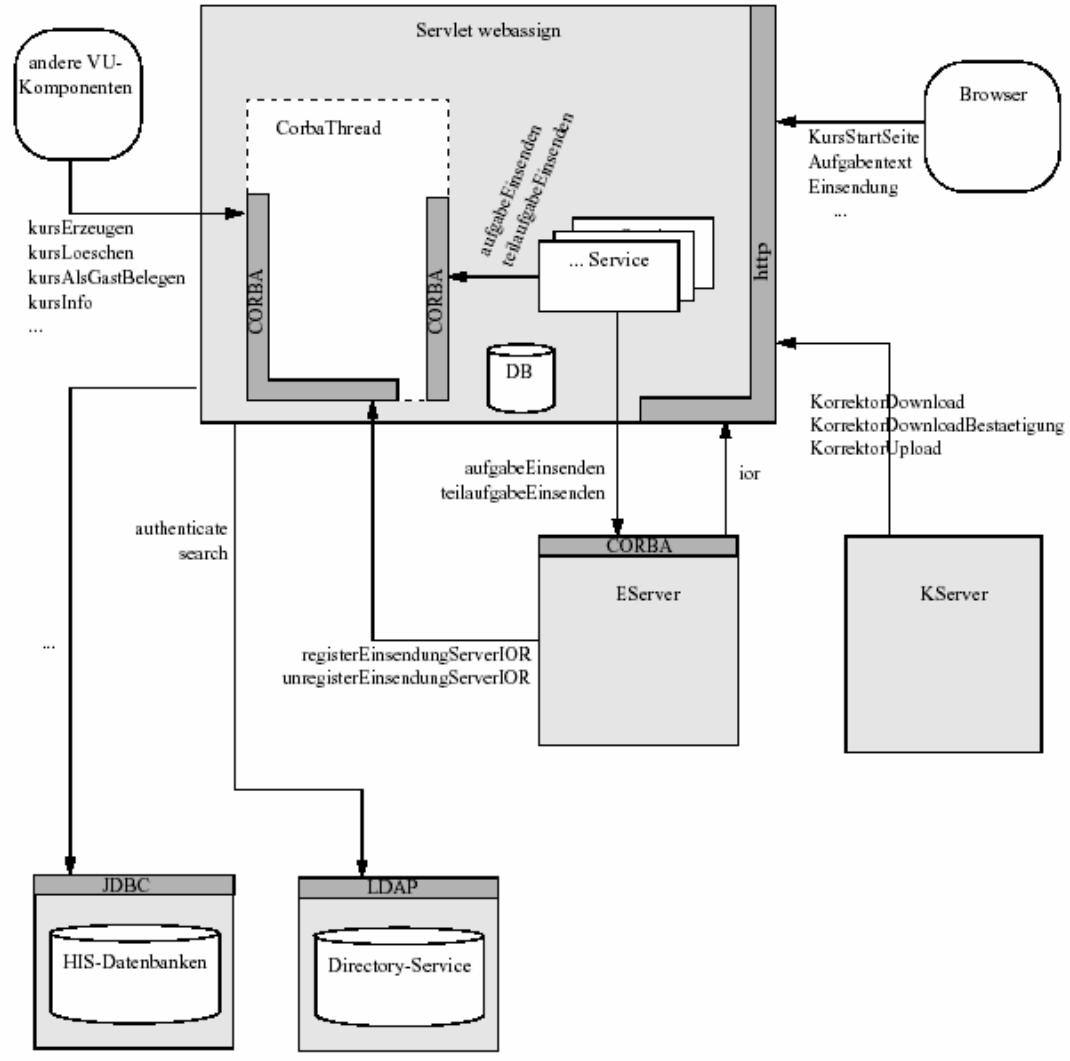
Übungsgruppe: Ueb18 Datum: 25.05.2003
Erarbeitet von: Marcel Meder, Fan Zhang, Rolf Sabsch

1. Allgemeines:

Bei WebAssign handelt es sich um ein Übungssystem, das auf einer Web-Architektur basiert. Es unterstützt sowohl Studierende als auch Lehrende. Insbesondere werden Aufgabenerstellung, Entgegennahme der Lösungen, Korrektur und Distribution über das Internet vorgenommen. Die Hauptbestandteile des WebAssign-Systems sind:

- ein Web-Server inklusive Servlet-Engine, zur Verteilung der HTML-Dokumente und Ansteuerung der Java-Servlets,
- ein relationales Datenbankmanagementsystem, zur Speicherung der angebotenen, dynamischen Inhalte, für das ein Treiber für die JDBC-Verbindungen vorhanden sein muss, und
- ein in Java implementierter Server zur Bereitstellung der Funktionalität.

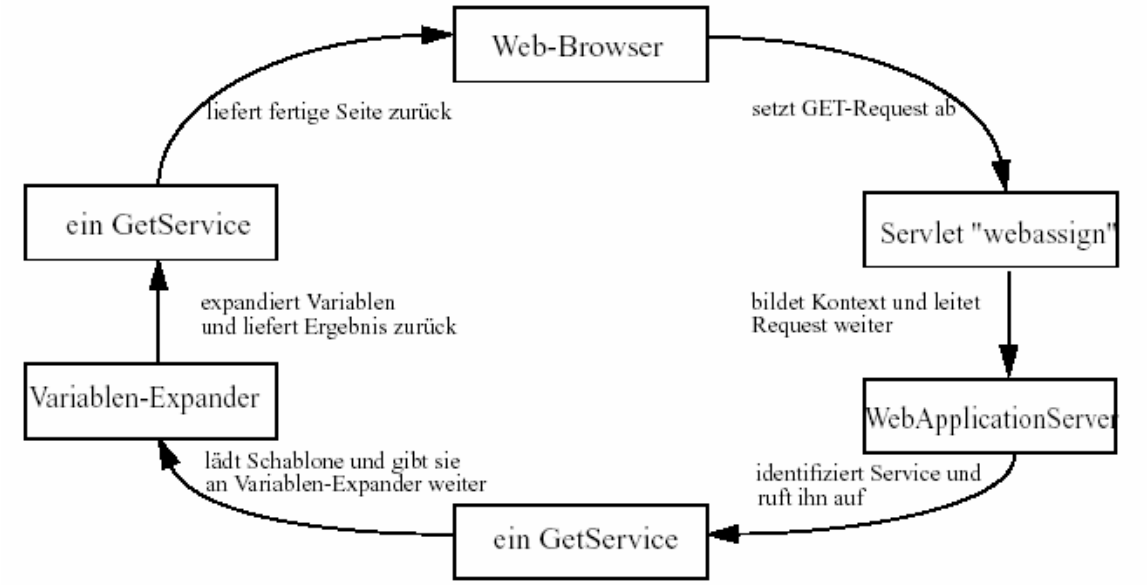
2. Produktübersicht:



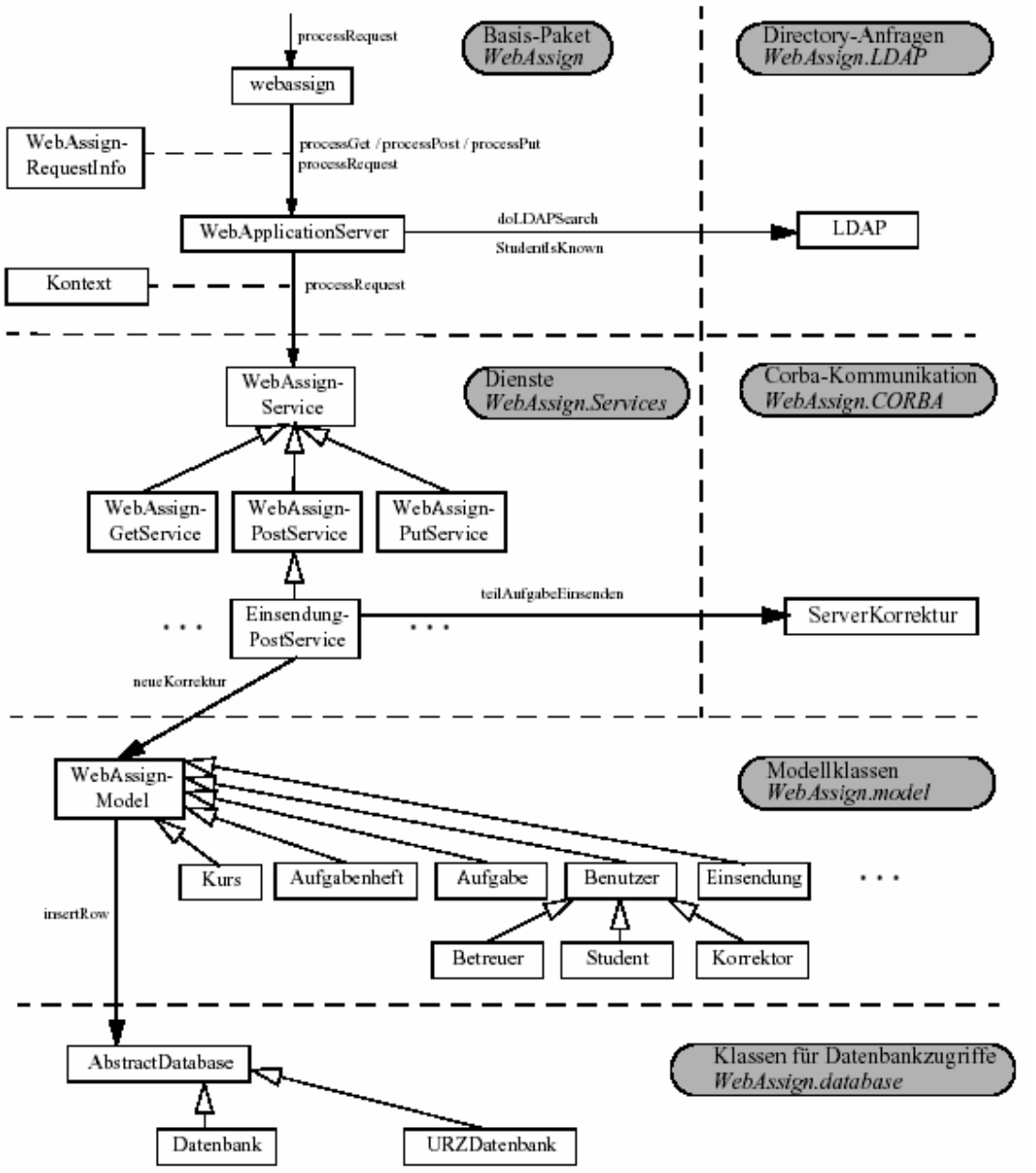
3. Grundsätzliche Design-Entscheidungen

Die Web-Architektur von WebAssign wurde in 4 Schichten realisiert: Web-Client (Browser), Web-Server (mit Servlet-Engine), Anwendungsserver, Daten-Server (relationale Datenbank). Man hat sich für diese Architektur aus zwei Gründen entschieden. Erstens Integration von WebAssign in das universitäre Rechenzentrum (Anbindung an das bereits vorhandene relationale Datenbankmanagementsystem) und zweitens die Notwendigkeit, dass jeder Student von zu Hause aus per Browser die Funktionalität von WebAssign nutzen kann.

Aus der Dokumentation ist ersichtlich, dass die Funktionalität der Software auf zwei Bestandteilen beruht. Einerseits der Servlet Architektur, die sich um jegliche Kommunikation kümmert, die von Außerhalb an das System gerichtet wird.
Wie diese Kommunikation funktioniert beschreibt die folgende Übersicht.



Eine detaillierte Beschreibung der Funktionsweise dieser Architektur stellt das folgende Diagramm dar, dass im Folgenden erläutert wird.



a.) *Servlet-Architektur*

WebAssign basiert auf der Java-Servlet-Architektur. Diese ermöglicht den Aufruf von Java-Programmen über einen Web-Server. Diese Java-Progamme werden Servlets genannt, weil sie auf der Server-Seite ausgeführt werden, im Gegensatz zu Applets, die auf der Client-Seite, also auf dem Rechner des Endbenutzers ausgeführt werden.
Um ein Servlet aufzurufen, fordert der Endbenutzer über seinen Browser eine URL (etwa: www-pi3.fernuni-hagen/servlets/webassign/six/KursStartseite/01612/WS00) an; der WebServer (www-pi3.fernuni-hagen.de) ist so konfiguriert, dass er alle Aufrufe

an ein bestimmtes Unterverzeichnis (/servlets) in einen Aufruf an die sog. Servlet-Engine umwandelt, die als ApplicationServer fungiert. Diese leitet den Aufruf an das angesprochene Servlet (webassign) weiter. Das Servlet interpretiert den Schlussteil der URL (six/KursStartseite/ 01612/WS00) als Aufrufparameter und setzt sie in die Aktivierung eines bestimmten Dienstes (service) um.

b.) *Aufbau und Arbeitsweise des zentralen Servlets*

Das über WebServer bzw. Servlet-Engine aufgerufene Servlet „webassign" leitet den Request – alle Angaben darüber sind im Objekt WebAssignRequestInfo gekapselt – weiter an den WebApplicationServer. Dort wird die URL analysiert, der aufgerufene Dienst extrahiert und die Zugangsdaten des Users geprüft, ggfs. über einen Aufruf an einen Directory-Server (Paket WebAssign.LDAP). Das Kontext-Objekt hält die Attribute des gerade agierenden Benutzers vor.
WebAssignService stellt die Basisfunktionalität für alle Service-Klassen zur Verfügung, etwa den Aufbau des Antwort-Datenstroms, die Expansion der darin enthaltenen Variablen und das Setzen von Content-type und -length. Die Services gliedern sich auf in Get-, Post- und Put-Services, entsprechend der Art des ursprünglichen HTTP-Requests. Unter den Service-Klassen kommt "EinsendungPostService" eine besondere Bedeutung zu: Neben der Speicherung der eingesandten Daten in die DB wird hier die automatische Korrektur einer Aufgabe angestoßen. Hiefür wird eine CORBA-Kommunikation (Paket WebAssign.CORBA) mit dem registrierten Korrekturmodul aufgebaut

Weiterhin wichtiges Merkmal wären die Schnittstellen des Programms. Auch dessen Beschreibung ist sehr allgemein gehalten worden, sodass sie hier auch Erwähnung finden.

c.) Die Schnittstellen

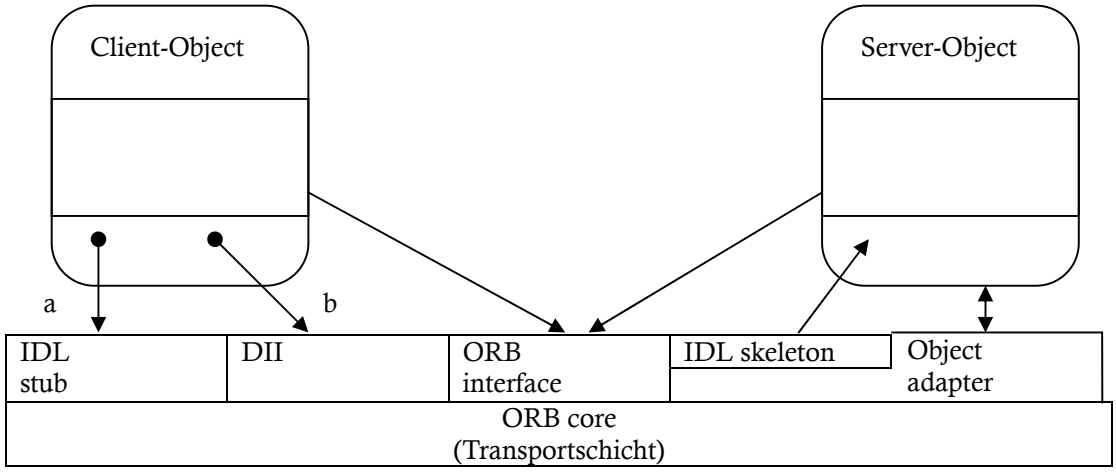
Auf der Applikationsebene kooperiert das webassign-Servlet, auch *WebAssign-Server* genannt, mit anderen Programmmoduln, teils als Dienstgeber, teils als Dienstnehmer. Als *Dienstnehmer* kommuniziert er mit sog. EinsendungsServern. Dies sind Korrekturmodule, in Java oder anderen Sprachen implementiert, die von den Übungsveranstaltungen für ihre speziellen Belange entwickelt und betrieben werden. Ihre physikalische Lokation ist im Allgemeinen nicht auf dem Host, auf dem WebAssign betrieben wird. Bei einer solchen Kommunikation „verschickt" der WebAssign-Server eingesendete Lösungen auf Basis von CORBA an einen EinsendungsServer und erhält die fertige automatische Korrektur zurück. Als *Dienstgeber* fungiert der WebAssign-Server z.B. für das Offline-Korrektorkit. Dieses Java-Progamm, das wegen seiner Realisierung als lokaler WebServer auch Korrektur-*Server* genannt wird, installiert ein Korrektor bei sich zu Hause auf dem Home-PC. Infolge entsprechender
Benutzereingaben fordert es beim WebAssignServer neue Korrekturen für den jeweiligen Korrektor an bzw. lädt korrigierte Einsendungen herauf. Basis dieser Kommunikation ist http. Für die lokale Datenhaltung wird eine relationale Datenbank verwendet, die über JDBC angsprochen wird. JDBC ist auch Grundlage des Datenaustauschs mit den zentralen Datenbanken der Hochschulinfrastruktur (HIS), etwa der Beleger-Datenbank. Für die Abfrage von Zugangsberechtigungen dient das

LDAP-Protokoll, über das ein Directory-
Server angesprochen werden kann. Außerdem bietet WebAssign eine
CORBA-Schnittstelle für andere VU-Komponenten, die externe Komponenten
befähigt, z.B. Kurse innerhalb von WebAssign anzulegen.

Der zweite herausragende Bestandteil von WebAssign ist das Packet von Corba, dass sich um die gesamte Kommunikation zwischen WebAssign (Client) und den automatischen Korrekturmodulen (Server) kümmert.
Der Grund warum dieses Konzept verwendet wurde liegt darin begründet, dass diese Art von Systemen unabhängig gegenüber der Anfrageprogrammiersprache ist. Was bei möglichst großer Portabilität sicherlich ein Vorteil ist.

Das Corba – Objektmodell folgt aber einer gewissen Spezifikation. Diese Spezifikation finden sich folgenden Forderungen wieder.

1. IDL-stub:
   Stellt Routinen, ähnlich wie Bibliotheksroutinen, zur Verfügung, die der clent benutzt, um Dienstleistungen des Servers anzufordern.
2. Inferface Repository:
   Speichert Informationen über die Schnittstelle. Diese Informationen können zur Laufzeit von Clients abgefragt werden, um dynamische Anforderungen aufzubauen. Dazu benutzen sie die DII (dynamic invocation interface).
3. IDL-skeleton:
   Rahmen, der vom Programmierer mit Code gefüllt werden muss. Dieser Code wird ausgeführt, wenn eine Anforderung eintrifft.
4. Implementation Repository:
   Verwaltet Informationen, die der ORB benötigt, um Server zu lokalisieren und zu starten



Innerhalb dieser Spezifikation gibt es 4 Möglichkeiten (nach Balzert), wie diese Kommunikation realisiert werden kann.

a.) ein client- und Serverbasierter ORB
Bei client- und Serverbasierten ORBS wird der ORB durch Routinen implementiert, die sich auf der Quell- und Zielplattform befinden. Jeder Client- und jeder Serverprozess interagiert mit diesen Routinen, die zusammengenommen die ORB-Funktionalität zur Verfügung stellen.

b.) ein Serverbasierter ORB
Bei einem Serverbasierten ORB geschieht die Interaktion mit ORB-Servern, die die ORB – Funktionalitäten zur Verfügung stellen.

c.) ein Systembasierter ORB
In Systembasierten ORBs sind die ORB – Dienstleistungen Teil des zugrunde liegenden Betriebssystems.

d.)
Die vierte Möglichkeit ergibt sich aus der Kommunikation mit Fremdsystemen.

i.) Basic Object Adapter (BOA)
Objekte innerhalb des fremden Systems werden als Server des BOA registriert.

ii.) Spezialisierter Objekt – Adapter
Wird eingesetzt, wenn das fremde System seine Objekte selbst verwalten will. Beispielsweise werden ODMG – kompatible Objektorientierte Datenbanken über den spezialisierten Object Database Apapter (ODA) in ORB - basierte Umgebungen eingebettet.

iii.) gateway
Ein fremdes System kann über ein gateway integriert werden. Das fremde System verhält sich wie ein anderer ORB in der Umgebung.

Das wären also alle Möglichkeiten, wie dieses Corba hätte implementiert werden können. Es wurde hier die Version von b genommen. Diese Version ist hier auf jeden Fall zu bevorzugen. Version a.) wäre unbefriedigend, da hierfür eine Clientversion auf jeden Rechner installiert werden müsste. Diese wäre für allen für eventuelle Gäste eine sehr schlechte Lösung.
Die Variante c.) fällt deswegen aus, weil ein neues Betriebssystem dafür nötig gewesen wäre. Dieser Programmieraufwand wäre aber zu groß für dieses Objekt. Man könnte aber auch kein anderes System verwenden, da bis dato ein solches System noch nicht vorlag.
An diesem System sind auch keine Fremdsystem beteiligt, also fallen alle Lösungen aus d.) weg. Somit ist, wie bereits erwähnt, b.) die beste Lösung, weil dort vor allen kein Extraaufwand an die Clienten abfällt.

Nun bleibt zu klären, wie die einzelnen Komponenten von oben innerhalb der Corba -Version von Webassign integriert sind.

- IDL-stub:
  Diese Funktionalität ist innerhalb von vier Klassen realisiert.
  _AufgabenHandlerStub
  _TeilaufgabenHandlerStub
  _WebAssignCorbaServerStub

_EinsendungServerStub
Diese Vergehensweise ist deswegen nötig, weil der Benutzer des Systems, einmal das Einsenden aller Aufgaben, bzw. einzelner Aufgaben, sowie durch andere Anfragen die Möglichkeit hat, mit dem System in Verbindung zu treten. Diese Anfragen werden dann direkt an den verantwortlichen Händlern weitergeleitet.
AufgabenHandler
TeilaufgabenHandler
Hier bleibt aber noch zu erwähnen, dass _WebAssignCorbaServerStub und _EinsendungServerStub direkt mit einer Klasse aus lang in Verbindung stehen. Das wird die Grundlage für die dynamische Kommunikation bilden, wo die Anfragen. Soweit erkenntlich müsste es sich hierbei um eine Art Interface Repository handeln.

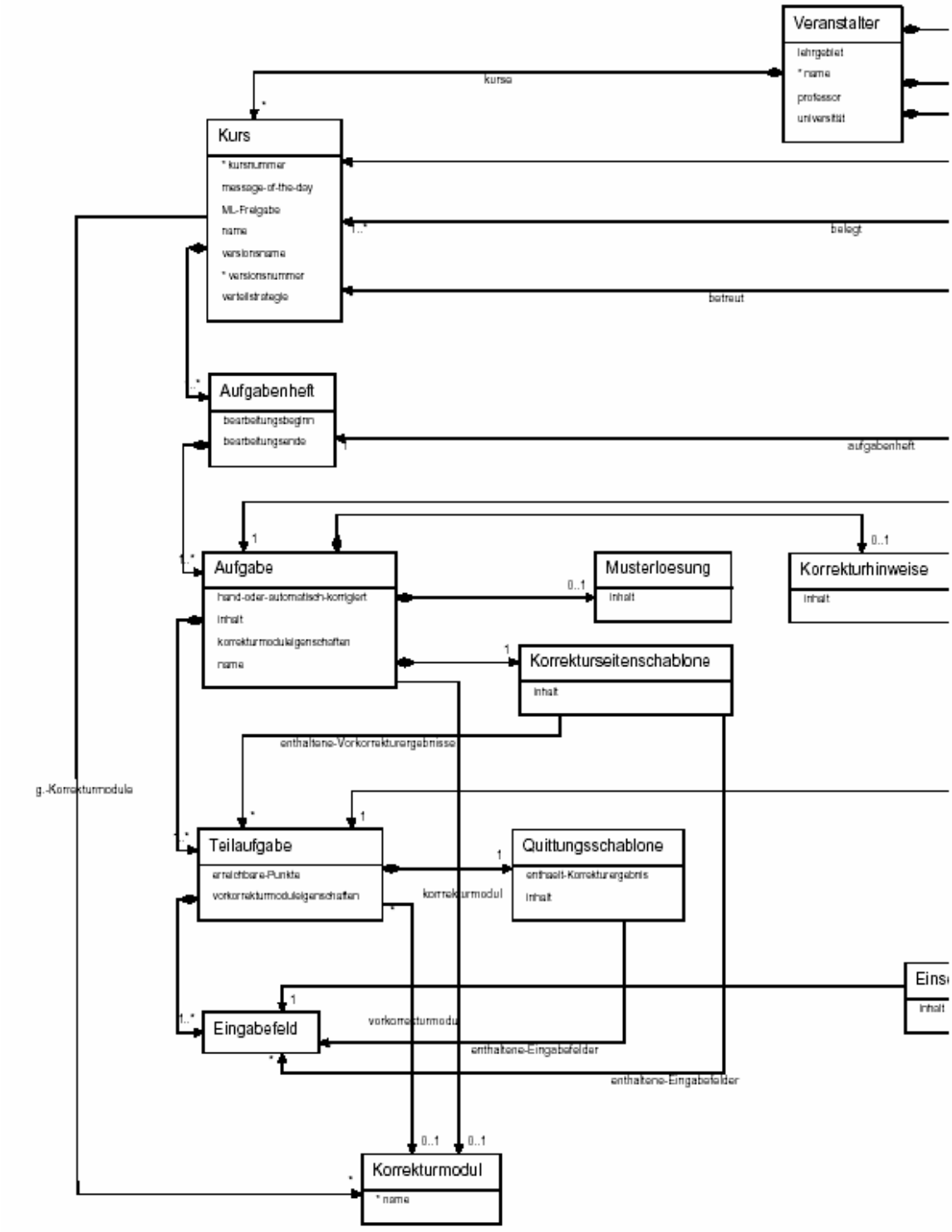
- IDL-skeleton:
  Hier stehen die Klassen, an welche die Operation– Versionen weitergeleitet worden.
  EinsendungServerPOA ➔ EinsendungServerOperations
  WebAssignCorbaServerPOA ➔ WebAssignCorbaServerOperations
  AufgabenHandlerPOA ➔ AufgabenHandlerOperations
  TeilaufgabenHandlerPOA ➔ TeilaufgabenHandlerOperations
  Wobei die Relation zwischen den Klassen eine einfache Assoziation ist. Prinzipiell werden also die Anfragen an die Operationen weitergeleitet.

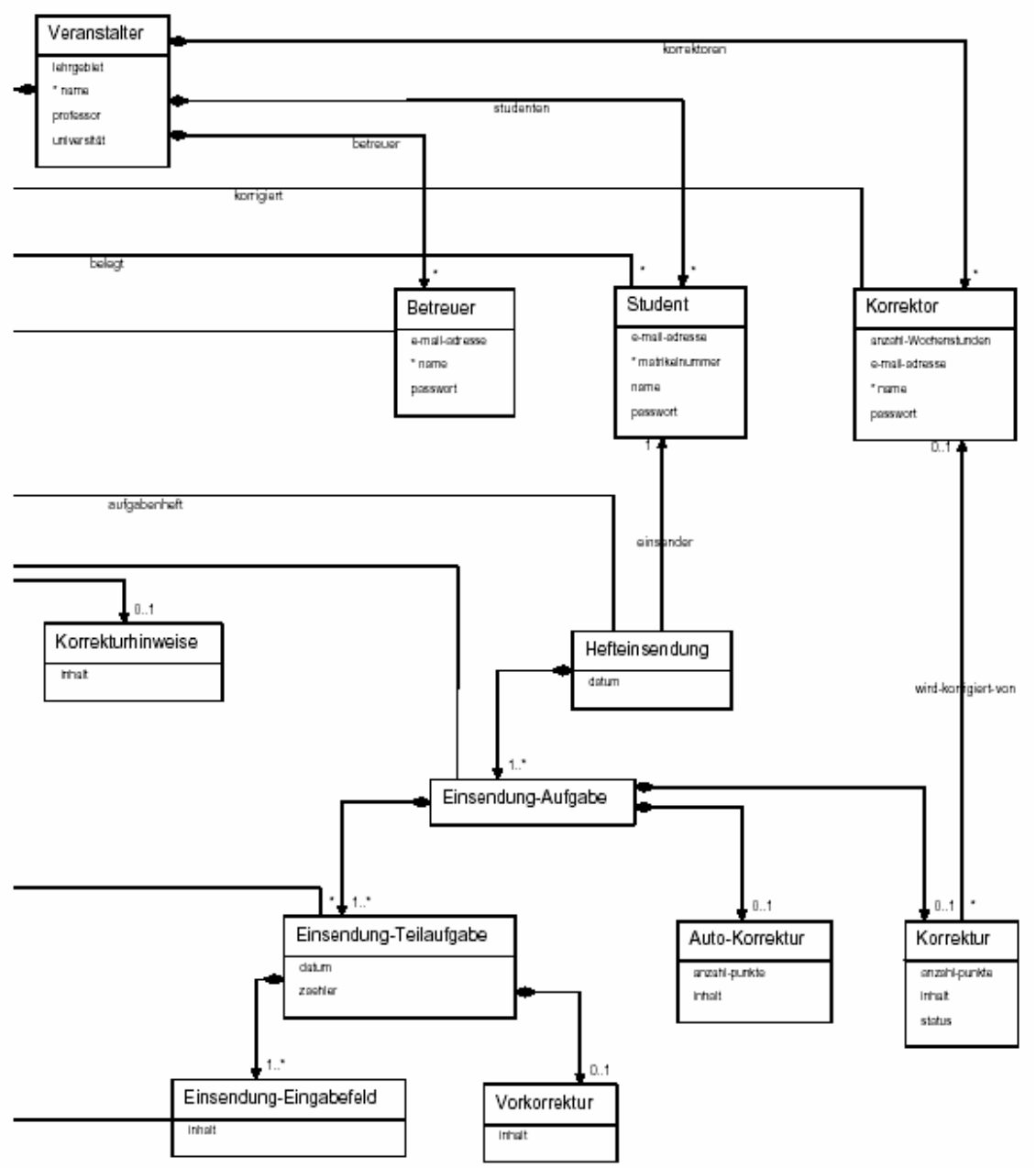
- Implementation Repository:
  Dieses Element ist nicht zu finden. Das könnte daran liegen, dass das verwendete Corba auch nur eine Instanz von der eigentlichen Corba – Spezifikation ist. Also fehlt sozusagen die Server – Komponente, da es sich hier um ausschließlich um Client – Dienste innerhalb des Packets Corba handelt. Also ist die eigentliche Weiterleitung an einen Server dort auch nicht zu finden.

Als letzte Komponente bleibt die EServerCaller – Klasse zu nennen. Sie besitzt die Funktionalität, die als hervorstehendes Merkmal von Corba erwähnt wurde. Also sie führt die Konvertierung von den Programmiersprachen der Anfrage in die Programmiersprache des Corba – Clients um. Oder mit dem worden von den Entwicklern: „Verbirgt Corba-Internals vor der aufrufenden Umgebung. Konvertiert BusinessObjects in die zugehörigen Corba- Äquivalente, ruft den EServer auf und rekonvertiert die Ergebnisse.“ Diese Klasse ist dann mit den TeilaufgabenHandler und AufgabenHandler verbunden. Damit die Anfragen innerhalb der internen Sprache von Corba ausgewertet werden können.

4. Klassen und Packetstruktur.

Hier ist die Domänenklassenstruktur, wie sie von den Machern von WebAssign mitgegeben wurde.


Veranstalter
lehrgebiet
* name
professor
universität
kurse
Kurs
* kursnummer
message-of-the-day
ML-Freigabe
name
versionsname
* versionsnummer
verteilstrategie
belegt
betreut
Aufgabenheft
bearbeitungsbeginn
bearbeitungsende
aufgabenheft
Aufgabe
hand-oder-automatisch-korrigiert
inhalt
korrekturmoduleigenschaften
name
Musterloesung
inhalt
Korrekturhinweise
inhalt
Korrekturseitenschablone
inhalt
enthaltene-Vorkorrekturergebnisse
g.-Korrekturmodule
Teilaufgabe
erreichbare-Punkte
vorkorrekturmoduleigenschaften
korrekturmodul
Quittungsschablone
enthaelt-Korrekturergebnis
inhalt
Eingabefeld
vorkorrekturmodu
enthaltene-Eingabefelder
enthaltene-Eingabefelder
Korrekturmodul
* name



Das Produkt ist in 9 Pakete aufgeteilt, welche folgende Bedeutung haben.

Corba:

Dieses Packet enthält alle Klassen zur Kommunikation mir den Auto-Korrekturmodulen, die zur Bearbeitung der Lösungen eingesetzt werden.

Database:

Dieses Packet enthält alle Schnittstellen für die Datenbank.

Korrektur:

Hier ist die komplette Funktionalität der „manuellen“ Korrektur enthalten.

LDAP:

Dient zur Authentifizierung der Studenten. In diesem Packet ist die komplette Abfrage der Passwörter und ähnliches realisiert.

Migrate:

Migrater unterstützt die Migration von WebAssign-Daten von einer Datenbank in eine andere Datenbank. Die Quelle und das Ziel der Migration müssen in der Konfigurations- Datei migrate.properties im Verzeichnis webassign/conf hinterlegt sein.

Model:

Dieses Packet enthält alle Typen von Benutzern und ihre gesonderten Status.

Services:

Beinhaltet alle Dienste (Put, Get und Post) die für die Kommunikation via HTML nötig sind.

Util:

Enthält Werkzeuge wie Zip, Mailer, Konverter und Dateizugriffe, die für die Arbeit mit dem Programm wichtig sind. Diese wurden aber nicht direkt vor dort eingebunden, da sie die Lesbarkeit verschlechtern würden. Deswegen sind all diese Funktionen in ein separates Packet gesteckt worden.

Work:

Das Packet repräsentiert Aufgaben, bei der ein oder mehrere Begriffe überprüfen werden. Also beinhalt dieses Packet die Funktionalität, die für die Bereitstellung der Aufgaben benötigt werden.

Wichtig für die Funktionsübersicht ist in diesem Zusammenhang die konkrete Speicherung der Daten innerhalb einer Datenbank. Dafür wurde eigens folgende Abbildung bereitgestellt. Dabei wird gleichzeitig eine Übersicht über die anderen Schnittstellen gegeben.