Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Ein Framework für Online-Tests in einer verteilten serviceorientierten Architektur

Diplomarbeit

Leipzig, März 2008

vorgelegt von

Dienst, Steffen
steffen.dienst@elateportal.de
Studiengang Informatik

Fakultät für Mathematik und Informatik
Abteilung Betriebliche Informationssysteme
Prof. Dr. H.G. Gräbe

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Integration von neuen Aufgabentypen in das Aufgaben- und Prüfungsframework des elatePortals. Dazu wird analysiert, wie eine Architektur zur Umsetzung von Tests nach aktuellem Forschungsstand aufgebaut sein sollte. Die daraus resultierenden Erkenntnisse werden durch entsprechende strukturelle Erweiterungen des bestehenden Frameworks umgesetzt. Verschiedene konkrete Anwendungsfälle von Aufgabentypen werden betrachtet und prototypisch implementiert. Dazu gehören beispielsweise die automatische Korrektur von Programmieraufgaben und Aufgaben aus der theoretischen Informatik mit Hilfe von Autotool. Die so entstandene Software erlaubt deutlich flexiblere Einsatzmöglichkeiten zur Unterstützung der Lehre.

Inhaltsverzeichnis

1 Einleitung

Prüfungen sind ein wesentlicher Bestandteil einer jeden Form von universitärer Ausbildung. Die Art und Weise ihrer Durchführung ist seit langer Zeit unverändert: Sie werden fast ausschließlich entweder schriftlich in Papierform oder mündlich abgelegt. Eine Vielzahl von Veränderungen lassen diese beiden Formen der Prüfung in einigen Fällen in zunehmendem Maße unadäquat erscheinen. Zum einen sind allein die Menge an Studierenden in einigen Studiengängen eine organisatorische Herausforderung, beispielsweise an die Raumplanung und die Terminvergabe, aber vor allem an die Korrektoren. Zum anderen lassen sich manche Wissenszusammenhänge mit den herkömmlichen Prüfungsformen nur ungenügend testen, etwa bei Informatikstudenten die Fähigkeit, konkrete Lösungen für algorithmische Probleme zu implementieren. Aus diesem Grund beschäftigt sich ein Teilbereich des E-Learnings mit der Frage, wie Prüfungen, oder allgemeiner überhaupt Tests, mit Hilfe von Software durchgeführt werden können.

Der Umfang vorhandener Literatur zum Thema E-Testing ist im Vergleich zur Bandbreite von Veröffentlichungen zu anderen Bereichen des E-Learnings vergleichsweise gering. Die wachsende Bedeutung dieses Teilgebietes zeigt sich jedoch an einigen wichtigen Projekten, wie z.B. dem E-Learning-Framework der britischen Initiative „Joint Information Services Committee“ (JISC)[1] zusammen mit dem australischen „Department of Education, Science and Training“ (DEST)[2]. Diese bemühen sich um allgemeingültige Standards und Beschreibungen von abstrakten Architekturen und allgemein notwendigen Softwareartefakten zur Unterstützung von E-Assessment.

Diese Initiativen gehen jedoch davon aus, dass die zu verwendenden konkreten Softwareartefakte erst noch zu entwickeln sind. Der Anwendungsfall der Integration von existierenden Testwerkzeugen in ein übergeordnetes Framework und die damit verbundenen technischen Anforderungen sind jedoch nicht deren Betrachtungsgegenstand.

1.1 Zielsetzung

Die vorliegende Arbeit beschäftigt sich mit der Erstellung einer Architektur zur Integration einer Vielzahl von Testwerkzeugen in ein umfassendes Framework zur Durchführung von vorlesungsbegleitenden Übungen mit automatischer Korrektur sowie für generelle universitäre Kompetenzmessbedürfnisse wie z.B. Prüfungen. Die Grundlage dieser Architektur ist das Aufgabenframework des elatePortals,

[1] http://www.jisc.ac.uk/
[2] http://www.dest.gov.au/

welches im Kapitel 2 vorgestellt werden soll.

Neben den einzelnen Aufgabenstellungen, die Ergebnis des Kapitels 2 sind, sollen folgende Fragen beantwortet werden:

1. Wie ist der aktuelle Stand der Verwendung von Testwerkzeugen an der Universität Leipzig?
2. Wie ist das Aufgabenframework des elatePortals aufgebaut?
3. Wie können die vorhandenen Testartefakte im elatePortal wiederverwendet werden? Welche Erweiterung sind dafür notwendig?

Zur Beantwortung der dritten Frage ist eine Auswahl von Testwerkzeugen notwendig, deren Integrierbarkeit demonstriert werden soll. Ihre Auswahl ist ebenfalls Bestandteil des Fazits von Kapitel 2.

1.2 Gliederung der Arbeit

Diese Arbeit gliedert sich im Wesentlichen in drei Abschnitte. Kapitel 2 beinhaltet eine Bestandsaufnahme von Werkzeugen und Systemen, die für die Anwendungsdomäne E-Testing bzw. E-Assessment verwendet werden. Diese Auflistung ist wiederum eingeteilt in die an der Universität Leipzig verwendeten Systeme und in sonstige im deutschsprachigen Raum verbreitete Anwendungen. Dieses Kapitel erfasst außerdem fachspezifische Anforderungen, die von Nutzern in Leipzig an ein E-Testing-System gestellt werden. Aus diesen Erkenntnissen ergeben sich dann die genauen Aufgabenstellungen, die in dieser Arbeit umgesetzt werden sollen (Kapitel 2.5).

Das anschließende Kapitel 3 stellt drei Projekte vor, deren Zielstellungen die Erfassung und Analyse der strukturellen Grundarchitektur von E-Testing-Software auf unterschiedlichen Granularitätsebenen sind. Diese Analysen führen zu den Entscheidungen, wie die zu erstellende Softwarearchitektur nach dem aktuellen Erkenntnisstand beschaffen sein sollte. Danach folgt die entsprechende Umsetzung in Form einer Architekturbeschreibung des erstellten Testystems sowie der einzelnen Aufgabentypen

Kapitel 4 beinhaltet die Details der erstellten Funktionalitäten, ihr konkreter Aufbau, Struktur und die Details der umgesetzten Implementierung sowie Beispiele für den Einsatz der neuen Aufgaben.

Kapitel 5 fasst die Ergebnisse dieser Arbeit zusammen und listet Themen auf, die für die Weiterentwicklung der entstandenen Software relevant sind.

2 Anforderungsanalyse

Um die genaue Aufgabenstellung dieser Arbeit definieren zu können, soll in diesem Kapitel ein Überblick über einige der aktuell verwendeten E-Testingsysteme in der deutschen Hochschullandschaft gegeben werden. Speziell soll auf die an der Universität Leipzig verwendeten Lösungen eingegangen werden und deren Funktionsumfänge und angebotene Workflows mit den von Nutzern geäußerten Anforderungen verglichen werden.

2.1 Aufgabenframework des elatePortal-Projekts

Das elatePortal[3] wird seit Ende 2005 an der Universität Leipzig entwickelt [Ber07a]. Das Ziel des Projekts ist mehrschichtig. Einerseits dient es als Versuchsplattform für Software-Produktlinien-Entwicklung (nach der Methodik in [PBL05]) und die Erprobung aktueller Technologien für die Webentwicklung am Lehrstuhl „Betriebliche Informationssysteme“ [Ber07b], andererseits entwickelt es konkrete Software zur Verwaltung von Vorlesungen und zur Durchführung von Onlineklausuren. Das diesen Klausuren zugrunde liegende Aufgabenframework soll hier nun kurz vorgestellt werden.

Im Folgenden ist das Aufgabenframework zu unterscheiden vom eigentlichen elatePortal: Das Framework (im Weiteren „taskmodel“ genannt) ist ein eigenständiges Artefakt, welches im Portal für Übungsaufgaben sowie in einem eigenständigen Prüfungsserver für Prüfungen verwendet wird. Das taskmodel definiert einen kompletten Workflow für eine Prüfung, von Definition der Aufgaben über die Durchführung der eigentlichen Klausur bis hin zur wahlweise automatischen bzw. manuellen Korrektur und Einsichtnahme durch die Benutzer. Damit steht eine Softwarebibliothek zur Verfügung, deren Funktionen deutlich über die Möglichkeit der Durchführung einfacher Übungsaufgaben zur Unterstützung der Lehre hinausgeht. Das System wird erfolgreich seit einiger Zeit für Prüfungen am Institut „Allgemeine Pädagogik“ der Universität Leipzig mit teilweise mehr als 900 Teilnehmern eingesetzt. Darüber hinaus findet eine Evaluation einer mögliche Verwendung für Prüfungen in der selbstständigen Abteilung „Allgemeinmedizin“ statt.

Momentan werden multiple-choice (MC-) Aufgaben, Zuordnungs-, Lückentext- und Freitextaufgaben angeboten. Die Aufgaben werden, je nach Typ, automatisch bzw. zumindest teilautomatisch korrigiert. Die Aufgabendefinitionen lassen, je nach Typ, jeweils eine Vielzahl von richtigen bzw. falschen Antwortalternativen

[3] http://www.elateportal.de

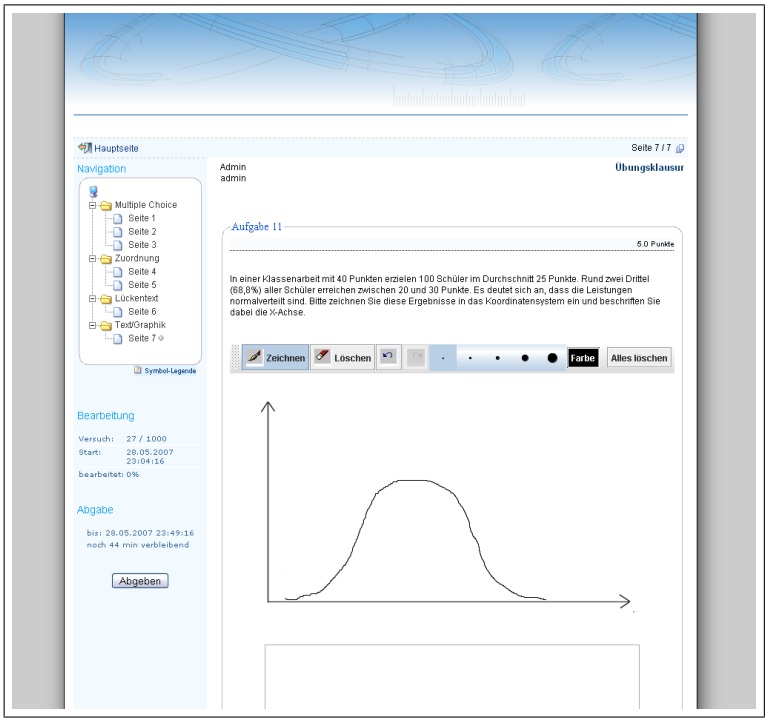


Abbildung 1: Übungsklausur mit dem elatePortal, Quelle: [Ber07a]

zu. Auf diese Weise ist es möglich, für jeden Benutzer eine eigene individuelle Prüfung zur Bearbeitung vorzulegen.

Das Framework bietet in der aktuellen Version einen umfassenden Workflow für Prüfungen bzw. prüfungsähnliche Abläufe: Eine Menge vordefinierter Aufgaben wird dem Nutzer zur Bearbeitung vorgelegt. Dieser bearbeitet die Prüfung an einem speziell abgesicherten PC unter Aufsicht mit Hilfe einer der Papierprüfung nachempfundenen Oberfläche. Die Korrektur der Prüfungen erfolgt teilautomatisch. MC- und Zuordnungsaufgaben können ausschließlich automatisch vorkorrigiert werden, die beiden anderen Aufgabentypen werden einzelnen Korrektoren vorgelegt und bewertet. Auch die anschließende Einsichtnahme und etwaige Einsprüche durch die Studenten werden durch dieses System angeboten. Damit ergibt sich eine umfassende Softwarelösung zur Abbildung einer vollständigen universitären Prüfung von der Erstellung der Prüfungsinhalte, über die Durchführung und Korrektur bis hin zur Einsichtnahme und Analyse der Ergebnisse.

Mit Hilfe des elatePortals konnten Prüfungen mit bis zu 900 Prüflingen erfolgreich durchgeführt werden. Die Umfänge dieser Prüfungen sind mit herkömmlichen Papierprüfungen kaum noch beherrschbar. Die Erfahrungen, die seit nunmehr 5 Semestern in einer Vielzahl von verschiedenen Prüfungsszenarien gewonnen wur-

den, führten zu der Anforderung, das Framework zu erweitern, um auch andere Formen von Kompetenzmessverfahren zulassen zu können. An dieser Stelle soll diese Arbeit ansetzen.

2.2 Weitere verwendete E-Assessmentwerkzeuge an der Universität Leipzig

2.2.1 Autotool

Autotool [WR02] ist ein Werkzeug zur automatischen Bewertung von Aufgaben aus den Bereichen von Mathematik und theoretischer Informatik. Zum Beispiel gibt es Aufgaben zu formalen Sprachen (z.B. Grammatiken, reguläre Sprachen), Automaten (z.B. Turing, Keller), Graphen, Logik und anderen Themenbereichen. Autotool wurde in der rein funktionalen Programmiersprache Haskell implementiert. Es wurde als frei verfügbares Werkzeug von Prof. Waldmann entwickelt und wird seit mehreren Jahren erfolgreich als Lernunterstützung an der Universität Leipzig und der HTWK Leipzig eingesetzt. Übungsaufgaben werden von Tutoren mit Hilfe einer Webapplikation erstellt. Jeder Student bekommt eine durch seine Matrikelnummer individualisierte Aufgabe, so dass ein einfaches Abschreiben von Lösungen nicht möglich ist. Die Lösung einer Aufgabe wird im Allgemeinen nicht nur durch die Korrektheit beschrieben, sondern bekommt auch einen Gütewert zugewiesen. Dieser Wert ermöglicht die Vergleichbarkeit der Güte von Lösungen zwischen den einzelnen Studenten. Mit dieser Funktion werden Highscorelisten erstellt und entsprechend der Platzierung Preise vergeben. Der dadurch entstehende Wettbewerbscharakter wirkt sich nach bisherigen Erfahrungen positiv auf die Motivation und den Arbeitseifer der Studenten aus.

Im Rahmen einer Diplomarbeit [Wan07] erfolgten umfangreiche Umbauten an Autotool. Das Ziel war die Extraktion eines zustandslosen Servers, der durch eine neu definierte Schnittstelle in Verwaltungssysteme für die Hochschullehre beziehungsweise andere existierende LMS integriert werden kann. Bis zu diesem Zeitpunkt bestand Autotool neben den rein semantischen Teilen zum Definieren und Bewerten von Aufgaben auch aus einer Datenbank zur Verwaltung von Aufgaben sowie den Lösungsversuchen von Studenten und einer rein funktionsorientierten Web-Oberfläche. Das Ziel dieser Arbeit war die Integrierbarkeit von Autotool in beliebige Testsysteme, so dass seine Autoren sich auf die Kernkompetenz, also die Bereitstellung von Aufgaben aus dem Bereich der theoretischen Informatik, konzentrieren können.

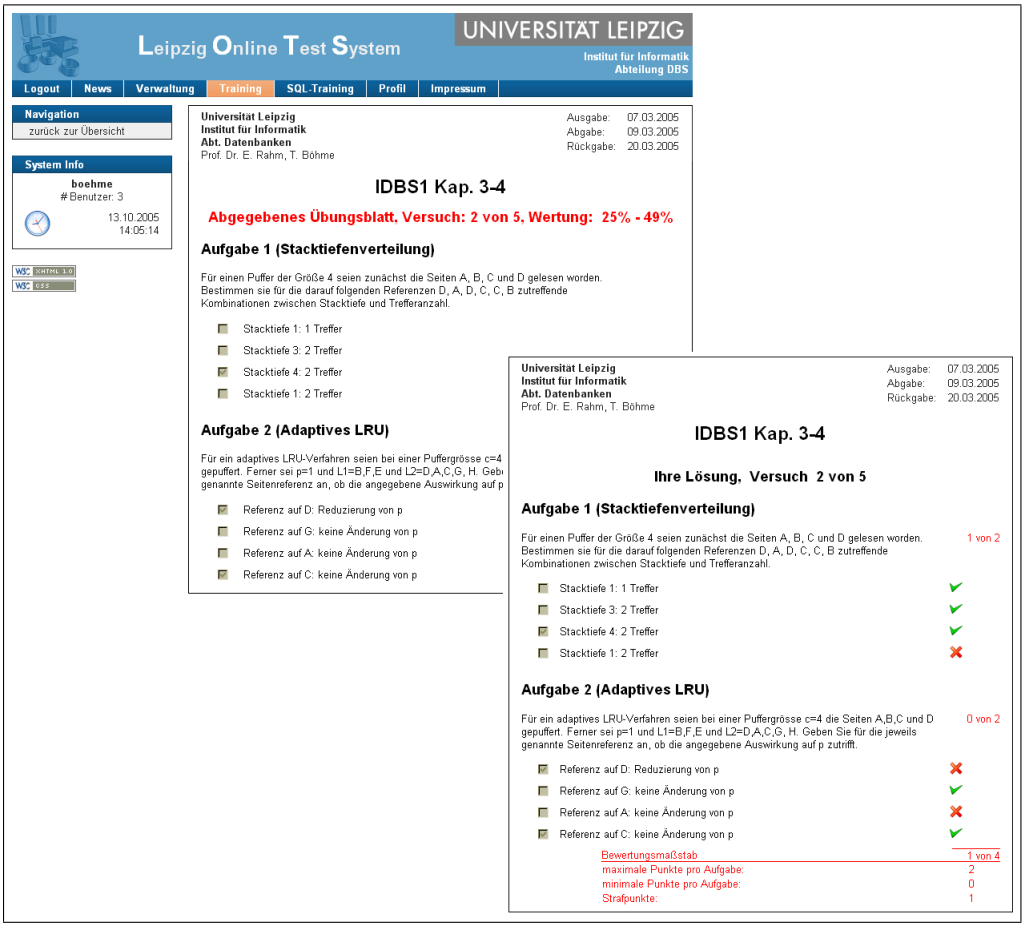


Abbildung 2: Beispielaufgaben in LOTS, Quelle: [BRS06]

2.2.2 SQL-Trainer/LOTS

LOTS (Leipzig Online Test System) [BRS06] ist ein aus dem SQL-Trainer [SR03] entstandenes Projekt zur Durchführung von Übungsaufgaben an der Abteilung „Datenbanksysteme“. Es beinhaltet, neben einer Verwaltung von Vorlesungs-, Übungs- und Studentendaten, eine interaktive Eingabe für SQL-Befehle. Mit Hilfe dieser Interaktionsmöglichkeit sind entsprechend formulierte MC-Aufgaben zu beantworten. Eine automatische Bewertung der Richtigkeit ausgeführter SQL-Statements ist bisher nicht vorgesehen. Der SQL-Trainer ist also nur als Hilfsmittel zur Bearbeitung der gestellten Fragen vorgesehen. Eine automatische Korrektur der Ergebnisse von formulierten Statements ist jedoch prinzipiell möglich, ihre Verwendung ist also auch in Prüfungen vorstellbar. Aufgrund der gemachten negativen Erfahrungen mit Lösungsplagiaten werden die Übungsaufgaben momentan nicht für prüfungsrelevante Leistungen herangezogen, sondern dienen

einzig der Unterstützung des Lernens.

2.2.3 Onyx

„Hochschulsprachtest“ [PDET07] ist ein interdisziplinäres Projekt des Herder-Instituts, des Sprachenzentrums, des Instituts für Informatik und der BPS GmbH zur Feststellung der Sprachfertigkeiten eines Testprobanden. Speziell sollen mit dem im Rahmen des Projekts entwickelten Testsystem Onyx die fremdsprachlichen Kompetenzen im Lesen, Schreiben, Sprechen und Hören nach dem europäischen Referenzrahmen für Sprachen abgeprüft werden können. Onyx setzt auf IMS-QTI v2.1 auf. Es wurden *customInteractions* entwickelt um Audiodateien abspielen und Sprache mit Hilfe eines Mikrophones aufnehmen zu können (siehe Seite 35). Die erstellten Fragen sind also zwar prinzipell mit anderen E-Assessmentsystemen auf QTI-Basis austauschbar, jedoch gibt es dafür aufgrund der in anderen Systemen fehlenden Implementierung dieser beiden speziellen customInteractions dafür keine Relevanz. Onyx ist als eigenständiges Softwaresystem implementiert, kann jedoch mit Hilfe einer Schnittstelle in das LMS Olat integriert werden. Besonders zu erwähnen ist die in Onyx enthaltene Implementierung des QTI v2.1 Standards. Onyx ist als Beispiel für die Erweiterbarkeit des QTI-Standards um neue Interaktionsformen, der Verwendung der aktuellsten Version und aufgrund ihrer freien Verfügbarkeit unter der freien Lizenz ASL 2.0[4] auch interessant als Komponente in anderen E-Testingsystemen, die QTI verwenden wollen. Eine mögliche Zusammenführung dieses Projekts mit dem elatePortal wird zur Zeit evaluiert.

2.3 Aktueller Stand des E-Testings im sonstigen akademischen Bereich

Das E-Testing ist ein wichtiger Bestandteil des E-Learnings. Dieses Kapitel soll eine Auswahl von existierenden Plattformen mit E-Testing-Funktionalität und die darin verwendeten Standards vorstellen. Insbesondere ist hierbei die Integrierbarkeit von neuen Aufgabentypen von besonderem Interesse.

2.3.1 R2Q2

Ein wichtiger Standard zur Beschreibung von Aufgabendefinitionen stellt QTI dar, welches im Kapitel 3.1.3 genauer beschrieben werden soll. Wie am Ende die-

[4] http://www.apache.org/licenses/LICENSE-2.0

ses Kapitels festgestellt wird, liegt die Schwierigkeit bei Erweiterungen des QTI-Standards nicht nur in der Komplexität desselben, sondern insbesondere in der Nichtaustauschbarkeit zwischen verschiedenen, den Standard umsetzenden Plattformen. Idealerweise sollte jede Plattform auf dieselbe Software zum Rendern und Bewerten von Items zurückgreifen. Aus diesem Grund wurde auf der JISC/CETIS Konferenz im Herbst 2005 „Online Educa Berlin 2005“ beschlossen, eine Software zu entwickeln, welche genau das leistet. Das Ergebnis heißt R2Q2 [Wil06a] (Rendering and Response processing services for QTIv2 questions) [5]. Dabei handelt es sich um eine nach den Gesichtspunkten der im Kapitel 3.2 beschriebenen Service Oriented Architecture konstruierte Software, die per Webservice QTI-Inhalte rendert und bewertet. Das Rendern erfolgt in dieser Implementierung in HTML, kann jedoch auch für andere Ausgabeformate erweitert werden. Wie Abbildung 3 zeigt bekommt R2Q2 die Testdaten von einem LMS, welches die von R2Q2 gerenderten Items darstellt und die Ergebnisse weiterverarbeitet.

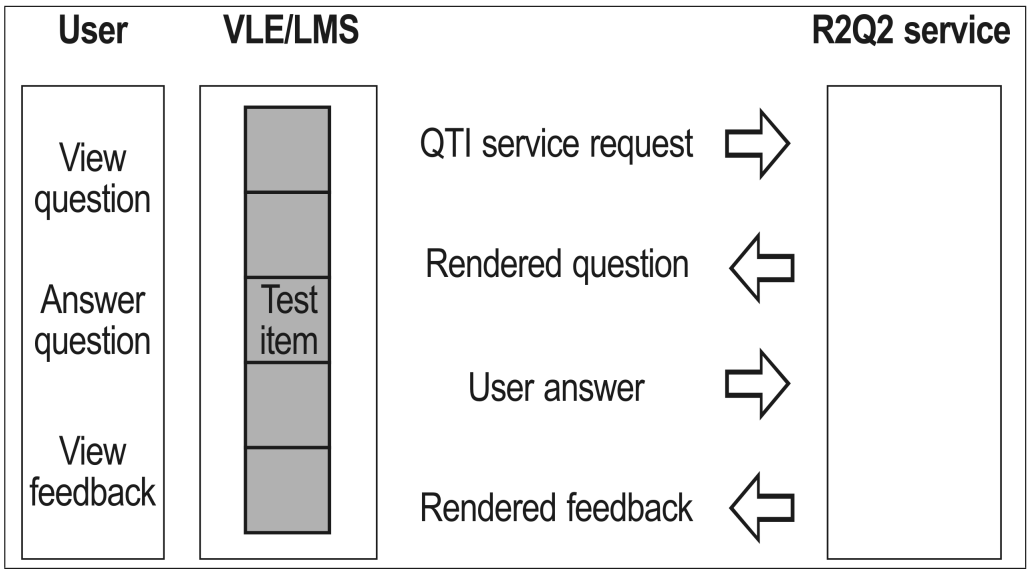


Abbildung 3: R2Q2 Übersicht, Quelle: [wil06b]

Diese Implementierung ist sehr modular aufgebaut, selbst die einzelnen internen Bestandteile sind als Webservice ansprechbar und auch auf diese Weise miteinander verbunden. Damit ergibt sich ein recht komfortables Baukastensystem mit Referenzimplementierungen für die typischen Aufgaben einer Delivery Engine. Die Hauptbestandteile sind:

ContentUnpacker
: Service zum Entpacken von Archiven nach der „QTI Content

[5] http://www.r2q2.ecs.soton.ac.uk/overview/

Packing Specification“

InitialisationEngine
Verwaltung von Benutzersessions

RenderingEngine
Service zum rendern von einzelnen Items in Html

ProcessingEngine
Auswertung der Interaktionsergebnissen, Rückgabe von Feedbackinformationen

Router Interaktion mit den einzelnen Engines, enthält den Zustand von externen Interaktionen mit R2Q2

Sinn dieser Aufteilung ist die Austauschbarkeit dieser Teilimplementierungen. Beispielsweise stellt die RenderingEngine von R2Q2 zwar 16 verschiedene Interaktionsformen dar, die *customInteraction* jedoch nicht. Dasselbe gilt für die Implementierung eines *customOperators*. Eine Testplattform, die nun R2Q2 als Delivery Engine verwendet, zusätzlich jedoch auf diese beiden Konstrukte angewiesen wäre, müsste sowohl die RenderingEngine als auch die ProcessingEngine entweder kopieren und entsprechend erweitern oder zu diesen Services delegieren. Die Verwendung von R2Q2 als Grundlage für die Implementierungen von E-Testingsystemen auf QTI-Basis würde also einen wichtigen Schritt zur Umsetzbarkeit der im Standard vorgesehenen Erweiterbarkeit darstellen. R2Q2 basiert zur Zeit auf der Version 2.0 des QTI-Standards, dieser unterstützt jedoch im Gegensatz zu den Versionen 1.2 bzw. 2.1 keine Zusammenfassung von Items zu Tests. Wegen dieser recht gravierenden Einschränkung ist ein einfacher Austausch der Delivery Engine in Plattformen, die eine dieser Versionen verwenden, nicht ohne weiteres möglich.

2.3.2 LMS mit auf QTI basierenden Testsystemen

2.3.2.1 OLAT

OLAT[6] ist ein Learning-Content-Management-System (LCMS) der Universität Zürich. Es zeichnet sich durch eine umfangreiche Unterstützung des Lernprozesses mit E-Learning-Inhalten, Lerngruppen, Chats und Fortschrittskontrollen als Onlinetests aus. Es vereint also ein Content-Management-System CMS zur Verwaltung und Aktualisierung von Lerninhalten mit den Funktionen eines Learning-Management-Systems zur Unterstützung des eigentlichen Lernvorgangs. Insbesondere ist bei OLAT die Verwendung des Standards IMS QTI für den Austausch

[6] Online Learning And Training, http://www.olat.org

von Testdaten zu betonen. Auf die Erweiterbarkeit von IMS QTI wird im Kapitel 3.1.3 genauer eingegangen.

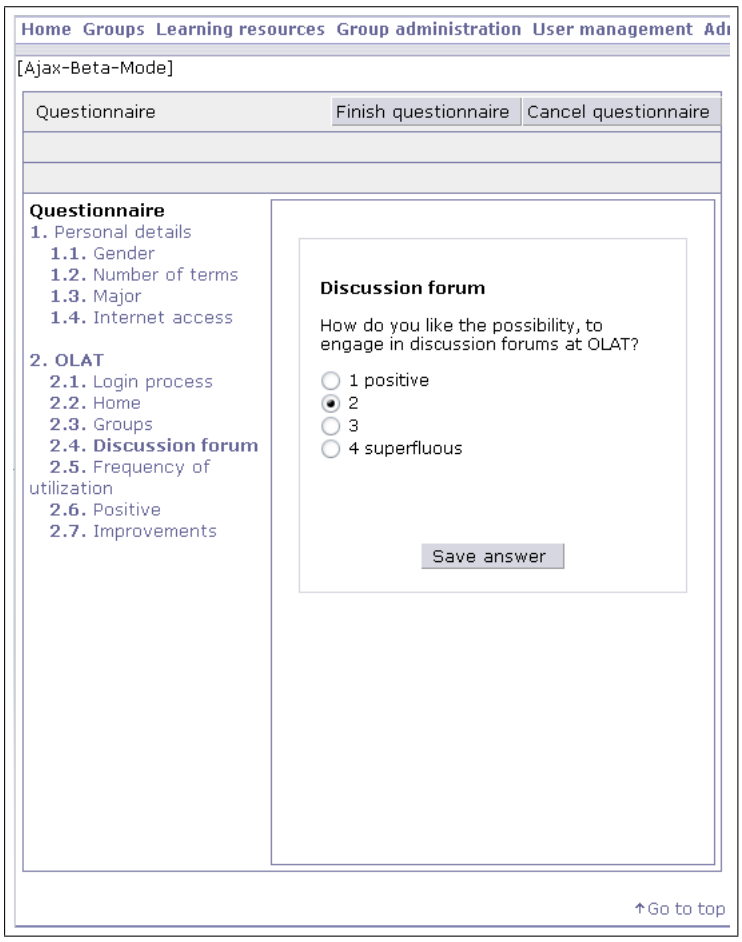


Abbildung 4: OLAT Beispielaufgaben, Quelle:

2.3.2.2 ECQuiz

ECQuiz[7] ist eine Erweiterung des Plone[8] CMS und ist unter der GPL-Lizenz frei verfügbar. Es bietet Multiple-Choice und Freitextaufgaben an. ECQuiz unterstützt IMS QTI als Datenformat für die Aufgaben, anders als OLAT wird jedoch die neuere Version v2.0 verwendet. ECQuiz selbst kam nun aufgrund seiner auf Python basierenden Implementierung nicht für eine Integration in das Aufgabenframework in Frage, so dass die Erweiterbarkeit um neue Aufgabentypen nicht

[7] http://plone.org/products/ecquiz/
[8] http://plone.org

erst betrachtet wurde, bzw. analog zu der Betrachtung in Kapitel 3.1.3 erfolgen müsste.

Weitere LMS, die QTI verwenden sind Moodle[9] und ILIAS[10]. Die darin enthaltenen Implementierungen unterscheiden sich jedoch in ihrem Umfang nicht wesentlich von OLAT bzw. ECQuiz, so dass auf eine genauere Beschreibung verzichtet werden soll.

2.3.3 Math-Kit

Das Projekt Math-Kit, abgeschlossen im Jahr 2005, hatte als Ziel die Bereitstellung von Lehrmaterial aus dem Grundstudium Mathematik und dessen Anwendungen in der Form eines Baukastens. Dabei entstand neben einigen unterstützenden Werkzeugen zur Erstellung von Inhalten (Lyssa[11] und NyX) eine große, modulare Sammlung von Texten, Grafiken und Animationen aus diesem Fachgebiet. Das entstandene Material steht sowohl komplett als CD-Rom als auch in einer Webanwendung zur Verfügung, die es erlaubt, einzelne Wissensgebiete gezielt auszuwählen und als Skript herunterzuladen. Neben dem eigentlichen Lehrstoff stellt das Projekt bzw. die entstandene Anwendung auch Übungen zur Selbstkontrolle zur Verfügung. Ein Student hat damit die Möglichkeit sein eigenes Stoffverständnis zu überprüfen. Diese Tests sind jedoch ausschließlich als persönliche Lernfortschrittskontrolle verwendbar, da die Korrektur der Aufgaben lokal, also im Browser des Studenten, geschieht. Auch fehlt eine Verwaltung der erzielten Ergebnisse. Insgesamt ist Math-Kit damit interessant als didaktisch wertvolle Sammlung von Lehrtexten, jedoch nur eingeschränkt als Werkzeug im Rahmen von Übungsaufgaben einzusetzen, für Prüfungen in dieser Form gar nicht.

2.3.4 In2Math

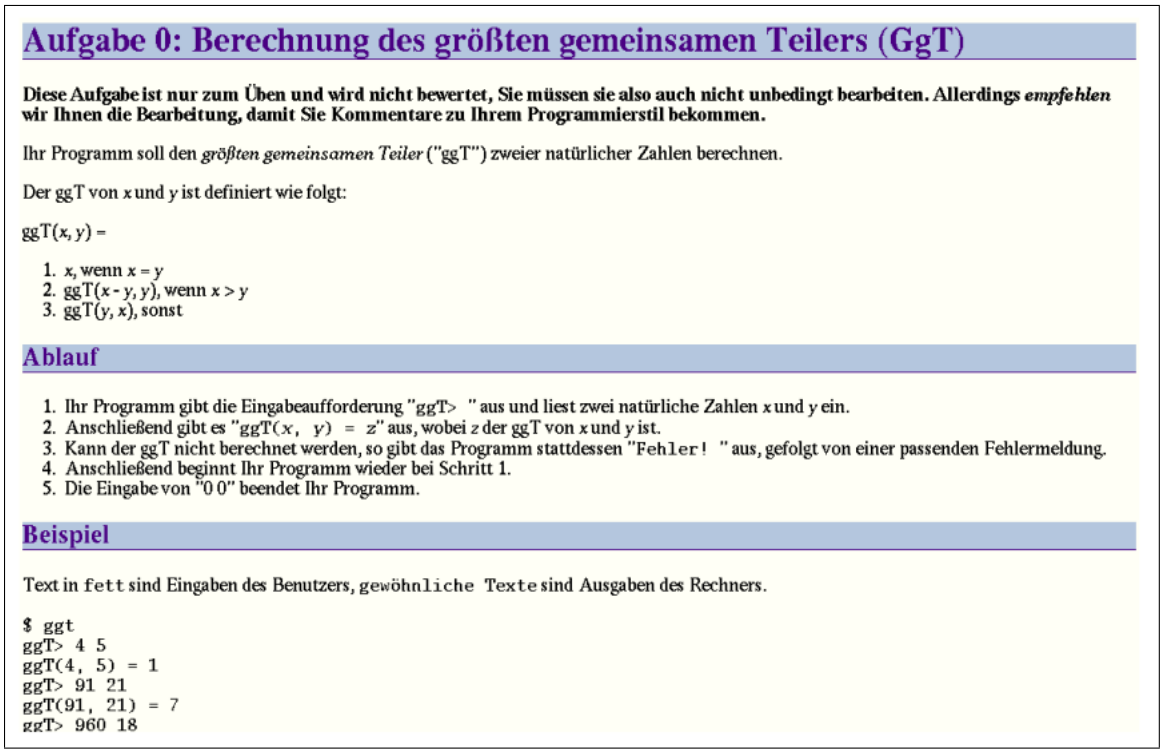
Die diesem Projekt zugrundeliegende Idee ist die Erstellung von Lehrmaterial zur Linearen Algebra mitsamt individuellen Rechenaufgaben für jeden Benutzer. Dazu gibt es einen internen Textkorpus, der inzwischen auch in [al01] auf CD als Aufgabensammlung erhältlich ist. In2Math will dabei die Studenten schon im Grundstudium an Computeralgebrasysteme heranführen, viele Aufgaben sollen mit einem solchen System bearbeitet werden. Die Onlineanwendung besteht aus einer Sammlung von Textbausteinen und Aufgabenschablonen, die konfigurierbare Parameter enthalten, so dass aus einer Aufgabe unterschiedliche Instanzen

[9] http://www.moodle.org
[10] http://www.ilias.de
[11] http://www.math-kit.de/downloads/index.html

generiert werden können. Insgesamt sind bisher 4700 verschiedene Aufgaben verfügbar. Die so erstellten Skripte und Aufgabenblätter werden als Dokument heruntergeladen und offline bearbeitet, eine automatische Korrektur ist nicht vorgesehen, genauso fehlt eine Uploadmöglichkeit für studentische Lösungen. Das System ist also eher eine konfigurierbare Textbasis als ein E-Assessment-System für lineare Algebra.

2.3.5 Praktomat

Praktomat[12] ist ein an der Universität Passau seit 1999 entwickeltes OpenSource-Softwaresystem zur Durchführung von Programmierpraktika. Es besteht zum einen aus einer Webapplikation zur Verwaltung von Praktika, Studenten, Aufgaben etc. sowie einer auf DejaGnu[13] basierenden Testumgebung. Die aktuelle Version ist v4.2, der Gesamtumfang der Funktionalität ist dementsprechend umfangreich. Für eine umfassende Programmierausbildung ist nun nicht ausschließ-



Aufgabe 0: Berechnung des größten gemeinsamen Teilers (GgT)

Diese Aufgabe ist nur zum Üben und wird nicht bewertet, Sie müssen sie also auch nicht unbedingt bearbeiten. Allerdings *empfehlen* wir Ihnen die Bearbeitung, damit Sie Kommentare zu Ihrem Programmierstil bekommen.

Ihr Programm soll den *größten gemeinsamen Teiler* ("ggT") zweier natürlicher Zahlen berechnen.

Der ggT von *x* und *y* ist definiert wie folgt:

ggT(*x*, *y*) =

1. *x*, wenn *x* = *y*
2. ggT(*x* - *y*, *y*), wenn *x* > *y*
3. ggT(*y*, *x*), sonst

Ablauf

1. Ihr Programm gibt die Eingabeaufforderung "`ggT> `" aus und liest zwei natürliche Zahlen *x* und *y* ein.
2. Anschließend gibt es "`ggT(x, y) = z`" aus, wobei *z* der ggT von *x* und *y* ist.
3. Kann der ggT nicht berechnet werden, so gibt das Programm stattdessen "`Fehler! `" aus, gefolgt von einer passenden Fehlermeldung.
4. Anschließend beginnt Ihr Programm wieder bei Schritt 1.
5. Die Eingabe von "0 0" beendet Ihr Programm.

Beispiel

Text in `fett` sind Eingaben des Benutzers, `gewöhnliche Texte` sind Ausgaben des Rechners.

```
$ ggt
ggT> 4 5
ggT(4, 5) = 1
ggT> 91 21
ggT(91, 21) = 7
ggT> 960 18
```

Abbildung 5: Praktomataufgabe „Größter gemeinsamer Teiler“, Quelle: [LS07]

[12] http://www.fim.uni-passau.de/de/fim/fakultaet/lehrstuehle/softwaresysteme/forschung/praktomat.html
[13] http://www.gnu.org/software/dejagnu/

lich die Funktionalität von geschriebenem Code wichtig. Deshalb geht in die Bewertung einer Aufgabe nicht nur die semantische Korrektheit des geschriebenen Quelltextes ein, sondern auch der Programmierstil, der mit Hilfe von CheckStyle[14] geprüft wird. Damit können z.B. ausreichende Dokumentationen oder Programmierkonventionen überprüft werden.

DejaGnu ist ein Framework zum Testen von Programmen unter Linux. Es basiert im Wesentlichen auf einem Vergleich von textuellen Ausgaben eines Programms mit erwarteten Werten. Es besteht aus einer Sammlung von Programmen, die mit Hilfe von zu definierenden TCL-Skripten Regressionstests durchführen. Trotz der durch dieses Werkzeug angebotenen Funktionen lassen sich also nur solche Programme testen, die Ein- und Ausgaben auf die Kommandozeile unterstützen. Es ist also nicht ohne weiteres möglich, Unittests mit etablierten Testwerkzeugen wie etwa JUnit für Java™ durchzuführen. Die Anforderungen an eine Aufgabenlösung sind also höher, da der Benutzer neben der Implementierung der eigentlichen Logik seiner Lösung auch Funktionen schreiben muss, die die Datenein- und -ausgabe beinhalten. Speziell bei Aufgaben, die ganz zu Beginn der Programmierausbildung gestellt werden stellt dies eine nicht geringe Hürde dar.

Trotz dieser Einschränkungen bietet der Praktomat einige zusätzliche interessante Funktionen: Die Aufgaben sind parametrisierbar, um das Kopieren von Lösungen zu erschweren. Dazu werden in der Aufgabenstellung Alternativen für Textbausteine definiert, die ausgetauscht werden können, um Varianten der Aufgabe zu erzeugen. Denkbar sind also feste Faktoren für z.B. zu implementierende numerische Algorithmen, unterschiedliche zu implementierende Algorithmen für ein Problem, etwa Dijkstra vs. greedy für die Suche nach dem kürzesten Pfad in einem Graphen. Auf diese Art und Weise lassen sich bis zu acht Variationspunkten mit jeweils zwei Ausprägungen angeben, es ergeben sich also maximal 256 Ausprägungen einer Aufgabe.

Eine weitere für die Programmierausbildung wichtige Funktionalität ist die Möglichkeit, Lösungen anderer Studenten zu bewerten und zu kommentieren. Damit bekommen die Nutzer die Möglichkeit die Lesbarkeit ihrer Lösung bewertet zu bekommen.

Der Praktomat lässt sich um alle Arten von Aufgabentypen ergänzen, deren Korrektheit mit Hilfe von dejaGnu festgestellt werden kann. Dazu wären entsprechende Testtreiber in TCL zu schreiben, die die abgegebenen Lösungen mit den jeweils erwarteten Werten vergleichen. Das System ist also prädestiniert für Programmieraufgaben in den verschiedensten Sprachen. Einzig der Aufwand für die Einarbeitung in den verwendeten TCL-Dialekt und die pro Aufgabentyp neu zu schreibenden Testtreiber ist jeweils recht hoch.

[14] http://checkstyle.sourceforge.net/

2.3.6 E-Claus

E-Claus steht für „Electronic Correction of onLine Assignments at the University of Stuttgart“ und wurde im Rahmen des in der Abteilung Formale Konzepte[15] an der Universität Stuttgart durchgeführten Studienprojekts ECÜ ab 2001 entwickelt. Es ist speziell ausgerichtet auf die Verwaltung von Übungen und Programmierpraktika im Grundstudium Informatik.

Diese Plattform bietet einen Upload für Aufgabenlösungen, die dann von einem Tutor heruntergeladen und bewertet werden. Diese Bewertung erfolgt momentan semiautomatisch, d.h. der Tutor verwendet Skripte, um Lösungen zu testen und zu bewerten. Leider gibt es auch hier keine Möglichkeit, Unittests zu definieren, lediglich eine Auflistung von Parametern und den erwarteten Rückgabewerten kann spezifiziert werden. Damit lassen sich in der Praxis nur rein algorithmische Aufgabenstellungen sinnvoll umsetzen. Die Bewertung von Lösungen erfolgt erst nach Beendigung des jeweils gesetzten Bearbeitungszeitraumes, ein unmittelbares Feedback zur Richtigkeit einer eingesandten Lösung ist also nicht möglich. Auch diese Plattform verwendet, wie auch der Praktomat, zusätzlich zu Funktionstests *CheckStyle*, um die Einhaltung eines definierten Programmierstiles zu überprüfen. Eine Besonderheit stellt die Plagiatkontrolle dar. Diese Funktion, umgesetzt im Rahmen einer Diplomarbeit von Herrn König [Kön05], bietet vor der Freigabe einer Bewertung die Möglichkeit, die aktuelle Lösung mit anderen zu vergleichen. Überschreitet die Ähnlichkeit einen definierten Schwellenwert, wird die Lösung als mögliches Plagiat gekennzeichnet und dem Tutor bzw. Dozenten zur Begutachtung vorgelegt. Diese Plagiaterkennung bedient sich dabei verschiedensten Algorithmen, so dass reine Umbenennungen bzw. unterschiedliche Einrückungen keine verschleiernde Relevanz mehr haben.

E-Claus ist in der aktuellen Version vollständig auf die Verwendung für Java™-Programmieraufgaben ausgelegt. Es wurde keine Erweiterbarkeit um neue Aufgabentypen vorgesehen.

2.4 Anforderungen der Benutzergruppen an der Universität Leipzig

Es gibt eine große Anzahl von Benutzern bzw. Interessenten an E-Testing an der Universität Leipzig. Das Hauptaugenmerk liegt dabei auf der Unterstützung der Lehre durch vorlesungsbegleitende Übungen und auf elektronisch durchgeführten Prüfungen. Es sollen nun kurz die individuellen Situationen und Anforderungen einiger der potentiellen Benutzer an ein E-Testingsystem zugeschnitten auf die

[15] http://www.informatik.uni-stuttgart.de/fmi/fk/

jeweiligen Erfordernisse aufgeführt werden. Dabei werden, aufbauend auf der Arbeit von Berger [Ber07b], nur die neuen fachbereichsspezifischen Anforderungen erfasst. Der zur Ermittelung dieser Anforderungen verwendete Fragebogen ist im Anhang A.1 enthalten.

2.4.0.1 Pädagogik

Am Institut „Allgemeine Pädagogik“ werden seit mehreren Semestern erfolgreich elektronische Prüfungen mit Hilfe des Aufgabenframeworks des elatePortals durchgeführt. Es werden alle vier angebotenen Aufgabentypen verwendet. Die einzig verbleibende Lücke zu Papierprüfungen stellt jedoch die fehlende Möglichkeit der Eingabe von Zeichnungen dar. Es wurde also ein neuer Aufgabentyp zur Erstellung von Skizzen, Zeichnungen und Vervollständigung von Graphiken durch den Benutzer gewünscht.

/LF0010/
Es soll eine erweiterte Freitextaufgabe mit graphischen Interaktionsmöglichkeiten geben. Es sollen vorgegebene Zeichnungen ergänzt und neue Zeichnungen erstellt werden können.

2.4.0.2 Sprachenzentrum und Herder-Institut

Kennzeichnend für die am Sprachenzentrum durchgeführten Tests sind das Aufnehmen und Abspielen von Sprache. Das Aufgabenframework unterstützt zwar prinzipiell beliebige darstellbare Medien, jedoch fehlen in der aktuell vorliegenden Version schon fertige Audiofunktionen. Eine denkbare Erweiterung des Frameworks wäre also die Einbindung des Onyx-Testsystems zur Wiederverwendung dieser Werkzeuge. Auch wird eine Möglichkeit zur Definition von Abhängigkeiten zwischen Fragen benötigt, um zum Beispiel für Sprachstandstests weiterführende Fragen je nach Korrektheit bisheriger Antworten stellen zu können. Besonders wird das für Sprachstandstests benötigt um Fragen auf den Umfang des vorhandenen Vokabelwissens abstimmen zu können.

/LF0020/
Es wird die Ausgabe von Audiodaten sowie die Aufnahme von Sprache mit Hilfe eines angeschlossenen Mikrophons benötigt.

/LF0030/
Schon existierende Tests basieren auf dem Datenformat des IMS-QTI. Ein Import bzw. eine Wiederverwendung dieser Tests ist notwendig.

/LF0040/
Es soll möglich sein, das Anzeigen von Fragen mit vorgegebenen Regeln zu steuern. Im Speziellen sollen die Ergebnisse von beantworteten Fragen als Vorbedingungen angegeben werden können.

2.4.0.3 Datenbanksysteme

Der bisherige Einsatz des LOTS dient nur dem Übungsbetrieb, wird jedoch nicht für Prüfungen oder für Prüfungsvoraussetzungen verwendet. Es fehlt die Möglichkeit, Studenten entsprechend zufällig erstellte MC-Aufgaben zu präsentieren, um Abschreiben von Lösungen zu erschweren. Die bisherigen Erfahrungen zeigen eine hohe Anzahl von abgeschriebenen Lösungen. Diese Erkenntnis steht im Widerspruch zu den Ergebnissen einer empirischen Studie zu diesem Thema von Wielicki [Wie06], sollte also genauer untersucht werden.

Die Interaktion mit einer Datenbank mit Hilfe von SQL ist als Kernbestandteil des Umgangs mit Datenbanken eine wichtige Interaktionsform für Aufgaben. Die Funktionalität des SQL-Trainers ist prinzipiell gut als Aufgabentyp in das Aufgabenframework integrierbar. Leider ist das angestrebte Nachfolgeprojekt O3 [Eck05] bisher nicht öffentlich verfügbar, es existiert also zur Zeit keine aktiv weiterentwickelte Version des SQL-Trainers. Eine Integration in das Aufgabenframework könnte eine Weiterentwicklung anstoßen.

/LF0050/
Der SQL-Trainer soll als Aufgabentyp/Interaktionsform genutzt werden können.

/LF0060/
Eine automatische Korrektur von SQL-Aufgaben soll möglich sein.

/LF0070/
MC-Aufgaben sollen randomisiert werden können.

2.4.0.4 Theoretische Informatik im Grundstudium

Analog zum Mathematikgrundstudium dominieren im Übungsbetrieb Beweisaufgaben. Zusätzlich wird in einigen Vorlesungen Autotool eingesetzt. Es werden damit Aufgaben gestellt, die das Verständnis theoretischer Konzepte durch Anwendung auf konkrete Instanzen von Problemen fördern. Aufgrund der internen Umstrukturierung und der eingestellten Weiterentwicklung der Autotool-Weboberfläche wurde ein LMS gesucht, welches Autotool einbindet und für Übungen zu Verfügung stellt.

/LF0080/
Autotool soll als Aufgabentyp verfügbar sein.

2.4.0.5 Programmierausbildung

Im Grundstudium wird als Einstiegsprogrammiersprache im Informatikstudiengang zur Zeit Java™[GJSB05] verwendet. Der praktische Teil der Ausbildung wird durch schriftliche Übungen begleitet. Dabei werden die eingereichten Lösungen nur visuell auf offensichtliche Fehler überprüft bzw. auf fehlende Dokumentation hingewiesen. Es erfolgt aber aufgrund dessen kein Übersetzungsvorgang und kein zur Aufgabe passender Unittest wie ihn z.B. Praktomat (siehe Seite 16) bietet. Es wäre also ein Aufgabentyp notwendig, der die Abgabe von implementierten Java™-Quellcodedateien ermöglicht und die Korrektheit mit Hilfe von automatischen Tests überprüft. Derselbe Ansatz kann dann auch für andere Programmiersprachen verwendet werden, etwa um neue Programmierparadigmen üben zu können, wie z.B. funktionale Programmierung.

/LF0090/
Es soll möglich sein, Java™-Quellcode automatisch zu übersetzen und mit Unittests zu testen.

/LF0100/
Neue Programmiersprachen müssen als Aufgabentyp eingebunden werden können.

2.4.0.6 Mathematik

Im Grundstudium Mathematik sind Übungsaufgaben ein wesentliches Element der Vorlesungen. Es werden wöchentlich Übungsblätter ausgegeben, die handschriftlich abgegeben und von Tutoren korrigiert werden. Es werden zwei Aufgabentypen verwendet: Rechenaufgaben und Beweisaufgaben. Rechenaufgaben ließen sich prinzipiell mit Hilfe von MC-Aufgaben abbilden (siehe auch die beschriebene Vorgehensweise in [BRS06]), jedoch sind bei der Korrektur von Rechenaufgaben nicht nur die Ergebnisse interessant, sondern insbesondere auch die zugehörigen Rechenwege. Damit ergäbe sich also genauso wie für Beweisaufgaben nur Freitextaufgaben als Aufgabentyp. Die automatische Korrigierbarkeit ist nach aktuellem Forschungsstand nicht möglich. Davon abgesehen ergäbe sich die Schwierigkeit einer entsprechenden Notation für mathematische Ausdrücke, etwa durch LaTeX. Diese Beschreibungssprache würde jedoch nach den vorhandenen Erfahrungen eine zu hohe Hürde für die Bearbeitung der Fragen durch Studenten

im Grundstudium darstellen. (Es ergibt sich insgesamt also keine Anforderungen an ein E-Testingsystem, da kein entsprechender Bedarf besteht, bzw. eine Umsetzung des momentan etablierten Übungsbetriebs auf elektronische Abgaben keinen Mehrnutzen bringen würde.) Eine Alternative wäre eine Verknüpfung von MC-Aufgaben mit einem Computeralgebrasystem CAS als Hilfsmittel zur Beantwortung der Fragen analog zur Praxis in LOTS.

/**LF0110**/
: Es soll eine benutzerfreundliche Notationsmöglichkeit für mathematische Ausdrücke als Eingabemöglichkeit für Freitextaufgaben geben.

/**LF0120**/
: Ein Anbindung an ein CAS kann als Hilfsmittel zur Fragenbeantwortung in Fragen genutzt werden.

2.5 Fazit und Aufgabenstellung der Arbeit

Das vorhandene Aufgabenframework deckt umfassend die Bedürfnisse einer elektronischen Prüfung am Institut „Allgemeine Pädagogik“ ab. Es zeigte sich jedoch, dass andere Anwender eine große Bandbreite an zusätzlichen unterschiedlichen Interaktionsmöglichkeiten und Aufgabentypen benötigen. Zum Teil existieren schon Lösungen, wie z.B. der SQL-Trainer oder Autotool, deren Verwendung im Aufgabenframework durchaus sinnvolle Ergänzungen darstellen würden. Die existierende formale Spezifikation eines Prüfungsablaufes kombiniert mit einer flexiblen Integrationsmöglichkeit für domainspezifische Aufgabentypen kann als Grundlage für eine umfassende Unterstützung für Übungsaufgaben und auch Prüfungen an den einzelnen Lehrstühlen dienen. Der Hauptschwerpunkt dieser Arbeit liegt auf der Erweiterung der aktuellen Architektur des Aufgabenframeworks. Es soll eine generische Schnittstelle geschaffen werden, die mit geringem Aufwand existierende Werkzeuge zur Erstellung und Korrektur von Aufgaben integrieren kann.

Für den praktischen Teil dieser Arbeit ergeben sich aus den erhobenen Anforderungen der unterschiedlichen Abteilungen also die nun folgenden Entwicklungsaufgaben.

2.5.1 Einfache Integrierbarkeit neuer Aufgabentypen

Das bestehende Aufgabenframework umfasste, wie in Kapitel 2.1 beschrieben, vier Aufgabentypen. Diese eignen sich recht gut zum Abfragen von Wissen, decken jedoch nicht genügend Szenarien ab. Es fehlten die verschiedenen fachspezifischen Aufgabentypen, wie z.B. Programmieraufgaben in der Informatik,

Computeralgebra-Aufgaben in der Mathematik, Sprechaufzeichnungen in der Fremdsprachenausbildung etc. Um das Framework erweitern zu können ist ein sehr detailliertes Wissen um die Struktur und den Aufbau desselben nötig. Außerdem wäre der Austausch von neu implementierten Aufgabentypen zwischen unabhängigen Entwicklern nur sehr umständlich möglich gewesen. Es war also gefordert, die bestehende Struktur um Schnittstellen zu erweitern, die folgende Eigenschaften aufweisen:

- Implementierungen der Schnittstelle stellen neue Aufgabentypen dar
- Aufgabentypen sind unabhängig von der Implementierung des Aufgabenframeworks
- Aufgabentypen sind zwischen Instanzen des Aufgabenframeworks frei austauschbar
- Aufgabentypen sind nicht in ihren Interaktionsmöglichkeiten mit dem Nutzer eingeschränkt

Eine Implementierung einer neuen Aufgabe entspricht in ihrer Gesamtheit also einem Plugin des Aufgabenframeworks.

Als Beispiele für mögliche Ausprägungen von Plugins sind die folgenden drei Aufgabentypen umzusetzen:

2.5.2 Zeichenaufgaben

Die bisherigen Aufgabentypen ließen nur textuelle Interaktionen des Nutzers zu, eine starke Einschränkung der Ausdrucksmöglichkeiten. In traditionellen papierbasierten Aufgabenbearbeitungen ließen sich auch Zeichnungen und Skizzen einbringen. Diese Interaktion sollte es daher auch im Aufgabenframework geben. Speziell im Hinblick auf eine mögliche zukünftige Einführung und Verwendung von Tablet-PCs ist dieser Aufgabentyp interessant.

2.5.3 Autotoolaufgaben

Die Entwickler von Autotool wollen sich in Zukunft möglichst ausschließlich auf die Weiterentwicklung des semantischen Teils konzentrieren, statt zusätzlich Webentwicklung betreiben zu müssen. Die Aufgaben der momentan durch eine Weboberfläche samt Datenbankanbindung realisierten Übungsverwaltung soll durch

dafür entwickelte Lernplattformen ersetzt werden. Eine Einbindung in das Aufgabenframework des elatePortal-Projekts stellt deshalb eine sinnvolle Trennung von Verwaltung und semantischem Teil des Autotools dar.

2.5.4 Automatisches Validieren von Java™-Programmieraufgaben

Ein wesentlicher Teil der studentischen Hochschulausbildung im Grundstudium ist die Vermittlung von algorithmischem Wissen. Der Lernfortschritt lässt sich dabei neben schriftlichen Aufgaben zur Analyse der Komplexität von Algorithmen auch über von den Studenten geschriebene Programme verifizieren. Wegen der Bedeutung, die die Programiersprache Java™ aktuell sowohl in kommerziellen Projekten als auch in der universitären Ausbildung hat, ist eine Möglichkeit, in Java™ implementierte Programme automatisch auf Korrektheit prüfen zu können gefordert.

2.5.5 Angepasster Workflow für Übungsaufgaben

Das Aufgabenframework des elatePortals ist prinzipiell nicht auf bestimmte Abläufe der Aufgabenbearbeitungen festgelegt. Es gibt keine inhärenten Restriktionen für alle irgend gearteten Interaktionsschemata. Leider existiert aber auch kein standardisierter Mechanismus, um diese dem Framework hinzuzufügen. Dazu wäre die Integration einer Workflow-Engine nötig gewesen, diese wurde jedoch in den bisherigen Einsatzszenarien nicht benötigt. Der bisher einzige vollständig implementierte Workflow ist derzeit auf die Bearbeitung von Komplexaufgaben ausgelegt, also einer klausurartigen Zusammenstellung von Aufgaben, die in einem fest vorgegebenen Zeitraum abgearbeitet werden müssen.

Die geforderten neuen Aufgabentypen lassen sich nun nicht ausreichend komfortabel in dieser Art der Aufgabenbearbeitung verwenden. Erfahrungsgemäß ist selten der erste Implementierungsversuch einer Programmieraufgabe erfolgreich. Neben syntaktischen Fehlern schleichen sich sehr schnell auch kleinere semantische Fehler (wie z.B. Zuweisungen statt Vergleichen etc.) ein. Deshalb ist es nicht angeraten, den Studenten nur einen einzigen Lösungsversuch zu gestatten. Aber selbst das Zugestehen mehrerer Lösungsversuche führt nicht zu einer komfortableren Benutzung. Der vorhandene Workflow zwingt den Studenten zu einer „Abgabe“ des Lösungsversuch, nur um sich im Anschluss den Korrekturstand ansehen zu können. Wünschenswert ist hier also eine Möglichkeit der direkten Rückmeldung über Erfolg oder Misserfolg des Lösungsversuches.

3 Architektur

Wie muss eine Architektur beschaffen sein, damit sie nicht nur die Anforderungen an ein allgemeines Werkzeug zur universitären Kompetenzmessung erfüllt, sondern darüber hinaus auch die Integration verschiedenster Testwerkzeuge erlaubt? Ein Werkzeug zur Durchführung von vorlesungsbegleitenden Übungen und Prüfungen als Lernfortschrittskontrolle, das alle Anforderungen an ein solches innerhalb einer Universität abdeckt, ist ein in seiner Gesamtheit kaum vollständig spezifizierbares Gebilde. Die Vielfalt an unterschiedlichen Aufgabentypen, Interaktionsformen innerhalb dieser Typen, sowie die unterschiedlichen Prüfungsabläufe sind in den einzelnen Fachgebieten zu verschieden, als dass ein einziges Werkzeug diese umfassend abbilden könnte. Es ist also erstrebenswert eine Plattform zu haben, welche es erlaubt, beliebige bereits bestehende Softwareartefakte aus dem E-Assessmentbereich als Dienstanbieter wiederzuverwenden. Die dazu notwendige Flexibilität kann nur erreicht werden, wenn bestehende Software als Komponenten mit festgelegten Serviceschnittstellen versehen werden und so auch außerhalb ihres eigentlichen Umfeldes genutzt werden können. Das folgende Kapitel beschäftigt sich daher mit dem aktuellen Stand der Literatur zum Thema SOA, der serviceorientierten Architektur, speziell im Umfeld des Aufgabenbereiches Assessment aus der Domäne E-Learning, sowie dem nach diesen Prinzipien entwickelten Standard OSGi.

In den folgenden Teilkapiteln soll dazu die Sichtweise von drei Projekten bzw. Spezifikationen dargestellt werden. Dabei wird der betrachtete Bereich immer spezieller, angefangen bei dem allgemeinen Aufbau von Architekturen im E-Learningbereich wie sie im *IMS Abstract Framework* (Kapitel 3.1.1) beschrieben wird, über die Darstellung der Struktur der Anwendungsdomäne Assessment als Teilbereich des E-Learnings im *Framework Reference Model for Assessment* (Kapitel 3.1.2) bis hin zur Struktur des Ablaufs eines allgemeinen Tests laut der *IMS Question & Test Interoperability Specification* (Kapitel 3.1.3). Aufgrund der diesen Projekten gemeinsamen Eigenschaften sollen danach die Gemeinsamkeiten der serviceorientierten Architekturansätze kurz beschrieben und danach die Architektur des Aufgabenframeworks im elatePortal, die im Rahmen dieser Arbeit entstanden ist bzw. erweitert wurde, analysiert werden. Anschließend soll die bestehende Architektur des elatePortal-Aufgabenframeworks beschrieben und auf die notwendigen Erweiterungen eingegangen werden.

3.1 Referenzmodelle für E-Assessment

Es gibt in der Literatur eine Reihe von Projekten, die sich mit der Frage beschäftigen, wie eine Architektur beschaffen sein muss, um den Teilbereich E-Assessment

der Anwendungsdomäne E-Learning problemadäquat abbilden zu können. Davon sollen exemplarisch drei vorgestellt werden, die auf unterschiedlichen Granularitätsstufen eine solche Architektur modellieren. Diese drei Projekte stellen gleichsam einen jeweils steigenden Detaillierungsgrad dar.

3.1.1 IMS Abstract Framework

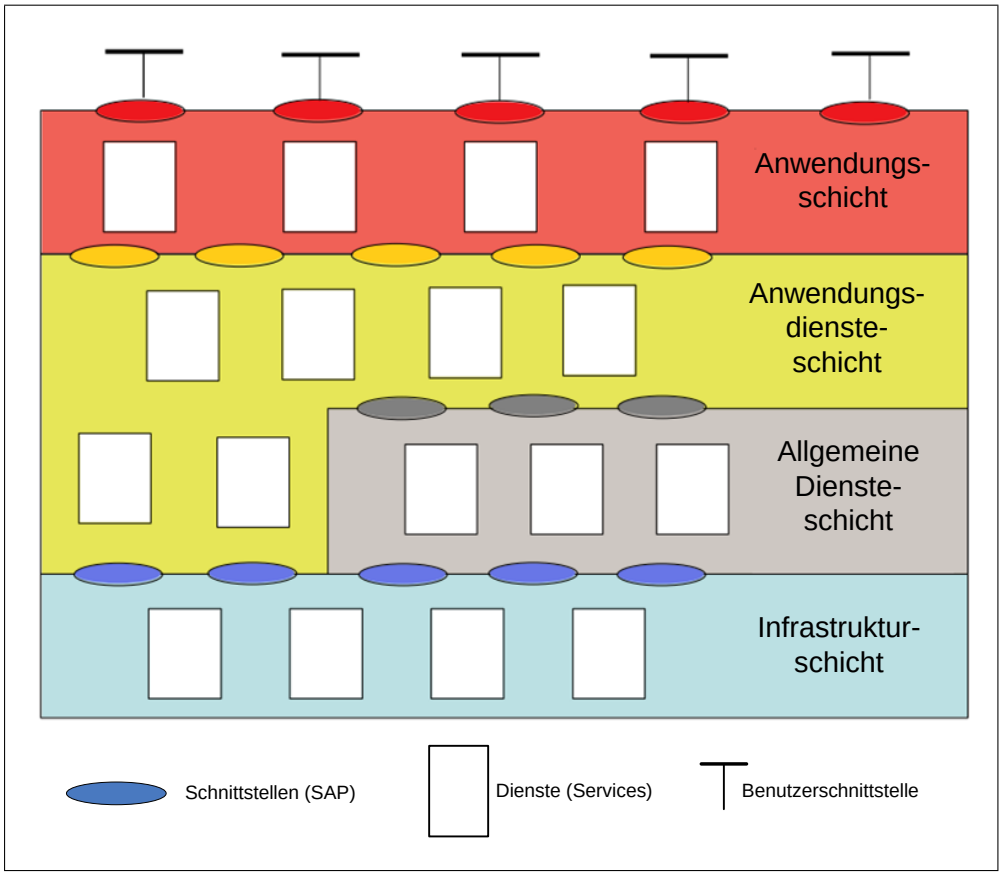


Abbildung 6: IMS Abstract Framework Schichtenmodell, nach [IMS03a]

IMS hat neben IMS QTI (siehe Kapitel 3.1.3) für die Beschreibung von Tests auch eine Reihe anderer Spezifikationen für Datenbeschreibungsformate für die unterschiedlichen Teilbereiche von E-Learning entwickelt. Um all diese Spezifikationen systematisch ordnen zu können wurde das theoretische *IMS Abstract Framework* IAF veröffentlicht. Dabei handelt es sich nicht um die Architekturbeschreibung eines konkreten Softwaresystems, sondern vielmehr um ein Gedankengebäude, eine Skizze der Struktur eines solchen Systems. Mit seiner Hilfe sollen die schon veröffentlichten Spezifikationen in einen Rahmen eingefügt und darüber

hinaus Bereiche identifiziert werden, für welche weitere Datenbeschreibungen erstellt werden müssen. Es sei jedoch angemerkt, dass wegen der Allgemeinheit der in diesem Framework enthaltenen Konstrukte durchaus Vergleiche mit real existierenden Architekturen gezogen werden können.

Das IAF ist in Schichten aufgeteilt wie sie Abbildung 6 zeigt. Jede Schicht enthält Dienste (Services) mit einer jeweils festen Schnittstelle. Auf diese Dienste kann mit Hilfe der *Service Access Points* (SAP) zugegriffen werden. Die oberste Schicht ist die Anwendungsschicht (application layer). Hier interagieren Benutzer mit dem System mit Hilfe verschiedener Anwendungen und Oberflächen. Diese nutzen die darunterliegenden Anwendungsdienste, welche wiederum sowohl die allgemeinen Hilfs- als auch die Kommunikationsdienste der Infrastrukturschicht verwenden. Eine große Zahl solcher Services werden im IAF aufgezählt und deren Semantik sowie Kommunikationsverhalten beschrieben. Eine exemplarische Auswahl der Services aus [IMS03b] ist etwa:

Application Services Layer Anwendungsdiensteschicht

- Assessment
- Kalender
- Kollaborationsdienste
- Inhaltsverwaltung etc.

Common Services Layer Allgemein nützliche Dienste

- Authentifizierung und Single Sign On (SSO)
- Autorisation
- Logging
- Rechtemanagement etc.

Infrastructure Layer Dienste für den physischen Datenaustausch (z.B. über XML-Dokumente)

Die Granularität der definierten Services zeigt beispielhaft die Tabelle 1. In dieser ist zu erkennen, dass im IAF zwar die grundlegende Absicht sowie die Abhängigkeit zwischen den identifizierten Services aufgeführt werden, jedoch deren innerer Aufbau bzw. das Zusammenspiel der zum Erreichen der jeweils angestrebten Funktionalität notwendigen Services ausgespart wurde. Für einzelne ausgewählte Services existieren jedoch konkrete Spezifikationen durch IMS, wie etwa IMS QTI, welches im Kapitel 3.1.3 vorgestellt werden wird.

Assessment	
Service Description:	A service that is responsible for the presentation and reporting of an appropriate low-stakes/high stakes assessment. The assessment presentation and reporting is managed at the group and individual level.
Service Access Points:	unspecified
Core Attributes:	unspecified
Core Components:	• Assessment • Item • Object-bank • Section • Result Report
Common Service Dependencies:	unspecified
Infrastructure Dependencies:	unspecified

Tabelle 1: Assessment - ein Service des IAF

Wichtig sind neben diesen ausformulierten Services vor allem die Prinzipien, die diesem Framework zugrunde liegen: Die Betonung liegt stark auf einem service- bzw. komponentenorientierten Ansatz, der aus Schichten aufgebaut ist und dessen Kommunikationsstruktur unabhängig von einer konkreten Ausprägung gehalten wird. Zwar wird angedeutet, dass die Kommunikation über etablierte Abbildungen auf XML bevorzugt werden (etwa SOAP für Webservices), jedoch ist kein festes Format vorgegeben. Es wird angestrebt, ein solches festes Format zu entwickeln.

Diese Framework ist als Modell für die Gesamtheit der E-Learning-Domäne gedacht, die darin spezifizierten Services und Elemente gehen also deutlich über den Bereich des Assessments hinaus. An dem Informationsgehalt der Beschreibung des Services *Assessment* (siehe Tabelle 1)ist gut zu erkennen, dass die genauen Details nicht Aufgabe des IAF sind. Damit ist es zu grobgranular, um Aussagen über die Struktur bzw. die Services, die innerhalb eines Assessments genutzt werden, zu treffen. Es ist jedoch auch nicht als Bauplan für eine konkrete Architektur zu verstehen. Dennoch zeigt die Tatsache, dass dieses Modell, welches prototypisch den Aufbau von Systemen in der Domäne E-Learning und E-Assessment aufzeigen soll, die Bedeutung eines serviceorientierten Ansatzes.

3.1.2 Referenzmodell FREMA

Im letzten Teilkapitel wurde ein Modell für eine serviceorientierte Architektur beschrieben, welches die Domäne Assessment nur als abstrakten Service enthält, auf dessen inneren Aufbau jedoch nicht genauer eingegangen wird. Das JISC Board (Joint Information Systems Committee), ein Komitee von Experten aus der gehobenen Lehre im UK, haben sich nun die Aufgabe gestellt, sowohl die gegenwärtigen als auch die zukünftigen Bedürfnisse von Forschung und Lehre zu erfassen, zu beschreiben und Vorschläge für deren technische Unterstützung zu realisieren. Ein dabei im Rahmen des „JISC e-learning programme“ entstandenes Projekt ist FREMA (Framework Reference Model for Assessment) [DMW+06]. Dabei handelt es sich um ein Referenzmodell für Anwendungen im Bereich Assessment. Der Begriff Referenzmodell wird dabei im Sinne von Wilson et. al. gebraucht:

> “...A Reference Model is a selection of Services defined in one or more Frameworks together with rules or constraints on how those Services should be combined to realize a particular functional, or organisational goal. „[WBR04]

Es wird also keine konkrete Architektur beschrieben, sondern Anwendungsszenarien, verbindliche Begrifflichkeiten, benötigte Services und eine Zuordnung von

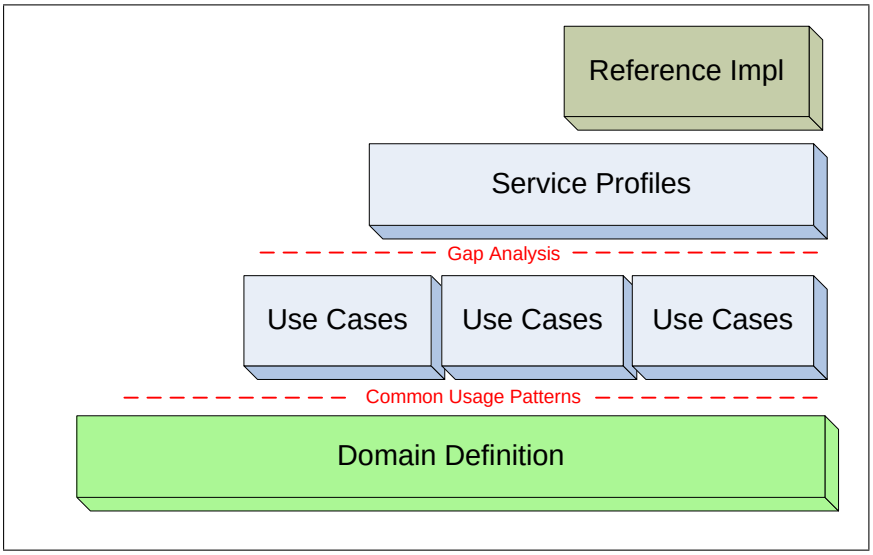


Abbildung 7: community reference model, nach [DMW+06, S. 4]

existierenden Softwareartefakten in diese entworfene Assessment-Landschaft. Die Autoren bezeichnen das entworfene Referenzmodell auch als „community reference model“, da es nicht von einem festen Gremium verabschiedet wurde, sondern durch die Mitarbeit einer ganzen Gemeinschaft von Interessierten entstanden ist. Einen Überblick zeigt die Abbildung 7. Realisiert wurde FREMA als stark verlinkte Website, das FREMA Semantic Wiki [16], welches eine Ontologie der Assessment-Domäne darstellen soll und neben den verschiedenen grundlegenden Begriffen auch eine Reihe wichtiger Standards aufführt (domain definition). In dieser Ontologie werden wichtige Begriffe verbindlich definiert sowie deren Bedeutungen und Abhängigkeiten aufgeführt. Darauf aufbauend werden *common usage patterns* identifiziert und daraus Anwendunsszenarien entwickelt, sogenannte *Use cases*. Diese führen wiederum zu einer Sammlung von allgemein verwendbaren Services. FREMA erlaubt eine *gap analysis*, d.h. eine Analyse, ob das Modell vollständig ist, also alle für einen Usecase notwendigen Services vorhanden sind bzw. alle Services den Usecases zugeordnet werden können und welche Implementierungen fehlen bzw. noch nicht in das Modell hineinpassen. Die Services, sowie deren Kollaborationen (service profiles), werden ebenso aufgeführt wie auch konkret existierende Implementierungen derselben, die entweder schon vor Projektbeginn existierten oder als Demonstration extra entwickelt wurden.

Das Referenzmodell bündelt abstrakte Services zu einem sogenannten

> “...Service Usage Model (SUM) for End-to-end Summative Assess-

[16] http://denebola.ecs.soton.ac.uk/wiki/index.php?title=Main_Page

ment.“ [DMW+06]

Das SUM zeigt dazu mit Hilfe mehrerer UML-Sequenzdiagramme die Kommunikation zwischen den definierten Services auf, welches insgesamt den Ablauf eines Assessments in seiner Gesamtheit beschreibt. Dieses Diagramm wurde in mehrere Abschnitte unterteilt und enthält das Erstellen eines Tests (Fragen und Termin), dessen Bearbeitung durch einen Testteilnehmer, die Korrektur und die Auswertung der Ergebnisse. Das sehr umfassende SUM ist in seiner Komplexität als Vorlage für den Entwurf und die Umsetzung eines allgemein einsetzbaren Assessmentsystems gut einsetzbar.

Die Verfügbarkeit bzw. Verwendbarkeit der im Modell aufgeführten konkreten Softwareartefakte ist jedoch nicht von durchgehend hoher Qualität, so dass der Einsatz etwa des Service „*Marking a Java Assignment v1.0*“ bzw. dessen Umsetzung „*AJM v1.0*“ als Aufgabentyp in dieser Arbeit nicht in Erwägung gezogen werden konnte.

3.1.3 IMS-QTI

IMS-QTI[17] ist ein vom IMS Global Learning Consortium verabschiedeter Standard zur inhaltlichen und strukturellen Beschreibung von Aufgaben und deren Zusammenfassung zu Aufgabenpools. Die aktuelle Version ist die v2.1. Er stellt im Sinne des IAF einen *Application Service* zur Durchführung von Assessments dar, wie er in der Tabelle 1 auf Seite 28 beschrieben wird. Der Standard liegt als XML-Schema vor, die Datenhaltung von Tests erfolgt also in XML. Ziel dieses Standards ist die Austauschbarkeit von Testbeschreibungen zwischen verschiedenen Autorenwerkzeugen und E-Learning-Plattformen. Der Inhalt dieser Beschreibungen geht jedoch über eine reine Formatbeschreibung hinaus. QTI beinhaltet außerdem eine hohe Konfigurierbarkeit der Aufgaben durch eine eingebettete auf XML basierende Programmiersprache. Der Standard gliedert sich in verschiedene Teile, die wichtigsten sind:

- Item and assessment model
- XML Binding
- Meta-data and Usage Data
- Results Reporting

[17] Question & Test Interoperability Specification, http://www.imsglobal.org/question/

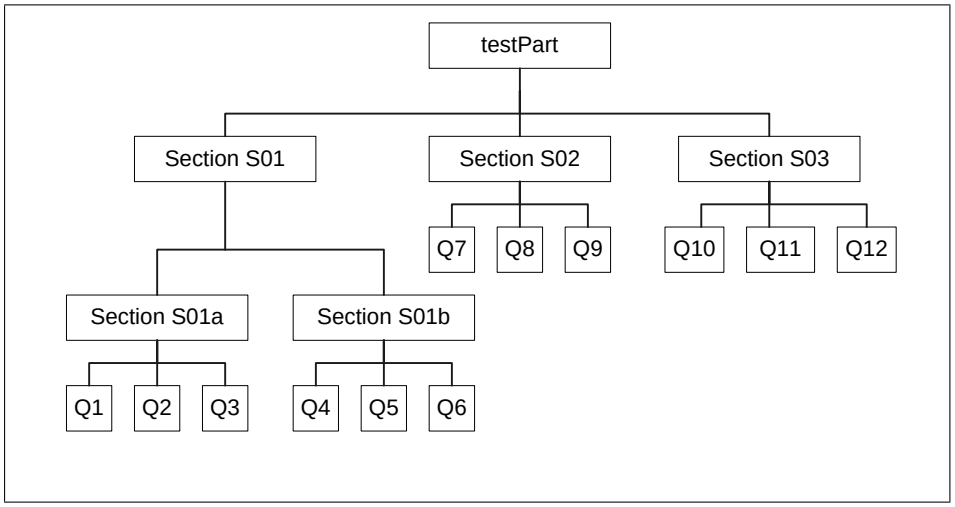


Abbildung 8: Aufbau eines *assessmentTest*, aus [IMS06]

XML Binding beschreibt die Abbildung von Datentypen auf XML, *Meta-data and Usage Data* enthält Daten über die Tests sowie statistische Auswertungen desselben und *Results Reporting* beschreibt die Struktur von Testergebnissen. Interessant im Kontext dieser Arbeit ist nun die Teilspezifikation *Item and assessment model*. Auf diese beiden Teilspezifikationen soll an dieser Stelle nicht genauer eingegangen werden, sie haben keinen ausreichenden Bezug zum Thema dieser Arbeit.

3.1.3.1 Item and assessment model

Den Kern des Standards bildet ein XML-Schema, welches detailliert die Strukturen der zulässigen XML-Beschreibungen von Testitems (Testaufgaben), deren Zusammenfassung zu Tests, Interaktionen, Feedbackinformationen, Response Processing (Bewertungsfunktionen) und Abhängigkeiten zwischen diesen beschreibt. Mit Hilfe dieser Beschreibungen lassen sich also austauschbare Fragenpools erstellen, welche mit verschiedenen Werkzeugen bearbeitet und visualisiert werden können. Die für diese Arbeit wesentlichen Elemente sollen nun kurz beschrieben werden.

Ein Test heißt *assessmentTest* und besteht aus einer Menge von Informationen über den Test bzw. Parameter zur Durchführung desselben, z.B. Art der Navigation (nicht-/ lineare Bearbeitung), Zeitlimits etc. Der wichtigste Teil ist die hierarchische Sammlung von Aufgaben (*Item*) in einer Baumstruktur, wie sie beispielhaft in der Abbildung 8 zu sehen ist. Diese Unterteilung in *testParts* und *sections* erlaubt die Erstellung von Aufgabenblöcken, die in Vorbedingungen adres-

siert werden können, welche die Auswahl und Reihenfolge der Präsentation dieser Blöcke erlauben. Jede *section* kann sowohl *sections* als auch *itemrefs* enthalten. Dies sind Referenzen auf jeweils ein *Item*. Die so entstehende logische Gliederung eines Tests erlaubt die Verwendung von Regeln, die Vorbedingungen, wie etwa die Korrektheit schon beantworteter Aufgaben, verwenden, um die Navigation eines Testteilnehmers zu steuern. Dieser Mechanismus wird etwa verwendet, um im vorgestellten Hochschulsprachtest (siehe S. 11) Sprachstandstests durchzuführen. Je nach Selbsteinschätzung des Teilnehmers („Waren die Fragen zu schwer, genau richtig oder zu leicht?“) wird der Schwierigkeitsgrad des Tests durch Auswahl der entsprechenden *testParts* variiert.

Der wesentlichste Bestandteil eines QTI-kompatiblen Fragenpools ist das sogenannte *Item*. Darunter wird eine Zusammenfassung aus Fragestellung, zulässigen Antworten, Hinweisen zur Bearbeitung, Transformationsanweisungen der Antworten in bewertbare Formate und deren Bewertungsregeln verstanden. Die Fragestellung selbst wird mit HTML beschrieben. Sie kann entweder fest definiert oder mit einem Templatemechanismus individuell abgeändert werden, etwa zur Verwendung unterschiedlicher Parameter bei mathematischen Aufgaben.

Den Ablauf der Bearbeitung eines Items bzw. den währenddessen stattfindenden Informations- und Kontrollfluss zeigt die Abbildung 9. Prinzipiell wird dieser in drei Phasen unterteilt: Die schon beschriebene Instantiierung (entweder durch Verwendung eines fixen Inhalts oder durch das Templatesystem), die Interaktionen des Benutzers sowie des daraus resultierenden Feedbacks (response processing) und abschließend die Ermittelung des Ergbnisses der Bearbeitung (outcome processing).

3.1.3.2 Interaktionen und Response Processing

Um verschiedenartige Fragetypen abzubilden wird in diesem Standard auf eine Reihe sogenannter „interaction“s zurückgegriffen. Diese definieren die Art der Interaktionen eines Benutzers mit dem Item. Es gibt eine Vielzahl vorgegebener Interaktionsmöglichkeiten, hier eine Auswahl:

- Einfache Interaktionen
  - choiceInteraction: Auswahl aus vorgegebenen Alternativen
  - orderInteraction: Sortieren von Werten
  - associateInteraction: Zuordnung von je zwei Elemente aus verschiedenen Mengen
  - gapMatchInteraction: Auswahl eines Elementes zum Füllen einer Textlücke

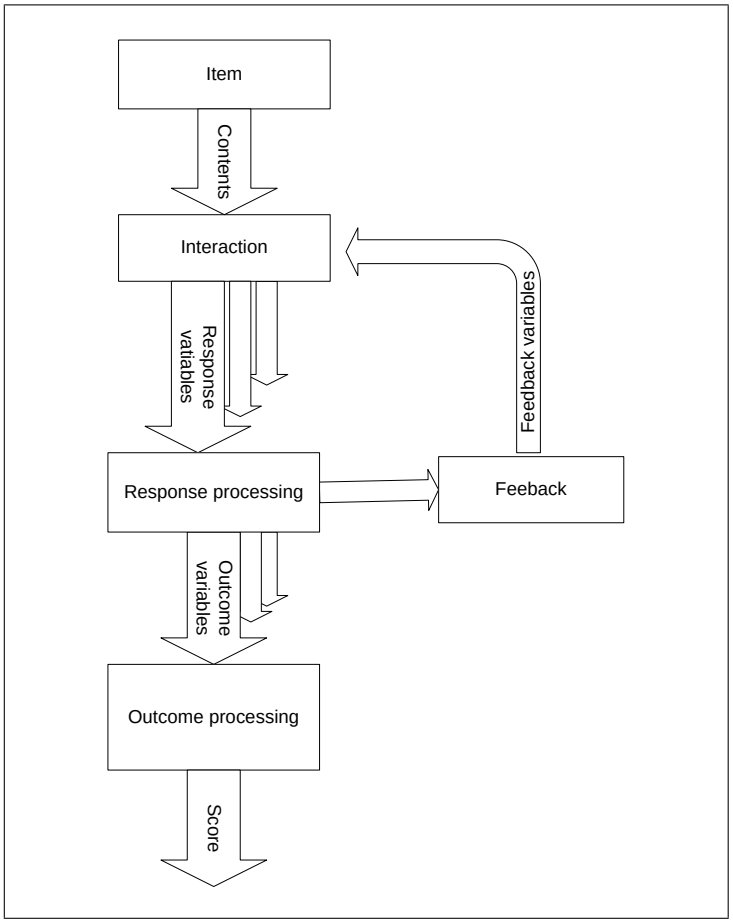


Abbildung 9: Informationsfluss innerhalb eines Items

- Textinteraktionen
  - textEntryInteraction: Eingabe von beliebigem Text
  - hottextEntryInteraction: Auswahl eines Teiltextes
- Graphische Interaktionen
  - hotspotInteraction: Auswahl von Punkten in einer Grafik
  - graphicOrderInteraction: Sortieren von Grafiken
  - positionObjectInteraction: Positionieren einer vorgegebenen Grafik in Relation zu einer anderen
- Sonstige Interaktionen
  - uploadInteraction: Hochladen einer Datei

- drawingInteraction: Bearbeitung einer vorgegebenen Grafik
- customInteraction: beliebige, nicht vom Standard vorgegebene Interaktion

Jede dieser *interactions* bildet das Ergebnis einer Benutzerinteraktion auf eine oder mehrere Antwortvariablen ab, deren Werte danach für die Bewertung des Items nach den in der „response processing“ festgelegten Regeln verwendet werden. Dazu gibt es eine entsprechende Regelsprache. Mit Hilfe einer vorgegebenen Menge von Operatoren, z.B. boolesche und arithmetische Operatoren, werden die Antwortvariablen transformiert und in sogenannte „outcomes“ gespeichert, ein weiterer Variablentyp. Diese können entweder für Feedbackinformationen für den Aufgabenbearbeiter oder für die Ergebnisspeicherung des Items verwendet werden.

3.1.3.3 Erweiterbarkeit

QTI beschreibt sehr umfassend Strukturen, die für Tests und Onlinefragebögen. Die Verwendung durch eine Vielzahl von Plattformen wie z.B. OLAT, Moodle, ILIAS und andere zeigt seine Bedeutung für das Thema E-Testing. Wie jedoch kann der Standard um neue Fragentypen erweitert werden? Für die geforderte Malaufgabe kann etwa die schon vorhandene *drawingInteraction* verwendet werden. Die Auswertung ließe sich natürlich nur manuell vornehmen. Dieser Aufgabentyp ist also mit Mitteln des Standards umsetzbar. Für die Anbindung von Autotool bzw. von Programmieraufgaben gibt es nun zwei Möglichkeiten:

1. **customInteraction** Eine customInteraction analog zur textEntryInteraction wäre zu entwickeln, welche jedoch zusätzlich den vom Benutzer als Lösung geschriebenen Code validiert und in eine Bewertungsvariable überführt. Damit würde aber ein wesentlicher Teil der Bewertung in einer der Benutzeroberfläche zuzuordnenden Komponente stattfinden, softwaretechnisch also sehr unelegant.

2. **customOperator** Es wäre ein Operator von customOperator abzuleiten, welcher eingesandten Quellcode übersetzt, ausführt und testet. Das Ergebnis dieses Operators wäre dann eine Angabe zur Korrektheit der Lösung. Leider scheint es in der Praxis noch keine Erfahrungen zur Verwendung dieses Konstrukts zu geben, selbst der Standard vermerkt dazu [IMS06]:

   > Implementors experimenting with this approach are encouraged to share information about their solutions to help determine the best way to achieve this type of processing.

Leider ergibt sich aus beiden Möglichkeiten ein wichtiger Nachteil: Durch die Verwendung dieser Konstrukte verliert man sofort die Eigenschaft der Austauschbarkeit von Testdaten zwischen verschiedenen den Standard implementierenden Systemen, da jede dieser Instanzen die entsprechenden Erweiterungen umsetzen müsste. Man wäre also an eine entsprechend angepasste *Delivery Engine*[18] gebunden.

3.2 Serviceorientierte Architekturen - Einige Definitionen

Im bisherigen Kapitel wurde wiederholt der Begriff „Service“ verwendet ohne eine allgemeine Definition zu geben. Dazu sagt das Referenzmodell der OASIS[19] : „...is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies...“ ([OAS07], S. 12). Ein Service ist zusammenfassend also ein beliebiges technisches Artefakt mit den folgenden Merkmalen:

- Ein Service bietet eine konkrete Dienstleistung über eine fest definierte Schnittstelle an (service contract).
- Die Schnittstelle hat eine detailliert dokumentierte Semantik.
- Die Verwendung eines Service liefert eine Information zurück oder führt zu einem genau definierten und erwünschten Seiteneffekt.

Diese Eigenschaften werden von einer großen Anzahl von sowohl Programmierparadigmen als auch konkreter Software erfüllt. Speziell ist die Frage der Anbindung eines Service nicht Bestandteil der Definition. Demnach kann schon eine einzelne in einer konkreten Programmiersprache implementierte Funktion mitsamt der dazugehörigen Dokumentation einen Service darstellen.

3.2.1 Abgrenzung zu objektorientierten Architekturen

Sowohl Objektorientierung als auch Serviceorientierung haben gemeinsame Prinzipien. Beide Ansätze verfolgen die Idee der Zerlegung von Problemlösungen in einzelne, unabhängige und wiederverwendbare Komponenten (Klassen bzw. Services). Dabei sind ihnen die folgenden Eigenschaften gemeinsam:

[18] Container zur Ausführung eines in QTI definierten Tests
[19] Organization for the Advancement of Structured Information Standards, www.oasis-open.org/

Abstraktion
: Der Verwendungszweck von Objekten bzw. deren Methoden sowie von Services ist durch eine Schnittstellendefinition fest vorgegeben, deren interne Struktur jedoch verborgen

Autonomie
: Die interne Logik eines Objektes bzw. eines Services ist gekapselt und nicht von außen sicht- oder manipulierbar.

Lose Kopplung
: Idealerweise haben Klassen bzw. Services nur über Schnittstellen Zugriff auf andere Klassen und Services. Ausschließlich auf diese Weise werden Services miteinander verknüpft. Es gibt keine feste Bindung untereinander, die Zusammenarbeit von Services kann jederzeit durch Änderungen an deren Konfigurationen geändert werden.

Wiederverwendbarkeit
: Beide Ansätze legen Wert auf die Wiederverwendung von existierendem Code. Diese stellt sich jedoch bei der Objektorientierung als schwierig heraus, als Beispiel sei dafür das Problem der „fragilen Basisklasse“[ES98] genannt.

Speziell für Anwendungen, die nicht lokal auf einem einzelnen System sondern verteilt auf ein Netzwerk laufen sollen sind objektorientierte Architekturen nur bedingt geeignet. Die fehlenden, serviceorientierte Architekturen jedoch eigenen, weitergehenden Eigenschaften sind:

Zustandslosigkeit
: Ein Service enthält idealerweise keinerlei internen Zustand, ein Service kann aber sehr wohl Daten bereitstellen.

Entdeckbarkeit
: Services werden durch entsprechende Spezifikationen beschrieben, die dafür gedacht sind, um mit Hilfe entsprechender Kataloge und Mechanismen von den interessierten Kunden gefunden werden zu können.

Zusammensetzbarkeit
: Services lassen sich zusammenfassen und koordinieren, so dass ihre Gesamtheit als zusammengesetzter Service zur Verfügung stehen kann.

Interoperabilität
: Services sind nicht auf eine gemeinsame Technologiegrundlage festgelegt. Eine entsprechende Kommunikationsstruktur vorausgesetzt sind Services prinzipiell technologieagnostisch.

3.2.2 Ausprägungen einer SOA

SOA als Architektur kann in einer Vielzahl von Formen umgesetzt werden. Im wesentlichen sind jedoch drei Formen verbreitet: Webservices, CORBA und lokale Anwendungen, die SOA per Nachrichtenaustausch umsetzen.

Die Spezifikationen zu CORBA und Webservices sind unterschiedlich, ähneln sich jedoch in der Art und Weise der Integration von Services: Eine Schnittstelle wird im Netzwerk veröffentlicht, die ausschließlich über ein bestimmtes Protokoll angesprochen werden kann. Eine Applikation, die als Service zur Verfügung gestellt werden soll muss eine solche Schnittstelle implementieren. Dieser Ansatz soll jedoch nicht verfolgt werden, da er verlangt, dass die Artefakte, die in dieser Arbeit integriert werden sollen, verändert werden.

Um diese Notwendigkeit und die damit verbundene Komplexität der Anpassung fremden Codes zu vermeiden bzw. die etwaige Unsinnigkeit der Einführung einer Netzanbindung von Applikationen, die sonst keine benötigen, soll eine lokale SOA entwickelt werden. Diese verlangt zwar auch die Kapselung von Anwendungen, die als Aufgabentyp zur Verfügung stehen sollen, jedoch ist der Aufwand der Umsetzung, und damit auch die Fehleranfälligkeit derselben, deutlich geringer.

3.3 Fazit und grundlegende Architekturentscheidung

Die in dieser Arbeit entworfene Architektur lässt sich dem Service „Assessment“ im IAF zuordnen. Durch die zu geringe Spezifikation der Interna dieses Services eignet sich diese Beschreibung jedoch nicht als Vorlage für eine Architektur, die diesen umsetzt. Die wesentlich ausführlichere Darstellung in FREMA bietet sich an dieser Stelle eher als Vorgabe an. Das Aufgabenframework des elatePortal, welches als Grundlage in dieser Arbeit genutzt wird, entspricht einer konkreten Implementierung der Ideen, die in den Services des *End-to-end Summative Assessment* aus FREMA spezifiziert werden. Die Funktionalität dieses Aufgabenframeworks erfüllt damit die Anforderungen an einen Assessmentservice in der Definition dieser beiden wichtigen Referenzmodelle. Auf diesen beiden Granularitätsebenen ist jedoch der eigentliche Ablauf eines Assessments nicht vorgegeben. An dieser Stelle wird jeweils die Verwendung von IMS QTI angenommen. Da dieser Standard in seiner Komplexität wegen unzureichenden Ressourcen nicht umgesetzt werden konnte, wurde er zumindest als semantische Vorlage verwendet, d.h. die Grundstruktur eines Assessments ist bis auf syntaktische Unterschiede der Aufgabendefinitionen äquivalent.

Gemeinsam ist den vorgestellten Modellen die Darstellung der Anwendungsdomänen mit Hilfe der Abstraktion von Funktionalitäten als Services. Die Vorzüge

einer Architektur nach diesem Prinzip sind im Umfeld von E-Learning eindeutig:

- Viele Funktionen sind verschiedenen Applikationen gemein und können als „Common services“ [YHDN05, S. 9] auch in fremden Anwendungsdomänen eingesetzt werden.
- Monolithische Anwendungen sind durch die wachsende Bedeutung von verteilten Systemen mehr und mehr obsolet. Ihre Erweiterbarkeit und Anpassbarkeit an die verschiedenen Umfelder, in denen sie eingesetzt werden sollen, sind nur eingeschränkt gegeben.
- Portale als Zugang zu - bzw. Bündelung von - bestehenden Applikationen und Funktionalitäten gewinnt immer mehr an Bedeutung. [BMMW04, S. 11]
- Objektorientierung ist als Modellierungswerkzeug nicht ausreichend für die Einbindung von beliebigen Aufgabentypen.

Die zu entwerfende Architektur soll also serviceorientiert sein. Damit stellt sich die Frage, welche technische Basis die Infrastruktur bereitstellen soll, so dass die in Kapitel 2.5 aufgezählten Anforderungen mit angemessenem Aufwand umgesetzt werden können. Die Verwendung von Webservices stellt zwar eine konzeptionell saubere Basis dar, die Menge der notwendigen Anpassungen an den so eingebundenen Services stellt jedoch eine wesentliche Hürde dar. Einfacher, und für den Einsatz von Prüfungen auch in nichtverteilten Szenarien geeigneter, ist die Verwendung einer lokalen SOA. Als wichtigster Standard hat sich dafür OSGi herauskristallisiert.

3.4 OSGi - SOA in a JVM

OSGi ist ein eingetragenes Markenzeichen und stand ursprünglich für „Open Service Gateway initiative“. Da „Gateway“ jedoch nicht mehr der aktuellen Ausrichtung entspricht wird es heute häufig mit „Open Specifications Group Initiative“ gleichgesetzt. Es handelt sich um einen Standard [All03], der von der OSGi Alliance vorangetrieben wird. Mit dem Java Standard Request JSR 291 [IBM06] wird angestrebt, OSGi als Bestandteil der Java-Laufzeitumgebung zu etablieren. Der Alliance gehören seit Mitte der 1990er zahlreiche namhafte Firmen an, unter anderem IBM, Motorola, Samsung, Siemens AG, Nokia und Oracle.

OSGi versteht sich selbst als „universal middleware“. Es stellt die Spezifikation für eine Plattform bereit, welche, nach den Prinzipien einer SOA, das dynamische Zusammensetzen einer Applikation aus einzelnen Modulen ermöglicht. Der Standard umfasst den Aufbau der Module, ihren Lifecycle, Versionierung, Standardservices für Querschnittsbelange wie z.B. Logging und Events, Sicherheitsaspekte, Management der Anwendungen etc. Damit ist es möglich, einzelne Dienste unabhängig voneinander zu entwickeln und auf einer großen Bandbreite von Plattformen von eingebetteten Systemen bis hin zu Serverfarmen einzusetzen, ohne Änderungen an der Codebasis ausführen zu müssen. Es wurde der Leitsatz einer „SOA in a JVM“ geprägt.

3.4.1 Adressierte Probleme

Die Spezifikationen von Java™bzw. der JVM sind nicht ausreichend für alle Anwendungsszenarien, die heute benötigt werden. Einige der größten Unzulänglichkeiten sind die folgenden:

3.4.1.1 Versionierung von Bibliotheken

Es gibt ein Vielzahl unterschiedlicher Lösungsansätze für das Problem der Beherrschung der Komplexität von Softwaresystemen. In der Sprache Java™wurde neben der Einteilung von Klassen in Pakete [GJSB05, S. 153ff], und damit in eigenständige Namensräume, für die physische Aufteilung einfache Archive auf Grundlage des Komprimierungsformates Zip gewählt. Jede Anwendung greift nun auf Klassen im Klassenpfad zurück. Ein *Classloader* löst dabei die jeweils durch Strings referenzierten Klassen auf, indem er diesen Klassenpfad durchsucht und die erste gefundene Klasse auswählt. Es gibt keinerlei Versionsinformationen in den Paketen bzw. Klassen. Werden also in einer Anwendung zwei unterschiedliche Versionen einer Klasse benötigt ist dieses Programm nicht lauffähig, da nur eine der beiden Versionen geladen werden kann.

3.4.1.2 Sicherheit innerhalb einer Anwendung

Die JVM hat eine Reihe von Sicherheitsmerkmalen eingebaut. Auf der niedrigsten Ebene verhindert die virtuelle Maschine fehleranfällige Konstrukte wie z.B. Zeiger, manuelle Speicherverwaltung und damit Fehlerquellen wie Pufferüberläufe. Darüber stellt die Sicherheitsarchitektur [LG03] eine Sandbox bereit, die es erlaubt, Code feingranulare Rechte zu vergeben. Damit lassen sich Zugriffe auf Ressourcen wie z.B. das Dateisystem regeln und die Echtheit von Code durch Zertifikate verifizieren. Alle Klassen, die jedoch einmal in eine VM geladen wurden, können prinzipiell frei miteinander interagieren. In einem denkbaren Szenario kann also eine bösartige Klasse beliebige Methoden einer laufenden Anwendung aufrufen, ohne dass diese Interaktionen eingeschränkt werden könnten.

3.4.1.3 Speicherverbrauch von simultanen Anwendungen

Jede Java™-Anwendung wird gewöhnlich in einer eigenen Instanz der JVM ausgeführt. Bibliotheken, die von mehreren Anwendungen genutzt werden, müssen also unabhängig voneinander jeweils vorhanden sein. Speziell auf Geräten mit beschränkten Ressourcen, etwa eingebettete Systeme, führt dies zu einem erheblichen Mehrbedarf an Speicherplatz. Auch die Kommunikation zwischen den einzelnen Anwendungen (IPC) ist nicht durch die Plattform spezifiziert.

3.4.1.4 Statische Struktur

Eine Java™-Anwendung ist im Allgemeinen vor dem Start in ihrer Struktur festgelegt. Es ist zwar möglich zur Laufzeit Klassen nachzuladen, jedoch ist ein Austausch von Klassendefinitionen bzw. ganzer Pakete nicht möglich. Speziell bei Anwendungen, die eine hohe Verfügbarkeit benötigen stellt der notwendige Neustart bei Updates ein Problem dar. Außerdem ist die Auflösung von Klassen- bzw. Paketnamen zu den entsprechenden Implementierungen durch den Classloader festgelegt. Es wird eine beliebige passende Klasse ausgewählt. Wenn also zwei eingebundene Bibliotheken dieselben Namensräume verwenden, kann zur falschen Klasse aufgelöst werden [GJSB05, S. 313ff].

3.4.2 Inhalt der OSGi-Spezifikation

OSGi besteht aus einer Reihe von aufeinander aufbauenden Schichten. Sie erweitern das durch Java™und die JVM vorgegebene Modell durch Konstrukte, die nicht nur Lösungen für die zuvor aufgeführten Probleme mitbringen, sondern

deren Funktionalität deutlich erweitern. Als Grundlage dieser Übersicht soll die Abbildung 10 dienen.

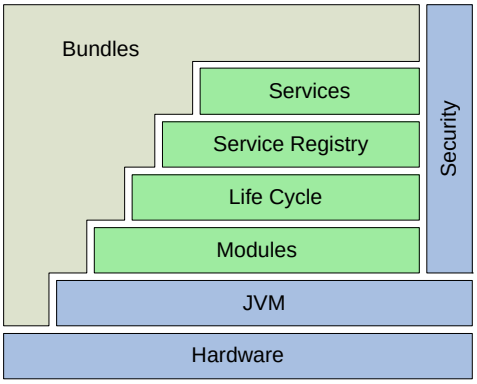


Abbildung 10: Bestandteile der OSGi-Spezifikation, nach [IBM05, S. 11]

3.4.2.1 *Execution environment*

Im Kern wird ein „execution environment“ spezifiziert. Dies umfasst die Klassen einer bestimmten Java-Version, z.B. *JRE-1.1* bis *JavaSE-1.6* für Desktop- und Serveranwendungen, *CDC* [Sun05] für mobile JVMs (J2ME) oder *OSGi/Minimum* für eine theoretische Minimalumgebung, in der OSGi lauffähig ist.

3.4.2.2 *Module layer*

Diese Schicht definiert Bundles. Ein Bundle ist ein Modul bestehend aus Javaklassen und benötigten Ressourcen. Sie liegen in der Regel als Jar-Archiv vor. Die enthaltene Manifest.mf-Datei wird um eine Anzahl von Metadaten angereichert, die explizit die Abhängigkeiten und angebotenen Pakete aufzählen. Das Sichtbarkeitskonzept von Klassen und Methoden in Java™wird damit auf Pakete erweitert. Das folgende Beispiel zeigt den Ausschnitt aus der Manifest.mf eines Bundles.

```
Bundle-SymbolicName: bundle1
Bundle-Version: 1.0.1
Export-Package: pack1.*, pack2.subpack1
PrivatePackage: privatePackage1.*
Import-Package: importPackage1.*;version=[1.0.0,2.0.0);resolution:=mandatory,
                importPackage2.*;resolution:=optional
```

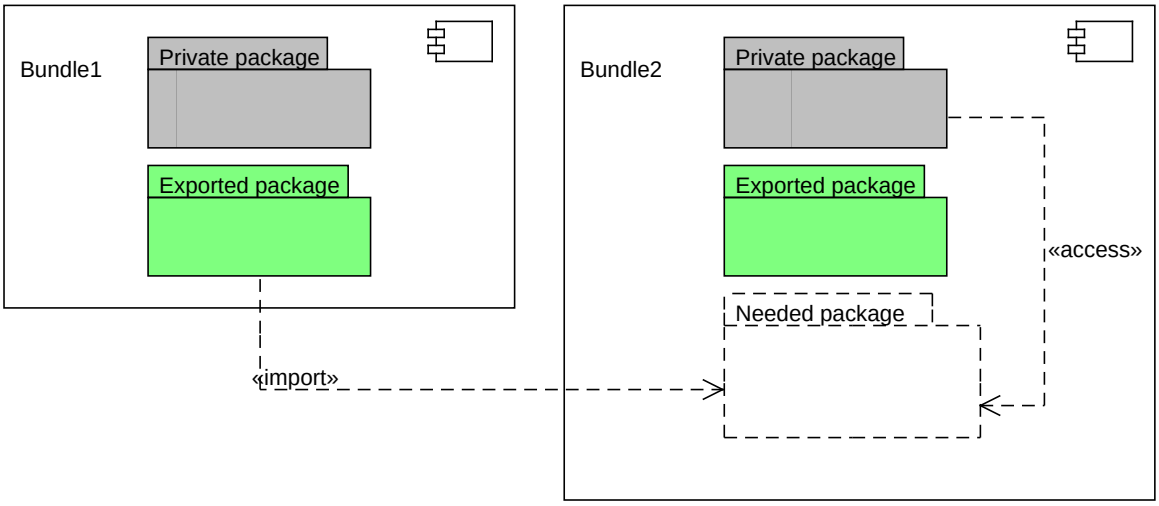


Abbildung 11: Beispiel für den Zugriff auf Pakete

```
Require-Bundle: meinBundle
Bundle-ClassPath: ., somePrivateJar.jar
Bundle-Activator: privatePackage1.Activator
```

Private-Package listet alle Pakete auf, die nur innerhalb des Bundles sichtbar sein sollen. Andere Bundles können nicht darauf zugreifen. *Export-Package* stellt Pakete im Framework bereit. Jedes Bundle hat einen eindeutigen Namen und eine Versionsnummer, die auf die exportierten Pakete angewendet wird. Das Importieren von Paketen bietet eine Reihe von zusätzlichen Optionen an. Im obigen Beispiel wird *importPackage1* und alle enthaltenen Pakete mit einer Version im angegebenen Intervall benötigt. Dieser Import ist zwingend (*mandatory*). Kann dieser Import nicht aufgelöst werden, sind die Voraussetzungen für die Benutzung dieses Bundles nicht gegeben und es kann nicht verwendet werden. *ImportPackage2* und dessen Unterpakete sind nicht zwingend notwendig, außerdem ist jede beliebige Version des Paketes zulässig. Dieser Mechanismus ermöglicht also die zeitgleiche Verwendung von verschiedenen Versionen ein und desselben Paketes innerhalb einer Anwendung. Die Austauschbarkeit von Paketen kann jedoch auch zu Problemen führen: wenn ein Paket nur dem Namen nach aufgelöst wird, jedoch ein ganz konkretes Bundle verwendet werden soll. Für diesen Fall kann eine direkte Abhängigkeit von einem Bundle mit *Require-Bundle* angegeben werden. Außerdem listet der Eintrag *Bundle-ClassPath* private Bibliotheken auf. Im Unterschied zu reinen Jar-Archiven kann ein Bundle andere Archive enthalten, die nur von diesem Bundle gesehen und genutzt werden können. Auf andere Einträge (z.B. Bundle-NativeCode für OS-spezifische Bibliotheken, Bundle-Activator

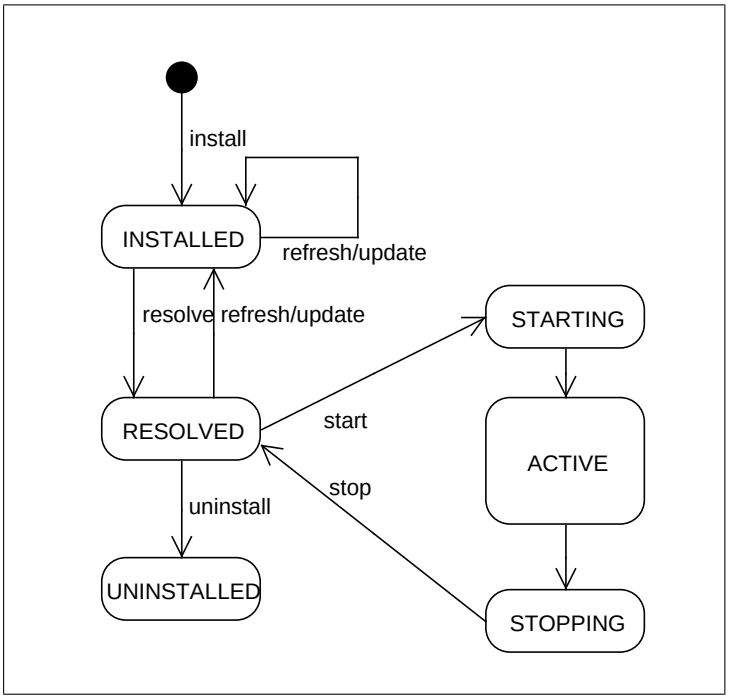


Abbildung 12: Lebenszyklus von Bundles

als Callback für den Lifecycle eines Bundles) soll an dieser Stelle vorerst nicht genauer eingegangen werden.

3.4.2.3 Life cycle layer

Das Vorhandensein von Bundles in einem OSGi-Framework ist hochgradig dynamisch. Zu jedem beliebigen Zeitpunkt können Bundles hinzugefügt oder entfernt werden. Aus diesem Grund gibt es einen festgelegten Lebenszyklus mit den entsprechenden Aktionen. Die Abbildung 12 zeigt dessen Zustandsdiagramm. Wenn ein Bundle geladen wird, befindet es sich im Zustand *INSTALLED*. Danach wird versucht, alle benötigten Abhängigkeiten aufzulösen. Wenn das gelingt ändert sich der Zustand in *RESOLVED*. Damit ist das Bundle ausführbar, alle vorausgesetzten Pakete sind vorhanden. Ein gestartetes Bundle bleibt nun solange *ACTIVE* bis es entweder neu geladen wird, z.B. weil ein Update erfolgen soll, oder es entfernt werden soll. Es ist jedoch zu beachten, dass Bundles, deren Pakete noch aktiv von anderen genutzt werden, zwar formal aus der Laufzeitumgebung entfernt werden, neuen Bundles also nicht zur Verfügung stehen, jedoch weiterhin von allen bisherigen importierenden Bundles weiter genutzt werden kann.

Der Vorgang des Auflösens von Abhängigkeiten des Bundles ist ebenfalls ge-

nau festgelegt. Der verwendete Algorithmus enthält neben den schon erwähnten Restriktionen auch die notwendige Logik um sicherzustellen, dass Bundles, die dieselben Pakete importieren und miteinander interagieren, auch die identischen Pakete zugeordnet bekommen. Andernfalls wäre eine Kommunikation zwischen beiden nicht möglich, zur Laufzeit würden sich dieses Problem jeweils durch *ClassCastExceptions* äußern.

3.4.2.4 Service registry

Die *Service registry* bietet sich als Konstrukt dar, welches es erlaubt, Implementierungen von Java-Interfaces als Services anzubieten und zu nutzen. Diese Registry existiert nur während der Laufzeit, ist also nicht persistent. Jedes Bundle kann eine spezielle *Activator*-Klasse angeben, welche Methoden enthält, die zu den einzelnen Zeitpunkten des Lifecycles aufgerufen werden:

```
package org.osgi.framework;

public interface BundleActivator {

   public abstract void start(BundleContext bundlecontext)
      throws Exception;

   public abstract void stop(BundleContext bundlecontext)
      throws Exception;

}
```

BundleContext bietet dabei Zugriff auf andere Bundles und Services. Um also anderen Bundles einen Service zur Verfügung stellen zu können muss dieser mit Hilfe der Methoden des *BundleContexts* im Framework registriert werden. Dazu wird eine Instanz eines definierten Interfaces mitsamt notwendiger Metainformationen als Service registriert. Auf analoge Art und Weise kann auf fremde Services zugegriffen werden. Beispielhaft zeigt das Sequenzdiagramm in Abbildung 13 den Ablauf bei der Registrierung eines Services gefolgt vom Aufruf desselben. Wegen der dynamischen Natur von OSGi, die es Bundles erlaubt, zu jedem beliebigen Zeitpunkt Services anzubieten bzw. zu entfernen, wird die Klasse *ServiceTracker* spezifiziert. Dieser informiert das Bundle über neu verfügbare, geänderte und entfernte Services, die seinen vorgegebenen Parametern entsprechen. Damit wird verhindert, dass jedes Bundle für jeden Service eigene Listener bzw. Events anbietet. Die mangelhafte Skalierbarkeit eines solchen Ansatzes wird ganz klar in [All04] dargestellt.

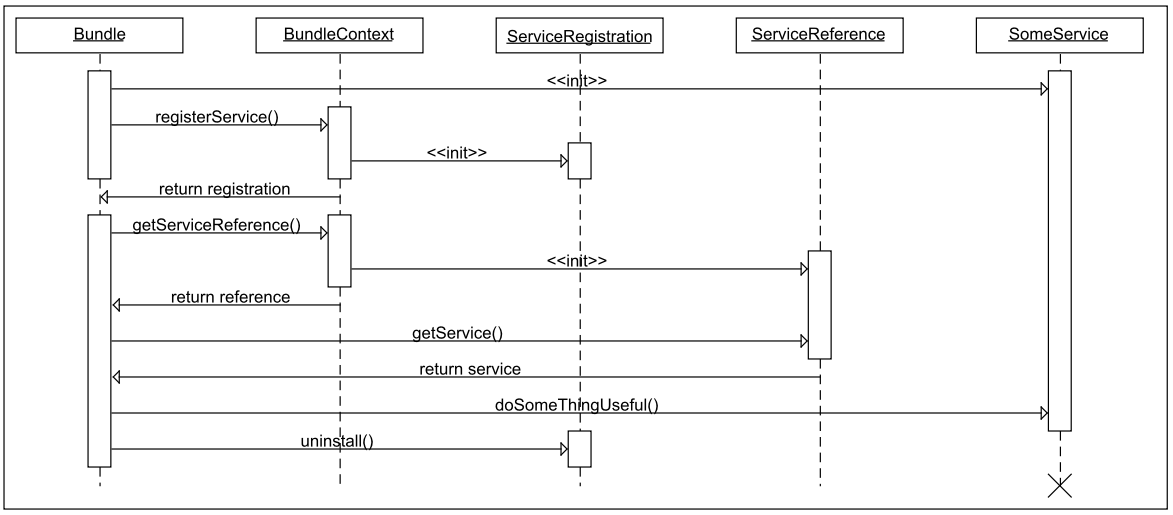


Abbildung 13: Ablauf von Serviceregistrierung und -verwendung

Aus dieser Art und Weise der Zuordnung von Serviceanbietern und Servicekonsumenten ergibt sich jedoch das schwerwiegende Problem der Abhängigkeiten zwischen Services: Bundles hängen voneinander ab und können nur funktionieren, wenn die entsprechend benötigten Services bei ihrer eigenen Aktivierung schon vorhanden sind. Außerdem führt die Notwendigkeit, dass die entsprechenden Serviceanbieter im Framework vorhanden sein müssen unter Umständen zu einem hohen Ressourcenverbrauch, der bei Nichtverwendung dieser Services nicht gerechtfertigt wäre. Aus diesen beiden Gründen wurde in der Version R4 des OSGi-Standards die *„Declarative services“* eingeführt. Sie erlauben die Beschreibung von Services als XML-Datei neben der Manifest.mf im Jar-Archiv. OSGi bietet diese Services anderen Bundles stellvertretend an, instantiiert sie jedoch erst, wenn sie tatsächlich verwendet werden. Das Ergebnis ist also ein, je nach Anwendung, reduzierter Speicherverbrauch sowie eine kürzere Anlaufzeit der Applikation. Außerdem bietet dieser Ansatz die Möglichkeit innerhalb des Bundles Abhängigkeit von der OSGi-API zu vermeiden.

3.4.2.5 Security layer

OSGi erweitert das Sicherheitskonzept der JVM auf zwei Arten. Einmal ermöglicht die genaue Spezifikation von exportierten und privaten Paketen die Absicherung vor Zugriffen auf private, bundleinterne Klassen. Damit wurden die in Java™ vorhandenen Sichtbarkeitsregeln für Methoden, Attribute und Klassen auf Pakete erweitert. Außerdem wird die Sicherheitsarchitektur von Java (siehe Einführung

in [SUN07]) um Bundle-, Framework-, Admin-, Package- und Servicepermissions ergänzt. Diese *Permissions* erlauben eine feingranulare Vergabe von Rechten für den Zugriff auf Ressourcen innerhalb der Bundles, auf Services und auf die Administratorfunktionen des Frameworks. Um diese *Permissions* nutzen zu können muss die JVM mit einem *SecurityManager* gestartet werden. Dieser ist sonst standardmäßig nur für Applets aktiv und erlaubt bzw. verbietet den Zugriff auf Ressourcen mit Hilfe der gesetzten *Permissions*.

3.4.2.6 Standard services

Auf diesen bisher dargestellten Schichten spezifiziert OSGi eine Reihe von Services, die in jeder Implementierung des Frameworks vorhanden sein sollten, da sie entweder wesentliche Funktionen von OSGi kapseln oder Querschnittsbelange enthalten, die in jeder Applikation benötigt werden. Eine Auswahl wesentlicher, wichtiger oder nützlicher Standardservices sind:

Configuration Admin
: Verwaltet Parameter und Konfigurationsdaten einzelner Bundles. Dieser Service erlaubt die Umkonfigurierung während der Laufzeit (z.B. setzen von Ports, Dateipfaden, URLs etc.)

Package Admin
: Kapselt den Algorithmus der Zuordnung von exportierten und importierten Paketen. Er kann genutzt werden um sich entweder über diese Zuordnung zu informieren oder sie neu anzustoßen.

Start Level
: Regelt die Reihenfolge des Starts der Bundles, vergleichbar den verschiedenen Startlevels in Linux.

Permission Admin
: Legt Rechte im Framework fest.

Declarative Services
: Entkoppelt ein Bundle vollständig von Abhängigkeiten von OSGi, indem Services nicht per Code in einer *Activator*-Klasse im Framework bereitgestellt werden müssen, sondern brauchen nur um eine XML-Beschreibung ergänzt werden.

Service Tracker
: Erlaubt die Vermeidung von servicespezifischen Listenern und informiert über die Statusänderungen von Services.

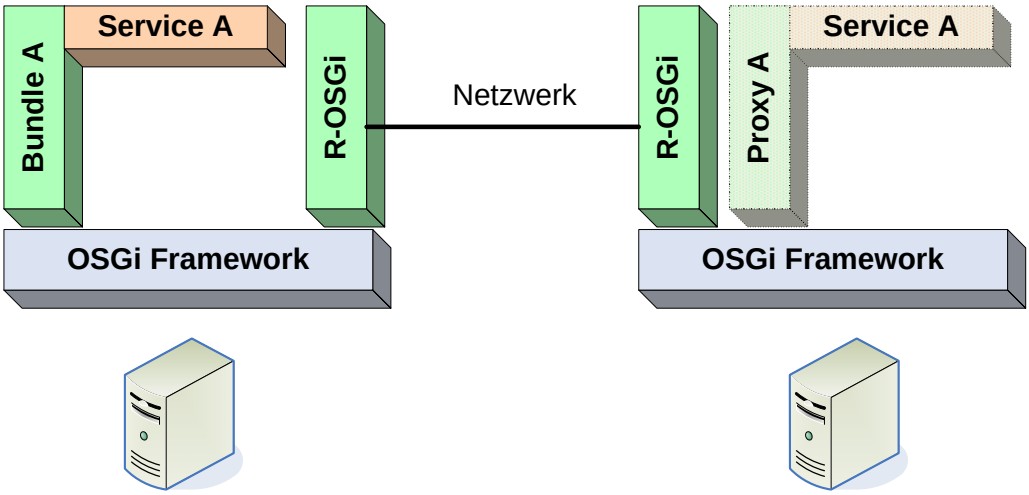


Abbildung 14: R-OSGi - Proxies für verteilte Services

Http Service

Dieser Service erlaubt die Zusammenstellung von Servlets und per Web erreichbare Ressourcen zur Laufzeit, kann also genutzt werden, um Applikationen über eine Webschnittstelle erreichbar zu machen.

3.4.3 Verteilte Services

Seit Veröffentlichung der Revision 3 im März 2003 kamen Fragen zur Verteilung von OSGi bzw. die darin verfügbaren Services auf mehrere Netzwerkknoten. Die Empfehlung lautete UPnP[20] für das *Service Discovery* mit Jini [21] als Bibliothek für die Realisierung von *Remote Services*. Diese Empfehlung wurde jedoch nicht aufgegriffen, es fehlt also ein im Standard verankertes Modell für verteilte Services. Ein an der ETH Zürich [22] entwickeltes Projekt greift genau diesen Punkt auf. Es bietet unter dem Namen „R-OSGi“ ein Bundle an, welches Services, die auf verschiedenen laufenden OSGi-Frameworks verteilt sind, transparent miteinander kommunizieren lässt. Ein *RemoteOSGiService* bietet dazu die Möglichkeit, Services als *remote* zu markieren. Diese können dann in anderen Laufzeitumgebungen als augenscheinlich lokale Services genutzt werden. Die Wahl des Übertragungsweges ist dabei flexibel wählbar. Es stehen eine Reihe von Implementierungen zur Verfügung, neben einem proprietären Format z.B. http oder Bluetooth. Benchmarks haben gezeigt, dass die Nutzung von verteilten Services mit R-OSGi in

[20] http://www.upnp.org/
[21] http://www.jini.org
[22] http://www.iks.inf.ethz.ch/

vielen Fällen sogar performanter sein kann als eine manuelle Implementierung mit Hilfe von RMI [RAR07, S.14ff]. Im Vergleich zu UPnP soll diese Lösung in der aktuellen Version 1.0.0 nach Angabe der Autoren sogar um den Faktor 280 schneller sein. Orthogonal dazu steht die Frage, wie diese verteilten Services gefunden werden können. R-OSGi bietet dafür zwei Möglichkeiten: Auf der Basis von Bluetooth, speziell für eingebettete Geräte, sowie mit Hilfe von SLP v2 [GPVD99].

Die Verfügbarkeit eines Services, ob lokal oder remote, ist für Nutzer desselben transparent. Dies wird erreicht indem R-OSGi für jeden Service, der verteilt werden soll ein Proxy erstellt wird (siehe Abb. 14), der das Serviceinterface implementiert und dessen Aufrufe an den jeweiligen realen Service delegiert. Dieser Mechanismus erlaubt die Verwendung der vom OSGi-Standard spezifizierten Konstrukte, wie z.B. des `ServiceTrackers`, zur Verteilung von Services. Es ist also keine Erweiterung der eigentlichen Plattform notwendig.

3.5 Architektur des Aufgabenframeworks

Das Aufgabenframework besteht im Wesentlichen aus der Spezifikation des Datenformats von Tests in Form von XML-Schemas sowie einer API mitsamt zugehöriger Referenzimplementierung. Diese sollen in diesem Kapitel vorgestellt und mit der Spezifikation von QTI verglichen werden. Die Komplexität der QTI-Spezifikation erlaubt viele Freiheiten bei der Definition von Testbeschreibungen und -abläufen. Erreicht werden diese durch die Möglichkeit, nicht nur Aufgabendefinitionen zu beschreiben sondern auch komplexe Ablauf- und Renderinformationen mit Hilfe einer eigenen XML-basierten Programmiersprache genau beschreiben zu können. Dies bedeutet jedoch auch einen hohen Einarbeitungsaufwand für jeden Testautor. Der dem elatePortal zugrunde liegende Geschäftsprozess einer universitären Prüfung erfordert nun einen zwar fest definierten, jedoch nur eingeschränkt variablen Testablauf. Aus diesen beiden Gründen wurde die Architektur des Aufgabenframeworks nicht nach den Maßgaben von IMS-QTI entwickelt, jedoch wurden einige notwendige Konzepte übernommen. Insbesondere wurde auf die zwar konzeptionell interessante jedoch in der angestrebten Domäne zu komplexe Möglichkeit der freien Programmierbarkeit von Testabläufen verzichtet. Stattdessen gibt es einen parametrisierbaren Testablauf.

3.5.1 Datenformat für Komplexaufgaben

Das Datenformat des Aufgabenframeworks ist zur Zeit nur für Komplexaufgaben definiert. Es liegt genau wie QTI als XML-Schema vor und beschreibt in seiner Gesamtheit einen Pool von Teilaufgaben mit den dazugehörigen Darstellungs-, Bewertungs- und Korrekturvorschriften. Es ist jedoch zugeschnitten auf die speziellen Bedingungen von universitären Kompetenzmessungen am PC mit einem fixen Satz von Aufgabentypen, bietet also einen auf diesen Anwendungsfall zugeschnittenen Funktionsumfang. Speziell findet sich schon hier die später angestrebte Darstellung eines Tests mit Hilfe der Papiermetapher. Die Studenten, die mit Elate Prüfungen absolvieren sollen nicht durch ungewohnte Navigationsstrukturen von dem Inhalt der Prüfung abgelenkt werden, sondern durch eine freie Navigation durch die einzelnen „Seiten“ in gewohnter Art und Weise die Aufgaben abarbeiten können. Deshalb fehlen die in QTI gebotenen Möglichkeiten der unterschiedlichen Navigationsarten wie z.B. linear bzw. durch frei wählbare Regeln festlegbare Abhängigkeiten zwischen Aufgaben (adaptiv). Im groben Vergleich ergeben sich zwischen den jeweiligen Elementen des Datenformats Äquivalenzen wie in Tabelle 2 dargestellt. Diese Begriffe sind jedoch keine reinen syntaktische Unterschiede bei komplett identischer Semantik, es existieren wesentliche Unterschiede. Eine Komplexaufgabe (`complexTaskDef`) besteht aus einer testweiten Konfiguration mit den wesentlichen Parametern Bearbeitungszeit (`time`), Anzahl der Be-

QTI	Aufgabenframework
test	complexTaskDef
testPart	category
assessmentSection	taskBlock
item	subTaskDef

Tabelle 2: Begriffsäquivalenzen zwischen Aufgabenframework und QTI

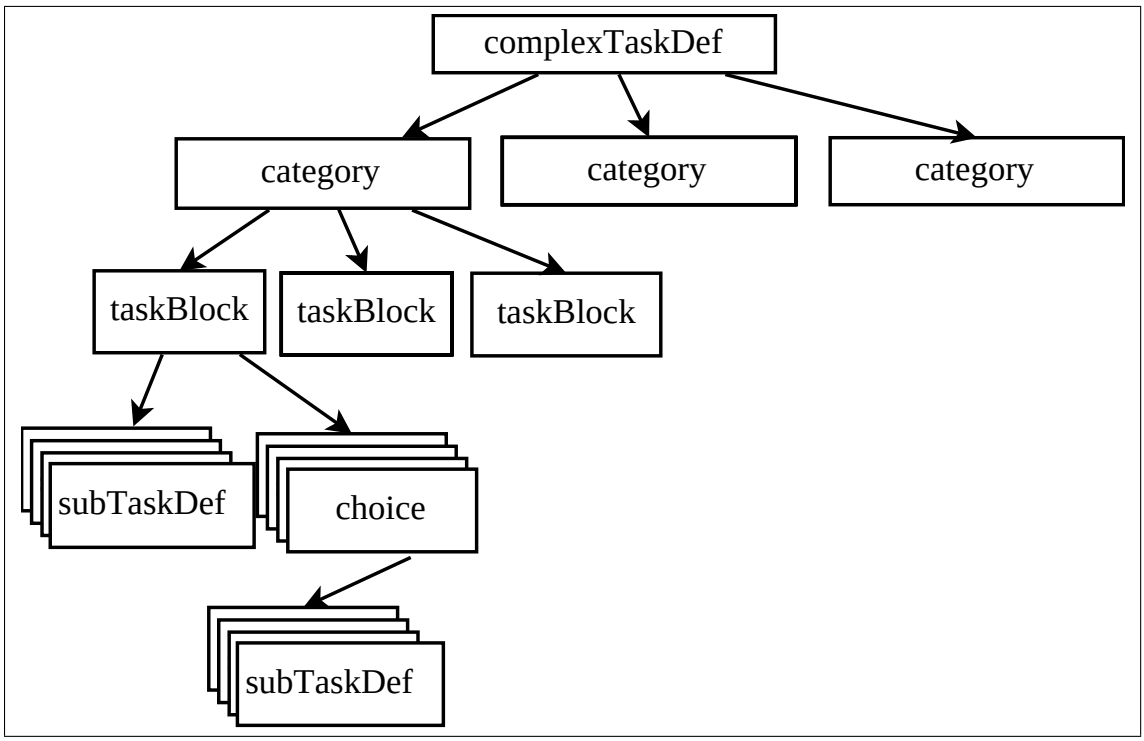


Abbildung 15: ComplexTaskDef - Struktur einer Aufgabendefinition

arbeitungsversuche (`tries`), Anzahl der Aufgaben pro Seite (`tasksPerPage`) und dem Korrekturmodus (correctionMode) sowie einer Liste von Kategorien, die die eigentlichen Aufgabendefinitionen, zusammengefasst in Aufgabenblöcken, enthalten. Diese Konfigurationsparameter finden sich in QTI an anderen Stellen, z.B. sowohl in `assessmentPart` als auch in den einzelnen Items. Eine `category` im Aufgabenframework entspricht einem `testPart` bzw. einer `assessmentSection`. Sie wird auch in der Darstellung am PC als erkennbare Kategorie dargestellt, dient also nicht wie bei QTI wahlweise als rein organisatorische Einheit. Eine Übersicht über die Struktur zeigt die Abbildung 15. Der wesentliche Unterschied zwischen einer Testbeschreibung im QTI-Format und einer Aufgabendefinition als `Komplexaufgabe` im elatePortal besteht in der stark auf den Anwendungsfall „Prüfung“ ausgerichteten Struktur. QTI bietet große Freiheiten bei der Art und Weise der Auswahl von anzuzeigenden Aufgaben, der Präsentation eines jeden

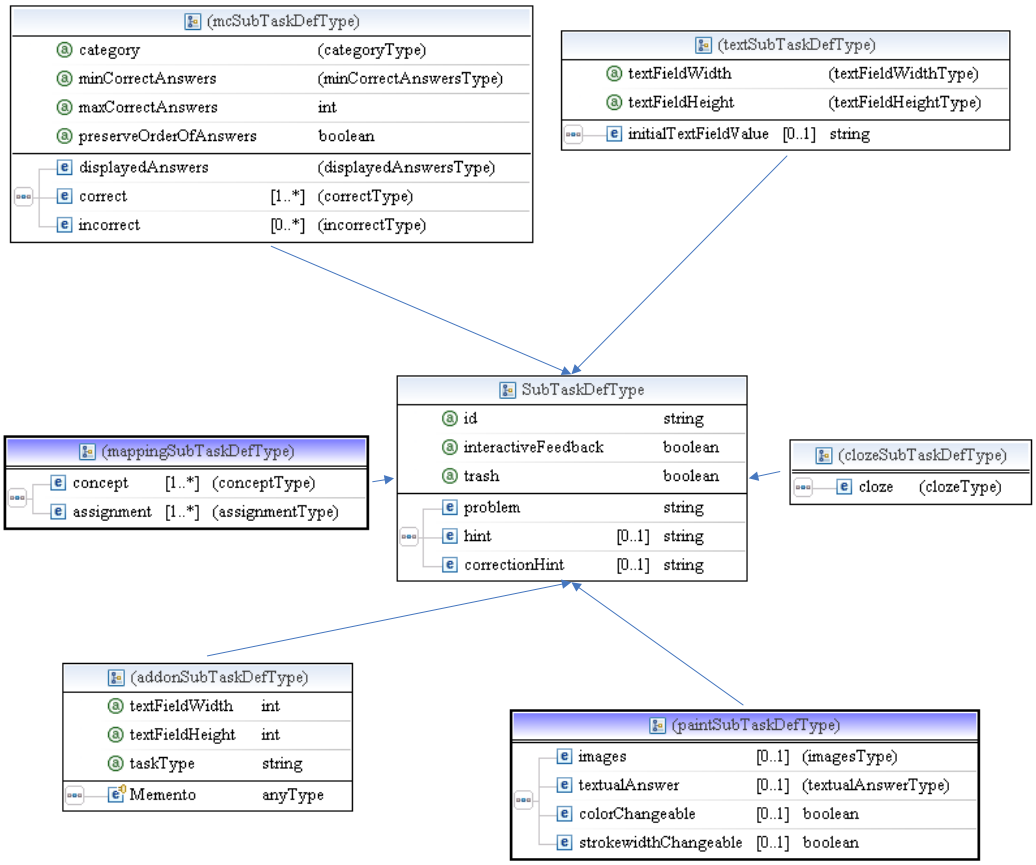
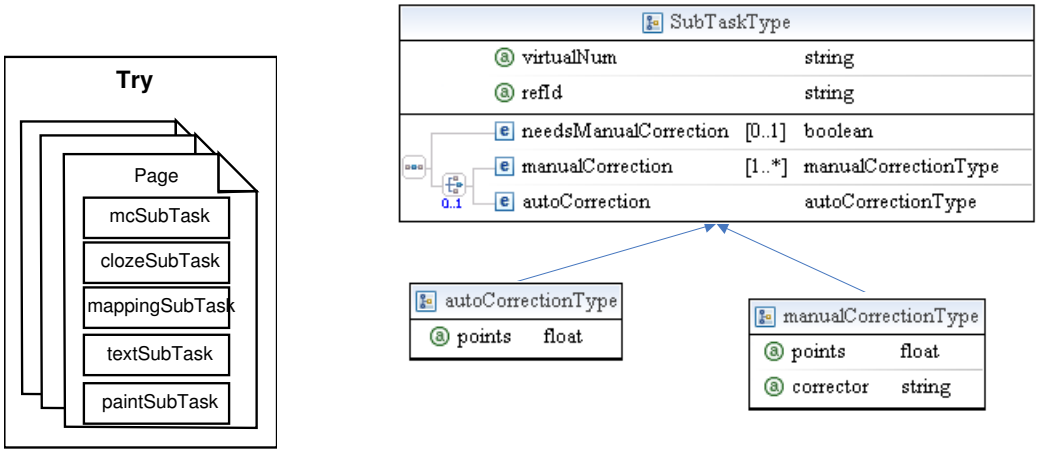


Abbildung 16: `SubTaskDef` - Übersicht über die vorhandenen Aufgabentypen

Items und der Bewertung mit Hilfe von explizit ausformulierten Anweisungen in XML. Dies geht jedoch mit einem stark erhöhten Aufwand für die Testautoren einher. Im elatePortal finden sich die entsprechenden Regeln zur Auswahl von Aufgaben und die Bewertung im Quelltext, nicht in der Testbeschreibung. Damit ergeben sich zwar vergleichsweise eingeschränkte Nutzungsmöglichkeiten, die geringere Komplexität der Implementierung und die daraus resultierende höhere Wahrscheinlichkeit der Korrektheit ist jedoch speziell für den Aspekt Sicherheit einer Prüfung wichtiger.

Der inhaltlich wichtigste Teil einer Komplexaufgabe sind die Teilaufgaben (SubTaskDef), aus denen sie besteht. Es gibt neben der abstrakten Klasse `SubTaskDefType` vier Aufgabentypen (siehe Abb. 16):

- `mcSubTaskDef` (MC-Aufgabe)



(a) Übersicht (b) Teilaufgabe

Abbildung 17: ComplexTaskHandling - Struktur einer bearbeiteten Komplexaufgabe

- `clozeSubTaskDef` (Lückentextaufgabe)
- `mappingSubTaskDef` (Zuordnungsaufgabe)
- `textSubTaskDef` (Freitextaufgabe)

3.5.2 Datenformat für Bearbeitungsversuche von Komplexaufgaben

Analog zu QTI existiert auch ein Schema für die Beschreibung von Ergebnissen, besonders für die Beschreibung von abgegebenen Antworten. Ein solches `ConmplexTaskHandling` besteht aus einer Liste von `Try`. Jeder `Try` repräsentiert einen eigenständigen Bearbeitungsversuch eines Prüflings, im Falle einer Prüfung existiert also im Normalfall nur eine Instanz. Er besteht aus einer Liste von Seiten (`Page`) mit den Antworten auf die präsentierten Aufgabenstellungen (`SubTasklet`). Analog zu den `SubTaskDefs` existieren individuelle `SubTasklets`, also `mcSubTasklet`, `texSubTasklet` etc. Durch eine jeweils eindeutige ID können diese den `SubTaskDefs` zugeordnet werden. Sie enthalten neben den gegebenen Antworten auch Korrekturinformationen, d.h. Korrekturstand, vergebene Punkte durch automatische bzw. eine manuelle Korrektur. Weitergehende Informationen, wie z.B. Namen des Korrektors oder Anmerkungen der Einsichtnahme sind nicht Bestandteil dieses Schemas, diese sind dem Hostsystem, in welches das Aufgabenframework eingebettet ist zugeordnet.

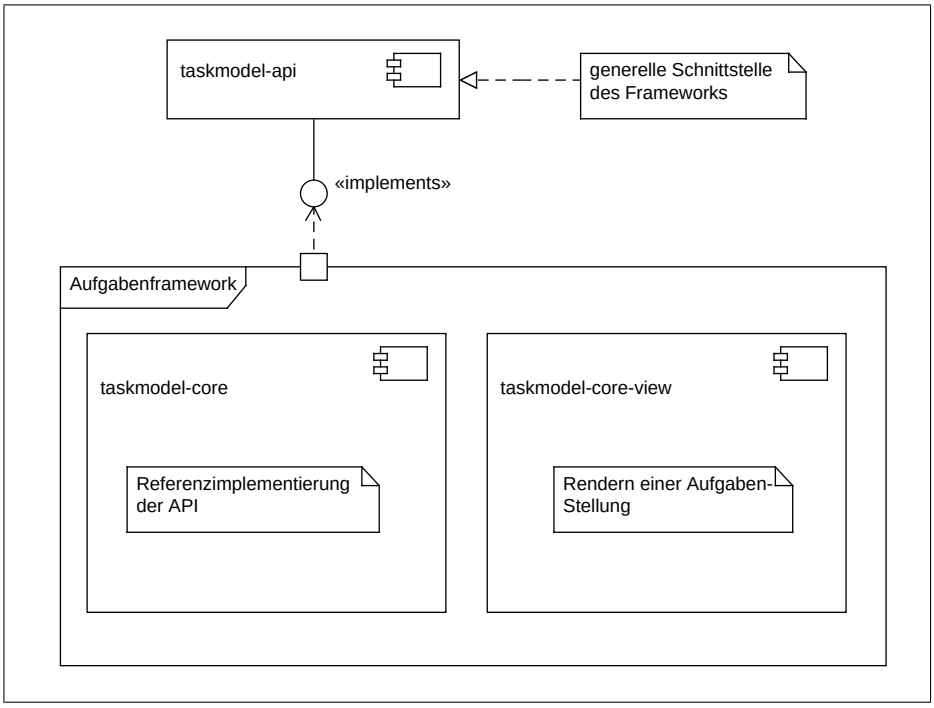


Abbildung 18: Vereinfachte Gesamtübersicht des Aufgabenframeworks

3.5.3 Erweiterung des Schemas

Für die Erweiterung des Schemas um neue Aufgabentypen boten sich zwei Möglichkeiten an: Ergänzung einer neuen Unterklasse von `SubTaskDefType` für jeden neuen Typen oder die Erstellung eines Platzhalters für beliebige Aufgabentypen, einen `addonSubTaskDef`. Die erste Variante fand für die Einbindung eines Aufgabentyps für graphische Interaktionen Anwendung, den `paintSubTaskDef`. Wesentlich flexibler ist jedoch der generische Typ `addonSubTaskDef`. Er bietet die Möglichkeit, beliebige Inhalte als Aufgabenbeschreibung zuzulassen, indem diese aufgabentypspezifischen Beschreibungen ohne Einschränkung der internen Struktur in einem XML-Knoten `Memento` gespeichert wird. Zugrunde liegt dem das gleichnamige Designpattern `Memento` [GHJV94]. Es beschreibt einen Mechanismus zur Speicherung und Wiederherstellung des internen Zustandes eines Objekts als äußerlich strukturloses Datum. Ein neuer Aufgabentyp hat damit die Möglichkeit beliebige Inhalte als XML spezifizieren zu können ohne dessen Struktur zuvor expliziert spezifizieren zu müssen..

3.5.4 API und Referenzimplementierung des Aufgabenframeworks

In diesem Teilkapitel soll die Architektur des bestehenden Aufgabenframework kurz dargestellt werden. Im Wesentlichen existieren drei Teilprojekte mit jeweils genau spezifiziertem Umfang:

taskmodel-api
: Eine Sammlung von Schnittstellen, abstrakten Klassen und Aufzählungen, die das Framework und seine Funktionalität beschreiben

taskmodel-core
: Referenzimplementierung der API

taskmodel-core-view
: Webapplikation zum Darstellen und Korrigieren von Komplexaufgaben, die ausschließlich über die API angesprochen wird

Entstanden sind große Teile der Architektur nach den Entwurfsrichtlinien der Softwareproduktlinienentwicklung [Ber07b], genauer nach einem angepassten Vorgehensmodell des SPL [PBL05]. Die aus diesem Prozess entstandenen Testartefakte erlauben eine große Flexibilität bezüglich der Einbettung dieses Frameworks in beliebige Hostsysteme. Abbildung 24 im Anhang zeigt ein Kompositionsdiagramm, welches die Anbindung durch eine `TaskFactory`-Komponente zeigt. Für eine Einbettung des Frameworks ist nur dieses Interface zu implementieren. Es enthält alle Persistenzfunktionen, die Aufgabendefinitionen und Benutzerinformationen bereitstellen sowie die gegebenen Antworten speichert. Im elatePortal wird das Framework sowohl im eigentlichen Portal für Übungsaufgaben eingebunden als auch im `examServer`, einem eigenständigen Prüfungsserver mit besonders hohen Sicherheitsstandards für die Durchführung von Prüfungen.

Die API stellt Interfaces bereit, die einen Prüfungsablauf modellieren. Im Wesentlichen besteht dieser aus den folgenden Klassen:

TaskDef
: Diese beschreiben Aufgabenstellungen. Es sind momentan zwei konkrete Unterklassen definiert, `TaskDef_Upload` für Aufgaben, deren Lösung als Dokument vom Studenten hochgeladen wird und `TaskDef_Complex` für eine zusammengesetzte Aufgabe, z.B. eine Prüfung. Der Aufbau von `TaskDef_Complex` wird im Klassendiagramm 19 dargestellt. Zu erkennen ist die Abbildung der Konstrukte des Aufgabenschemas auf entsprechende Interfaces.

Lebenszyklus	
Bearbeitung	INITIALIZED INPROGRESS SOLVED
Korrektur	CORRECTING CORRECTED
Einsichtnahme	ANNOTATED ANNOTATION_ACKNOWLEDGED

Tabelle 3: Zustände von Tasklets

Tasklet
: Ein `Tasklet` ist eine Aufgabe in Ausführung. Es existiert für jeden Testteilnehmer (`UserInfo`)und jede Aufgabe (aka Instanz eines `TaskDef`) genau ein Tasklet. Ein Tasklet enthält die Antworten des Testteilnehmers und die Korrekturinformationen.

TaskFactory
: Diese Klasse erstellt auf Anfrage Instanzen aller wichtigen Klassen des Frameworks. Sie ist der Integrationspunkt für eine Einbindung des Aufgabenframeworks in andere Systeme.

TaskletContainer
: Der Lebenszyklus eines `Tasklets` wird hier gesteuert.

Zugriff auf `TaskDefs` und `Tasklets` ermöglicht ein `TaskManager`. Zu einem `Tasklet` existiert eine Instanz von `TaskletCorrection`, diese kapselt die Korrekturinformationen: automatisch bzw. manuell vergebene Punkte, Namen der Korrektoren, Anmerkungen der Testteilnehmer während einer Einsichtnahme sowie der Korrektoren. Das Klassendiagramm in Abb. 23 im Anhang zeigt den Zusammenhang zwischen den wichtigsten Klassen.

Analog zur Abbildung der Konstrukte aus dem Beschreibungsschema für Komplexaufgaben (siehe Seite 53) gibt es für Klassen für das `TaskHandling`. Jedes `Tasklet` enthält `SubTasklets`. Diese stellen die einzelnen Teilaufgaben in Ausführung dar. `Tasklets` durchlaufen während ihres Lebenszyklus verschiedene Phase, beginnend bei der Bearbeitung über deren Korrektur bis zur Einsichtnahme. Dieses allgemeine Modell ist generisch genug um eine Vielzahl von Aufgabentypen und Szenarien abzubilden. Momentan sind zwei Szenarien umgesetzt: `Upload` für Aufgaben, deren Bearbeitung die Erstellung beliebiger Dokumente bzw. Artefakte erfordert, die vom Testteilnehmer in das System hochgeladen und manuell korrigiert werden sowie Komplexaufgaben für die Durchführung Prüfungen und Fragebögen.

3.5.5 Umsetzung von Komplexaufgaben

Komplexaufgaben sind, wie aus dem Datenschema ersichtlich, aus Teilaufgaben zusammengesetzt. Um einem Testteilnehmer eine Komplexaufgabe präsentieren zu können ist eine Auswahl und Zusammenstellung dieser Teilaufgaben notwendig. Die dafür zuständige Logik wird in der Implementierung des Interfaces `ComplexTaskBuilder` bereitgestellt. Es ergibt sich also eine Zweiteilung der Auswahllogik in die Auswahlparameter in der Aufgabendefinition (z.B. Antwortalternativen bei MC-Aufgaben, Anzahl auszuwählender Aufgaben pro Block, Aufgabenalternativen in den Choice-Blöcken etc.) und die Logik im Quellcode. Diese Trennung hat sich in der praktischen Umsetzung wesentlich unempfindlicher gegen Implementierungsfehler erwiesen als die komplette Umsetzung der Testlogik innerhalb der Testbeschreibung, wie sie in QTI vorgesehen ist. Beispielhaft zeigt das Sequenzdiagramm in Abbildung 25 den Ablauf der Erstellung einer individuellen Instanz einer Komplexaufgabe mit einer Erweiterungsaufgabe für Autotool. Der allgemeine Algorithmus entspricht ungefähr dieser Beschreibung:

1. Erzeuge eine Instanz von `ComplexTasklet` für einen Testteilnehmer
2. Starte einen neuen Versuch (`Try`)
3. Wähle die Unteraufgaben (`SubTaskDef`) aus einer Kategorie:
   (a) Wähle die Unteraufgaben aus den Blöcken aus (einzelne Aufgabe oder Auswahl aus einem `Choice`-Block) gemäß den vorgegebenen Parametern
   (b) Instantiiere ein `SubTasklet` pro Unteraufgabe (`ComplexTaskFactory`)
   (c) Mische die Aufgabe
4. Verteile die Aufgaben auf einzelne Seiten (`Page`)

Die Darstellung der Aufgaben wird von der Webapplikation `taskmodel-core-view` übernommen. Sie bekommt die dazu notwendigen Informationen übergeben (UserInfo, Tasklet und TaskDef) und zeigt neben einer Navigationsübersicht und einigen Informationen zum Bearbeitungsstand jeweils eine Seite mit Aufgaben an. Dazu erstellt die Klasse `SubTaskViewFactory` zu jedem `SubTasklet` eine Instanz der jeweils entsprechenden Unterklasse von `SubTaskView`. Diese Instanzen enthalten dann die Darstellungslogik dieses speziellen Tasklet-Typs sowie die entsprechende Logik zur Abbildung der gegebenen Antworten auf die entsprechenden Konstrukte des `SubTasklets`.

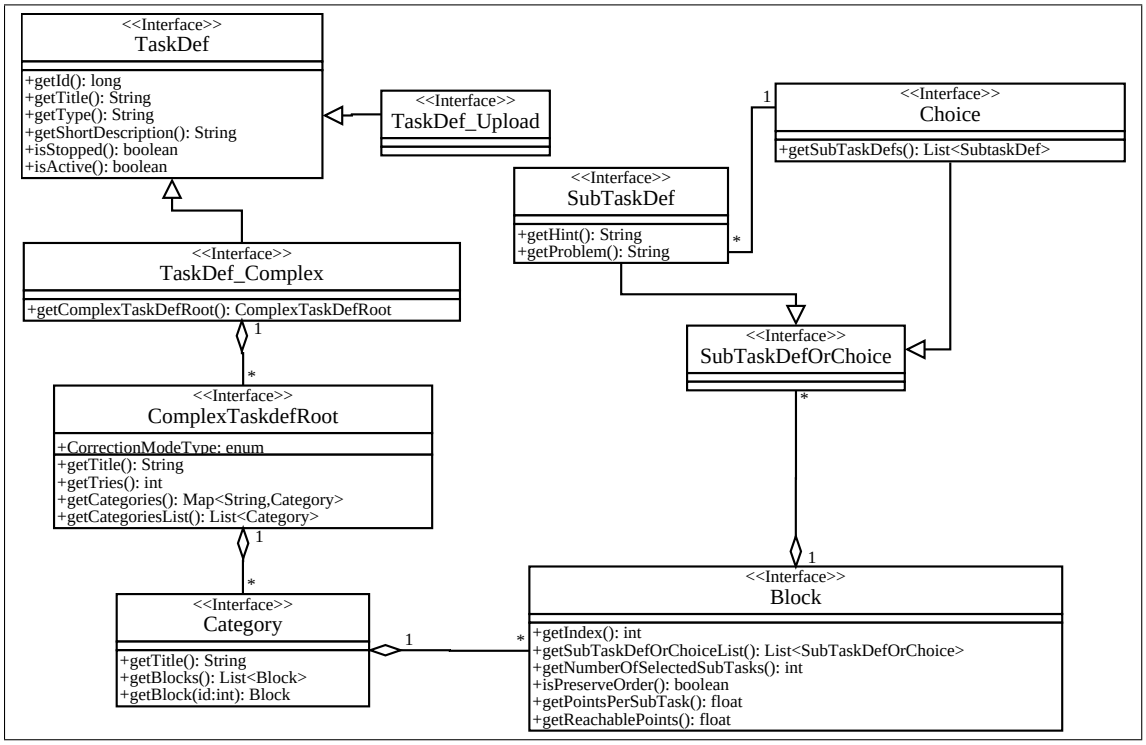


Abbildung 19: ComplexTaskDef-Klassendiagramm

3.5.6 Erweiterung der Architektur für Addontasks

Die Erweiterbarkeit des Aufgabenframeworks um neue Aufgabentypen stellt gemäß der Nomenklatur der Softwareproduktlinienentwicklung eine Einführung eines neuen Variabilitätspunktes dar. Bisher war dazu jeweils eine Erweiterung sowohl der API als auch der Referenzimplementierung notwendig. Dies stellt eine Bindung der Variabilität vor dem Kompilieren dar, ergibt also einen vergleichsweise hohen Aufwand bei der Integration. Wegen der angestrebten hohen Verfügbarkeit eines Prüfungsservers ergibt sich also die Notwendigkeit einer extrem späten Bindung, idealerweise erst zur Laufzeit. Das OSGi-Framework bietet dazu ideale Voraussetzungen, da es erlaubt, Bundles zur Laufzeit beliebig hinzuzufügen, zu verändern oder zu entfernen.

Grundlage der Erweiterung ist die Einführung weiterer Interfaces zur Beschreibung von Erweiterungsaufgaben. Notwendig war neben dem abstrakten Typen `AddonSubTask` die Einführung von Variationspunkten für die Erstellung von Instanzen von `AddonSubTasklet` und `SubTaskView`. `AddonSubTasklet` beschreibt eine Erweiterungsaufgabe in Bearbeitung. Jeder neue Erweiterungsaufgabentyp stellt eine entsprechende Unterklasse bereit. Instanzen dieser Klassen werden jeweils durch eine `AddOnSubTaskletFactory` entsprechend dem Factorypattern

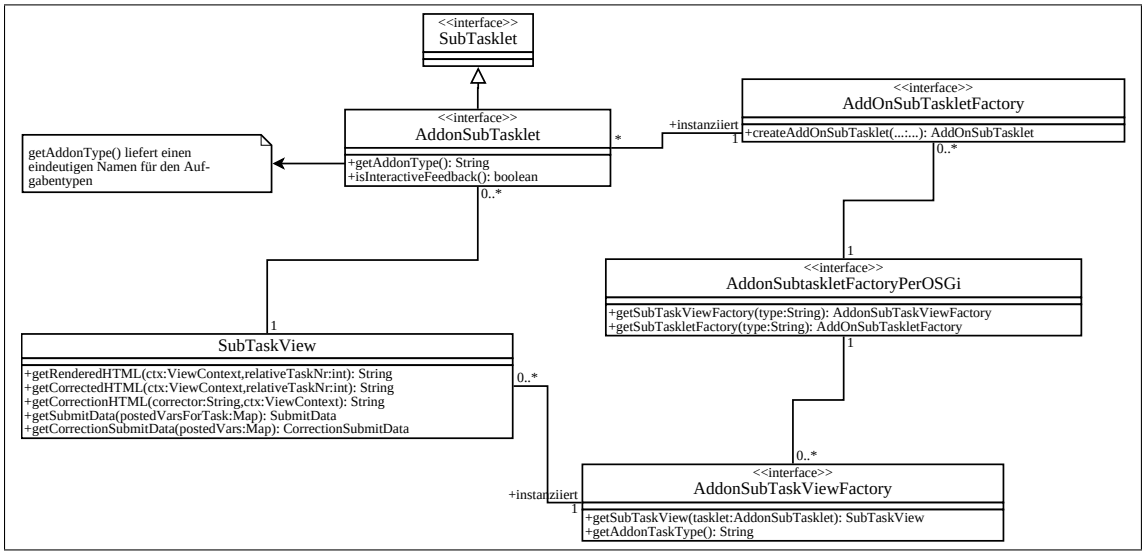


Abbildung 20: Neue Interfaces für Erweiterungsaufgaben

[GHJV94] erstellt. Analog dazu war die Erweiterung für die Darstellung der `SubTasklets` in der Webapplikation `taskmodel-core-view` notwendig. Zu jedem `AddonSubTasklet` gibt es eine passende `AddonSubTaskViewFactory`, die jeweils Instanzen von `SubTaskView` erstellt. `SubTaskViewFactory` wurde um ein dispatcher-artiges Konstrukt erweitert. Soll für ein `AddonSubTasklet` die passende Unterklasse von `SubTaskView` instantiiert werden wird im eingebundenen OSGi-Framework nach einem Bundle gesucht, das genau diese zur Verfügung stellt.

Ein neuer Aufgabentyp besteht also im Wesentlichen aus den folgenden vier Elementen:

- Implementierung von **AddonSubTasklet** mit der aufgabentypspezifischen Funktionalität
- Implementierung von **AddOnSubTaskletFactory** zur Instantiierung des entsprechenden SubTasklets
- Implementierung von **SubTaskView** mit den notwendigen Darstellungsinformationen
- Implementierung von **AddOnSubTaskViewFactory** zur Instantiierung des SubTaskViews

Wichtigstes Konstrukt ist das neue Interface `AddonSubtaskletFactoryPerOSGi`, welches eine konkrete Implementierung des OSGi-Frameworks darstellt. Diese

Kapselung führt zur Vermeidung von Abhängigkeiten von OSGi zur Kompilierungszeit innerhalb des Aufgabenframeworks. Besonders wichtig ist die Kapselung der dynamischen Eigenschaften der von OSGi bereitgestellten Services. Jedes Bundle, und damit auch jeder Service der von diesem zur Verfügung gestellt wird, kann zu jedem Zeitpunkt nicht verfügbar sein, z.B. wegen eines Updates des Bundles. Dieser Sachverhalt soll durch die Verwendung eines `ServiceTrackers` (siehe Seite 45) entsprechend transparent gemacht werden. Das Sequenzdiagramm in Abbildung 25 zeigt beispielhaft die Kollaboration der Klassen des Frameworks beim Start der Bearbeitung einer Komplexaufgabe, die aus einer Autotool-Erweiterungsaufgabe besteht.

Eine Erweiterungsaufgabe besteht also physisch aus einem OSGi-Bundle, welches zwei Services bereitstellt: `AddOnSubTaskletFactory` und `AddOnSubViewFactory`. Der `ServiceTracker` erhält eine Benachrichtigung über die Verfügbarkeit dieses speziellen Aufgabentyps und liefert diese Services beim Aufruf der entsprechenden Methoden von `TaskModelServices` zurück.

3.5.7 Realisisierung der einzelnen Aufgabentypen als Bundles

3.5.7.1 Autotool als Bundle

Der modulare Autotoolserver stellt insgesamt fünf Funktionen zur Verfügung. Der Zugriff erfolgt über eine Schnittstelle, die durch XML-RPC angesprochen werden kann. Die angebotenen Funktionen, angelehnt an deren eigentliche Syntax in Haskell, sind:

list_ types :: -> IO (Task)
: Liefert eine Liste mit allen verfügbaren Aufgabentypen, die der semantische Teil von Autotool anbietet

get_ config :: -> Task -> IO (Config)
: Generiert eine Konfigurationsbeschreibung für einen gewählten Aufgabentypen. Diese wird entsprechend angepasst, z.B. durch Veränderungen der in dieser Aufgabe zu verwendenden Wertebereiche.

verify_ config :: -> Task -> Config -> Signed Config
: Stellt sicher, dass eine geänderte Aufgabenkonfiguration sowohl syntaktisch als auch semantisch korrekt ist und signiert diese, so dass bei einer späteren Verwendung dieser Konfiguration sichergestellt werden kann, dass die Aufgabe fehlerfrei erstellt und korrigiert werden kann.

get_ instance :: -> Task -> Signed Config -> Seed -> Pair (Signed Instance Solution)

Erstellt aus einer signierten Aufgabenkonfiguration und einem zufälligen Startwert eine Aufgabeninstanz und liefert den Aufgabentext und eine syntaktisch korrekte Beispiellösung zurück.

grade :: -> Task -> Signed Instance -> Solution -> Pair (Bool Double)
Bewertet eine Aufgabenlösung und liefert neben dem Ergebnis auch einen Wert zurück, der die Güte der Lösung beschreibt. Diese kann z.B. für Highscoretabellen verwendet werden.

Die ersten drei Funktionen sind für den Aufgabenautoren interessant, sie werden für die Erstellung von Aufgaben benötigt. Die beiden letzten Funktionen werden genutzt um aus einer gegebenen, signierten Aufgabenkonfiguration Instanzen für einzelne Nutzer zu generieren und zu bewerten.

Die Integration von Autotool ist ein unkomplexes Beispiel für die Einbindung eines neuen Aufgabentypes als Bundle. Es wurde ein Bundle erstellt welches die entsprechende Kommunikationsmittel zur Anbindung an Autotool enthält, entsprechend ausprogrammierte Unterklassen von *SubTasklet* und *SubTaskView* sowie die beiden Factories, die diese Klassen auf Anfrage instantiieren. Diese werden im OSGi-Framework als Services registriert und vom Aufgabenframework entsprechend genutzt.

3.5.7.2 Zeichenaufgaben als Bundle

Dieser Aufgabentyp war etwas aufwendiger in der Umsetzung. Wegen seiner grafischen Natur war eine entsprechende Interaktionsmöglichkeit zu schaffen, ein Applet. Dieses ist Bestandteil des Bundles, die Frage ist also, wie dieses Applet von außen, also vom Testteilnehmer, aufgerufen werden kann. Die Darstellung einer Teilaufgabe, also das Ergebnis des Aufrufs einer der Methoden von *SubTaskView*, ist in der jetzigen Version des Aufgabenframeworks ein String, also reiner Text. Alle Ressourcen, die zur Darstellung sonst noch notwendig sind müssen zur Laufzeit in der Webapplikation vorliegen, in die das Aufgabenframework eingebettet ist. Das bedeutet jedoch, dass es nicht reicht ein Bundle zur Verfügung zu stellen, welches die gesamte Funktionalität kapselt, sondern das verwendete Applet bzw. anzuzeigende Grafiken müssten zusätzlich in die entsprechenden Ordner im Dateisystem kopiert werden. Hier bot sich die Verwendung des *HttpService* an, einer der Standardservices von OSGi. Mit seiner Hilfe ließ sich das Applet als per http zugreifbare Ressource anmelden. Ein Zugriff darauf wird nun also von der Webapplikation an den *HttpService* durchgereicht werden, der wiederum den Aufruf an das Bundle weiterreicht. Damit wurde erreicht, dass alle zu dieser Aufgabe gehörigen Artefakte in einem Bundle gekapselt bleiben. Ein Deployment in verschiedenen Anwendung wird so erheblich vereinfacht.

3.5.7.3 Java™-Aufgaben als verteilter Service

Der Aufgabentyp zur automatischen Korrektur von Java™-Aufgaben steht prototypisch für eine Reihe von ähnlichen Programmieraufgaben. Die Umsetzung dieses Typs kann also für andere Programmiersprachen analog betrachtet werden. Prinzipiell ist die Architektur vergleichbar mit der einer Autotoolaufgabe, d.h. es ist ein Bundle bereitzustellen, welches die geforderten Erweiterungsinterfaces implementiert. Eine Besonderheit ergibt sich jedoch aus den Sicherheitsanforderungen, die entstehen, weil ungesicherter und damit potentiell gefährlicher Quellcode ausgeführt werden soll. Es ist also unbedingt notwendig, dass dieser Code auf einem entsprechend abgesicherten Server läuft. Es ergeben sich dafür zwei mögliche verteilte Designs:

1. Anbindung eines Korrekturservers mit RMI oder
2. Anbindung über einen *RemoteOSGiService* mit R-OSGi.

Es wurde der zweite Ansatz gewählt, er ermöglicht die transparente Einbindung eines Korrekturservers unabhängig von seiner tatsächlichen Position im Netzwerk. Der Korrekturservice kann also nicht nur zur Entwicklungs- bzw. Testzeit auf demselben Server lokalisiert sein wie das Aufgabenframework, sondern ist auch im eigentlichen Einsatz hochgradig flexibel. Z.B. ist eine Migration zur Laufzeit auf einen neuen Server möglich, ohne dass der Testbetrieb unterbrochen werden muss. Auch ist ein transparentes Clustering denkbar, wenn die Anzahl zu korrigierender Aufgaben zu groß wird. All diese Möglichkeiten müssten mit dem RMI-Ansatz jeweils manuell ausprogrammiert werden. Grundlage für diese Flexibilität ist die Verwendung von SLP zur automatischen Entdeckung von OSGi-Services in einem Netzwerk. Auf einem abgesicherten Server läuft eine eigenständiges OSGi-Framework, welches, wie das Aufgabenframework auch, die Bundles für R-OSGi und SLP ausführt. Im Aufgabenframework wird ein *ServiceDiscoveryListener* gestartet, der informiert wird, sobald ein neuer Korrekturservice vorhanden ist. In diesem Fall baut er eine Verbindung zu dem betreffenden Server auf und R-OSGi erstellt eine Proxyinstanz des Services, die dann transparent lokal zur Verfügung steht.

4 Implementierung

Die Referenzimplementierung des OSGi-Frameworks im Aufgabenframework findet sich in der Klasse *de.elatePortal.engine.EquinoxAddonTaskFactoryImpl* des Unterprojekts *elatePA* und verwendet das Framework „Equinox“ der Eclipse Foundation. Dieses zeichnet sich wegen der Verwendung als Grundlage der Entwicklungsumgebung Eclipse und der damit verbundenen kontinuierlichen Weiterentwicklung durch eine deutlich höhere Zukunftssicherheit aus als andere Opensource-Implementierungen wie z.B. *Knopflerfish* [23] oder *Apache Felix* [24].

Neben der Equinox-spezifischen Initialisierung gibt es in dieser Klasse zwei Methoden, die entsprechend auch für beliebige andere OSGi-Frameworks verwendet werden können. Es werden zwei Instanzen des ServiceTrackers erstellt. Sie beinhalten jeweils die Informationen darüber, welche Aufgabentypen im Framework vorhanden sind. Sie werden vom OSGi-Framework jeweils darüber informiert, wenn Bundles Implementierungen der entsprechenden Interfaces *AddOnSubTaskletFactory* bzw. *AddonSubTaskViewFactory* neu bereitstellen, aktualisieren oder löschen. Dieses auch als „whiteboard pattern“[All04] bekannte Pattern ermöglicht eine wesentlich unkompliziertere Handhabung von passenden Services. Diese Trackerinstanzen finden dann bei Bedarf die zu einer ID passenden Services, welche die entsprechenden Instanzen für die zusätzlichen Aufgabentypen generieren:

```
...
private ServiceTracker subTaskletFactoryTracker;
private ServiceTracker subTaskletViewFactoryTracker;
...
public EquinoxAddonTaskFactory(FileRepositoryManager
   fileRepositoryManager) {
  ...
  subTaskletFactoryTracker =
    new ServiceTracker( bctx,
                    AddOnSubTaskletFactory.class.getName(),
                    null);
  subTaskletFactoryTracker.open();
  subTaskletViewFactoryTracker =
    new ServiceTracker( bctx,
                    AddonSubTaskViewFactory.class.getName(),
                    null);
  subTaskletViewFactoryTracker.open();
}
```

[23] www.knopflerfish.org
[24] http://felix.apache.org

```
public AddonSubTaskViewFactory getSubTaskViewFactory(String id) {
  Object[] factories=subTaskletViewFactoryTracker.getServices();
  for (Object o : factories) {
    AddonSubTaskViewFactory f=(AddonSubTaskViewFactory) o;
    if(f.getAddonTaskType().equals(id))
      return f;
  }
  return null;
}

public AddOnSubTaskletFactory getSubTaskletFactory(String id) {
  Object[] factories=subTaskletFactoryTracker.getServices();
  for (Object o : factories) {
    AddOnSubTaskletFactory f=(AddOnSubTaskletFactory) o;
    if(f.getAddonTaskType().equals(id))
    return f;
  }
  return null;
}
...
```

Listing 1: Auffinden von zusätzlichen Aufgabentypen

4.1 Schemaerweiterung

Dem Aufgabenframework des elatePortal-Projekts liegt eine Datenhaltung auf XML-Basis zugrunde. Die Struktur der XML-Dokumente wurde mit einem XML-Schema definiert. Dieses enthält Datentypen für Komplexaufgaben, Aufgabenblöcke, MC-Aufgaben, Zuordnungsaufgaben, Lückentextaufgaben und Freitextaufgaben. Dieses Schema ermöglicht die Verwendung von Werkzeugen, welche nach dem MDA-Prinzip z.B. Programmquelltext, Editoren oder Validatoren erstellen. Es stellte sich jedoch heraus, dass dieses Schema nur eingeschränkt erweiterbar ist. Speziell ist es nur mit hohem Aufwand möglich, mehrere unabhängige Erweiterungen des Schemas wieder miteinander in einem gemeinsamen Schema zu vereinigen. Selbst nachdem dies geschehen wäre, müsste bei allen Beteiligten daraus neuer Quelltext erstellt werden, das Framework neu kompiliert und gestartet werden. Der in dieser Arbeit verfolgte Ansatz bestand daher darin, im Schema einen neuen Aufgabentypen *AddonTask* zu erstellen. Er erbt alle Eigenschaften des abstrakte Typs SubTaskDefType, erweitert ihn jedoch um das Element Memento sowie einen Bezeichner *taskType*. Memento erlaubt beliebige XML-Strukturen als

Inhalt. Auf diese Weise kann eine Implementierung eines neuen Aufgabentyps beliebige Informationen speichern, ohne sie explizit vorher im Schema definieren zu müssen. Der Nachteil dieser Herangehensweise besteht jedoch darin, dass Werkzeuge, die auf diesem Schema und den dafür gültigen XML-Daten agieren diese Strukturen nicht validieren können, da eben diese Strukturinformationen fehlen. Jede Implementierung eines Aufgabentypes ist also selbst für syntaktisch korrekte Inhalte in diesem Eintrag verantwortlich. Das neue Attribut *taskType* stellt einen eindeutigen Bezeichner für diesen Aufgabentypen dar. Dieser wird z.B. verwendet, um die entsprechenden Implementierungs- bzw. Visualisierungsklassen zur Laufzeit zu finden.

```
...
<xs:extension base=SubTaskDefType>
  <xs:sequence>
    ...
    <xs:element name="Memento" type=xs:anyType>
      <xs:annotation>
        <xs:appinfo>
          <xjc:dom />
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  ...
  <xs:attribute name="taskType"
    type=xs:string use=required>
  </xs:attribute>
</xs:extension>
...
```

Listing 2: Erweiterung des Schemas

4.2 Autotool-Aufgaben

Autotool stellt einen zustandslosen Server zur Verfügung, der mit Hilfe von XML-RPC alle relevanten Funktionen zum Generieren und Bewerten von Aufgaben anbietet. Die Anbindung dieses Servers basiert dabei auf XML-RPC [Win99]. Dabei handelt es sich um ein Protokoll auf der Basis von http zum Aufruf von Funktionen, die Übertragung der Aufrufparameter und der Ergebnisse erfolgt in XML-Form.

Die Hauptschwierigkeit bei der Umsetzung der Anbindung (siehe die Schnittstel-

lenbeschreibung auf S. 60) ist die fehlende Typisierung der Daten, die per XML-RPC verschickt werden können. Die verwendbaren einfachen Typen beschränken sich auf `boolean`, `integer`, `double`, `dateTime`, `string` und `base64`. Komplexere Typen lassen sich nur als `struct` abbilden, d.h. als Zuordnung von Namen auf Werte. Die Implementierung des Clients konnte deshalb nicht statisch vom Compiler auf Korrektheit geprüft werden. Statt dessen wurde ein Java™-Client mit den entsprechenden Methoden geschrieben, der Aufrufe an Autotool delegiert und die Antworten als semantisch äquivalente Java™-Klassen zurückliefert. Einen Ausschnitt aus der physischen Kommunikation zeigt Anhang C. Dort ist ein Mitschnitt des Aufrufs der Funktion „autotool.grade“ zur Bewertung einer Aufgabe aufgeführt. Aus Gründen der Übersichtlichkeit wurde jedoch auf unnötige Details verzichtet.

```
public interface AutotoolService{

public List<String> getTaskTypes();

public AutotoolTaskConfig getConfig(String taskType);

public SignedAutotoolTaskConfig getSignedConfig(AutotoolTaskConfig
    config);

public AutotoolTaskInstance getTaskInstance(
   SignedAutotoolTaskConfig cfg, int seed);

public AutotoolGrade gradeTaskInstance(AutotoolTaskInstance inst,
   String solution);

}
```

Listing 3: Java™-Interface für Autotool

Die Implementierung von `AutotoolService` kapselt die Kommunikation mit Autotool, so dass es genau wie ein lokal verfügbarer Aufgabentyp verwendet werden kann. Der Ausschnitt 4 aus dem Quelltext zeigt beispielhaft zwei Methoden, die auf diese Schnittstelle zugreifen:

```
...
/**
 * Bewerte die Aufgabe automatisch durch Delegation an Autotool.
 * Im Falle einer Exception (etwa durch fehlende Netzwerk-
 * verbindung) muss die Korrektur entweder erneut angestoßen
 * oder aber manuell durchgeführt werden.
 */
```

```
public void doAutoCorrection(){
  try {
    //Instanz der Autotoolaufgabe des aktuellen Studenten
    AutotoolTaskInstance ati =
      new AutotoolTaskInstance.AutotoolTaskInstanceVO(
        autotoolTaskConfig.getTaskType(),
        (Map) deserialize( autotoolSubTask.getAutotoolInstanceBlob()
            ) );

    //sende eingegebene Lösung an Autotool
    AutotoolGrade grade =
      getAutotoolServices().gradeTaskInstance(ati,getAnswer());

    //speichere die Korrektur
    setCorrection(
     (float) ( grade.isSolved() ? block.getPointsPerSubTask(): 0 ),
     grade.getGradeDocumentation(),
     true);

    autotoolSubTask.setAutotoolScore( grade.getPoints() );
    autotoolSubTask.setLastCorrectedAnswer( getAnswer() );
  }catch (Exception e) {
    logger.log( e );
  }
}
...
/**
 * Erstelle eine neue Instanz der Autotoolaufgabe mit Hilfe eines
 * Zufallswertes. Alternativ ist auch die Matrikelnummer des
 * Studenten denkbar.
 */
public void build() throws TaskApiException {
  //signierte Konfigurationsparameter dieser Autotoolaufgabe
  SignedAutotoolTaskConfig satc =
    new SignedAutotoolTaskConfig.SignedAutotoolTaskConfigVO(
      this.autotoolTaskConfig.getTaskType(),
      autotoolTaskConfig.getConfigString(),
      "",
      autotoolTaskConfig.getSignature());

  //erstelle eine individualisierte Instanz der Aufgabe
  try {
```

```
    AutotoolTaskInstance task =
      getAutotoolServices().getTaskInstance( satc, RandomUtil.getInt
          ( 1000000 ));
    autotoolSubTask.setProblem( task.getProblem() );
    autotoolSubTask.setDefaultAnswer( task.getDefaultAnswer() );
    autotoolSubTask.setAnswerDoc( task.getAnswerDoc() );
    autotoolSubTask.setAutotoolInstanceBlob( serialize( task.
       getSignedInstance()));
  } catch (Exception e) {
    logger.log( e );
  }
}
...
```

Listing 4: Ausschnitt aus `AutotoolSubtasklet`

4.2.1 Beispieldefinition einer Autotoolaufgabe

Der eindeutige Bezeichner, also der Wert der Schemavariable *taskType* ist „autotool“. Das Element „Memento“ enthält vier Unterelemente:

taskType
: entspricht einem der von „list_types“ zurückgelieferten Strings

configString
: Instanz einer Aufgabenkonfiguration, Ergebnis von „get_config“

signature
: Signatur der Aufgabenkonfiguration zum Schutz vor Veränderung, Ergebnis von „verify_config“

autotoolServerUrl
: URL des modularen Autotool-Servers, wird benötigt um relative URLs in den Aufgabenstellungen umschreiben zu können

Ein Beispiel für die Aufgabenstellung „Geben Sie den minimalen deterministischen Automaten zu folgendem nichtdeterministischen Automaten an.“ ist folgendes:

```
<addonSubTaskDef
   id="auto1"
   taskType="autotool"
   interactiveFeedback="true">
```

```
  <Memento>
    <taskType>Equiv-Quiz</taskType>
    <configString>
      Conf { alphabet = mkSet "ab" , nfa_size = 7 , min_dfa_size = 9
      , max_dfa_size = 13
    }</configString>
    <autotoolServerUrl>
      http://dfa.imn.htwk-leipzig.de/cgi-bin/modular-server.cgi
    </autotoolServerUrl>
    <signature>-150108046754628</signature>
  </Memento>
</addonSubTaskDef>
```

Listing 5: XML für eine Autotoolaufgabe

4.3 Umsetzung der Zeichenaufgabe

Zeichenaufgaben wurden mit Hilfe eines Swing-Applets umgesetzt. Basis ist ein entsprechend implementiertes `JPanel`, welches die Darstellung der Benutzereingaben übernimmt. Dazu kommt eine `JToolBar` mit den einzelnen Aktionen (Zeichen, Löschen, Strichstärke, Farben, Undo-Funktion). Eine, speziell im Falle einer Prüfung, besonders wichtige Funktion ist ebenfalls enthalten: Undo und Redo. Interaktionen mit der Zeichenfläche lassen sich über mehrere Schritte rückgängig machen als auch wiederholen, auch nachdem die Aufgabe gespeichert wird und wieder geladen wird. Damit wird verhindert, dass ein Student, der etwa kurz vor der Ablauf der regulären Bearbeitungszeit aus Versehen das Gezeichnete löscht und abspeichert diese Aufgabenlösung verliert. Außerdem ergibt sich aus diesem „Zeichenprotokoll“ ein zusätzliches Logging der Benutzereingaben.

Eine Aufgabendefinition besteht aus bis zu drei einzelnen Bildern: Dem Hintergrund, der vom Studenten nicht verändert werden kann, einem veränderlichen Lösungsvorschlag (etwa als Hinweis zur Bearbeitung) sowie einer Lösungserwartung, die nur Tutoren als Korrekturhilfe angezeigt wird.

Implementiert wurde das Zeichenapplet als eigenständiges Projekt `drawTask`. Es kann nicht nur als Aufgabentyp sondern auch als eigenständiges, wenn auch vergleichsweise funktionell eingeschränktes Zeichenprogramm verwendet werden, etwa als Komponente in einer Swingapplikation.

Auf eine umfassende Darstellung der Implementierungsdetails soll an dieser Stelle verzichtet werden um den Rahmen dieses Kapitels nicht zu sprengen. Einen Ein-

druck der Darstellung des Zeichenapplets ist jedoch im Anhang in der Abbildung 27 zu sehen.

4.4 Umsetzung von Java™-Aufgaben

Die Aufgabe war, zur Laufzeit ein in Textform vorliegendes Programm zu kompilieren, zu laden und mit Hilfe von Unittests zu verifizieren. Damit ergaben sich zwei wesentliche Aufgaben:

- Übersetzen des Quellcodes in lauffähigen Bytecode
- Test der semantischen Korrektheit des Codes

Im folgenden sollen nun die verschiedenen verfolgten Ansätze erläutert werden.

4.4.1 Kompilieren des Quelltextes mit Hilfe von Skriptsprachen

Der erste Ansatz war die Verwendung einer dynamischen Skriptsprache für die JVM mit Java™-ähnlicher Syntax. Dazu wurden Beanshell[25] [NK05] und Groovy [Kön07] evaluiert.

BeanShell bietet die Möglichkeit Java™-Code mit einer so genannten „strict"-Einstellung zu parsen. Diese Einstellung bewirkt die Deaktivierung aller BeanShell-spezifischen Sprachkonstrukte, so dass nur gültige Java™-Syntax akzeptiert wird. Leider konnte diese Skriptsprache wegen fehlender Unterstützung von inneren Klassen nicht verwendet werden, die daraus resultierenden Einschränkungen für kompilierbaren Quelltext sind nur für einfachste Aufgabenstellungen annehmbar.

Der zweite Kandidat war Groovy. Dabei handelt es sich um eine im JSR 241 [Laf07] standardisierte dynamische Skriptsprache für die JVM. Sie besteht aus einer Obermenge der zulässigen Java™-Syntax und einer umfassenden Bibliothek von zur Java™-API supplementären Klassen. Das für diese Arbeit interessante ist besonders die Eigenschaft, dass syntaktisch valider Java™-Quellcode auch als Groovy-Skript lauffähig ist, so dass es in einem Prototyp eingesetzt werden konnte.

[25] http://bsf.apache.org

4.4.2 Kompilieren des Quelltextes zu Bytecode

Der zweite Ansatz verwendete die im frei verfügbaren JDK von Sun in der Bibliothek tools.jar verfügbare Klasse com.sun.tools.javac.Main, welche die Implementierung eines kompletten Compilers für Java™enthält. Ist diese Klasse zur Laufzeit vorhanden, kann also ein in Textform vorhandener Quelltext in Bytecode übersetzt und mit einem entsprechenden Classloader dynamisch geladen werden. Dabei war zu beachten, dass es bei jeder Aufgabe pro Student und pro Versuch eine neue Ausprägung der Lösung gab. Damit war ein einfaches Laden und Instantiieren der Klasse nicht möglich. Dieses Problem konnte jedoch mit Umbenennen der zu ladenden Klasse und einem speziellen Classloader gelöst werden.

4.4.3 Tests

Für die Überprüfung der semantischen Korrektheit ergaben sich wieder zwei mögliche Ansätze:

- Aufzählung von Eingaben mit erwarteten Ausgaben des zu testenden Codes
- Definition von beliebigem Testcode (Unittests)

Die Aufzählung von Testdaten mit Angaben zu erwarteten Ausgaben stellt eine einfache doch auch umfassende Art des Tests dar. Leider ergeben sich dabei zwei Probleme:

- Es wird eine nichttriviale Grammatik benötigt, um diese Daten beschreiben zu können. Außerdem wären ein Parser und Lexer zu schreiben, ein hoher zusätzlicher Implementierungsaufwand. Dazu wäre der Nutzer gezwungen sich diese neu definierte Sprache aneignen zu müssen, um Tests definieren zu können.
- Es können keine Faktoren wie Effizienz des Codes überprüft werden.

Das Schreiben von Unittests erfolgt in der Regel in einer vollständigen Programmiersprache, in diesem Fall in Java™. In dieser Arbeit wurde nun JUnit eingesetzt. JUnit ist ein Werkzeug zum Definieren und Ausführen von Unittests. Damit stehen dem Autoren der Aufgabe beliebige Testszenarien zur Verfügung. Er wird nicht beschränkt auf die Bereitstellung von Eingabedaten mitsamt den erwarteten Ausgaben. Durch die Möglichkeit, beliebige Tests selbst in einer Programmiersprache beschreiben zu können können auch andere weiche Anforderungen getestet werden, wie z.B. Laufzeiten, Introspektionen in den Code etc.

Die Testklasse muss das Interface JavaTaskTester implementieren, um sicherzustellen, dass der Test Instanzen der zu testenden Klasse bekommt. Auf diese Weise wird verhindert, dass der Test mit Hilfe des java.reflection-Pakets [26] selbst Instanzen von zur Compilezeit unbekannten Klassen erstellen muss:

```
public interface JavaTaskTester{
  /**
   * Setter für das Objekt, das getestet werden soll.
   */
  public void setTestObject(Object toTest);

}
```

4.4.3.1 *Beispielaufgabe*

Ein Beispiel wäre ein Unittest für die Aufgabe, ein „Hello world!"-Programm zu schreiben. Es ist also eine Klasse zu schreiben, die folgendes Interface implementiert:

```
public interface IGreetings{
  public String greet();
}
```

Der entsprechende Unittest besteht in diesem Beispiel aus zwei Tests, einem Vergleich des Strings mit dem erwarteten „Hello world!" und einem Vergleich der Stringlänge:

```
import junit.framework.*;
import junittask.JavaTaskTester;

/**
 * Einfacher Testfall, erwartet "`Hello world!"' als Antwort.
 */
public class DummyTest extends TestCase implements
    JavaTaskTester {

  private Object toTest;

  public void testContent() {
    IGreetings toTest=(IGreetings) this.toTest;
    assertEquals("Hello world!",toTest.greet());
```

[26] API zur Introspektion

```
  }

    public void testLength() {
      assertEquals("Hello world!".length(), toTest.greet().length())
          ;
    }

  public void setTestObject(Object toTest) {
    this.toTest=toTest;
    }
}
```

4.4.4 Sicherheitsaspekte

Durch das Ausführen von beliebigem Quellcode auf dem Prüfungserver entsteht eine große Sicherheitslücke: Studenten könnten mit den Rechten des laufenden Serverprozesses beliebigen Code laufen lassen. Eine Möglichkeit, Schadcode zu verhindern ist die Verwendung eines speziellen Classloaders, der nach dem Blacklist-Prinzip das Verwenden aller sicherheitsrelevanten Klassen und Pakete verhindert. Diese Lösung bedeutete jedoch, dass der Classloader über jede in der JVM verfügbare Bibliothek entsprechend informiert werden muss, um den Zugriff entsprechend zu gestatten bzw. zu verhindern. Damit war dieser Ansatz kaum wartbar und damit nicht einsetzbar.

Um nun die durch eingesandten Code nutzbaren Funktionen einzuschränken wurde die in der JRE verfügbare Sicherheitsarchitektur „Java Security“ [SUN07] verwendet. Es wird eine zweite JVM in einer Sandbox gestartet, welche ähnlich den Bedingungen beim Ausführen eines Applets nur die absolut notwendigen Funktionen zulässt, alles andere jedoch unterbindet. In dieser abgesicherten JVM werden nun alle zu testenden Klassen ausgeführt und die Testergebnisse per RPC zum Aufgabenframework zurückgeschickt. Alle Klassen, deren Code als Lösung einer Java™-Aufgabe eingesandt werden bekommen eine Codebase zugeordnet. Darunter versteht [SUN07] einen beliebigen eindeutigen String, durch den die Regeln für entsprechend zulässig Aktionen durch den SecurityManager zugeordnet werden können. Diese Zuordnung erfolgt in einem sogenannten *policyfile*, einer Textdatei mit dem Inhalt:

```
grant codeBase "/restrictedCode" {
};
```

Damit werden sämtliche sicherheitsrelevanten Klassen und deren Methoden ge-

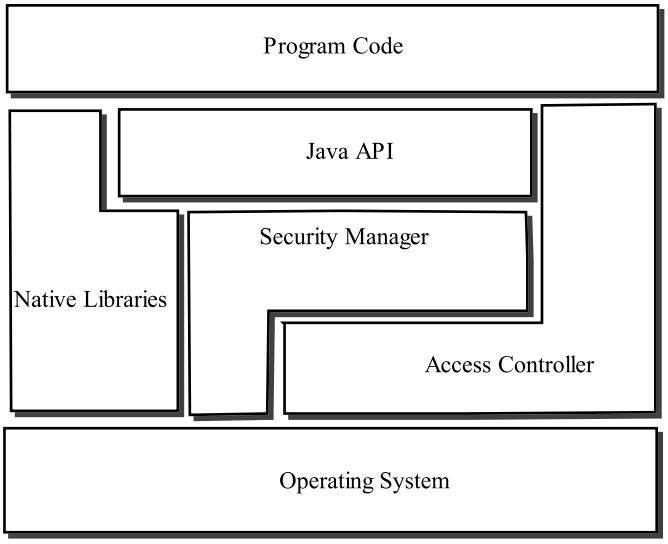


Abbildung 21: Sicherheitsarchitektur in der JVM (nach [LG03])

sperrt, es ist also weder möglich, direkt auf den Hostrechner bzw. die laufende JVM zuzugreifen noch einen Kommunikationskanal zu öffnen, sei es über eine Netzverbindung oder eine Datei. Durch diesen Mechanismus ergibt sich eine Sicherheit, die eng an die entsprechende Implementierung in der JVM gebunden ist. Die darin enthaltene Erfahrung mit dem Umgang mit unsicherem Code in Applets kann also gut als Vertrauensbasis genutzt werden.

4.4.5 Verteilter Korrekturservice

Ein wichtiger Aspekt bei der Ausführung von Code aus unsicheren Quellen ist die Frage der Sicherheit. Aus diesem Grund bietet sich an die Funktionalität dieses Aufgabentyps auf zwei Server zu verteilen. Im Aufgabenframework selbst ist das eigentliche Bundle des Aufgabentyps vorhanden. Zur Korrektur wird jedoch ein eigenständiger Service genutzt. Dieser soll nun entweder zur Entwicklungszeit lokal innerhalb derselben Plattform vorhanden sein oder aber im produktiven Einsatz auf einem eigens abgesicherten Server ausgeführt werden. Aus diesem Grund wurde OSGi zusammen mit dem Projekt R-OSGi (siehe Kapitel 3.4.3) genutzt, um diese Funktionalität zu erreichen. Die Schnittstelle im Listing 6 stellt den Zugriffspunkt auf einen Korrekturservice dar.

```
package correction.junit;

public interface JUnitTestCorrector {
 /**
```

```
  * Testet eine Java-Klasse mit Hilfe eines Unittests.
  * @param testInterfaceDef Interface der zu testenden Klasse
  * @param classUnderTest Klassendefinition die getestet werden soll
  * @param junitTestClass JUnit-Testklasse
  * @param timeOut Max. Berechnungszeit pro Unittest
  * @return Status des Unittests und Punktzahl
  */
  public JUnitTestResult runUnitTest(
    String testInterfaceDef,
    String classUnderTest,
    String junitTestClass,
    long timeOut);
}
```

Listing 6: Schnittstelle zu einem Korrekturservice

Die Implementierung dieses Services wurde als eigenes Bundle `junit-addon-corrector` realisiert und kann sowohl in das lokale OSGi-Framework installiert werden als auch auf einem eigenen Server ausgeführt werden. Im zweiten Fall wird dieser Service mit Hilfe von R-OSGi dem Aufgabenframework zur Verfügung gestellt. Um nun den Serviceconsumern den Zugriff auf diesen Service transparent gestalten zu können wurde ein erweiterter ServiceTracker verwendet, der beide Fälle abdeckt und Zugriffe auf die jeweils tatsächlich vorhandenen Services delegiert. Die dazu verwendete Klasse `CorrectorServiceTrackerProxy` setzt die Kommunikation mit Hilfe von R-OSGi auf und verwendet Instanzen von `JUnitTestCorrector` als Korrekturservices.

```
/**
 * Diese Klasse ist ein Adapter für einen JUnitTestCorrector. Sie
 * delegiert Aufrufe an einen tatsächlich vorhandenen lokalen
 *  oder verteilten JUnitTestCorrector-Service.
 */
public class CorrectorServiceTrackerProxy
      extends ServiceTracker
      implements JUnitTestCorrector,RemoteServiceListener{

  private JUnitTestCorrector service;
  private RemoteOSGiService ros;

  public CorrectorServiceTrackerProxy(
      BundleContext context,
      String remoteUrl){
```

```
    super(context, JUnitTestCorrector.class.getName(),null);
    //starte ServiceTracker
    this.open();

    ServiceReference remoteserviceref = context.getServiceReference
        ( RemoteOSGiService.class.getName() );
    if( remoteserviceref != null ) {
      //R-OSGi ist vorhanden
      Dictionary<String, Object> map = new java.util.Hashtable<
          String, Object>();

      map.put(RemoteServiceListener.SERVICE_INTERFACES,new String[]
          { JUnitTestCorrector.class.getName() });

      //starte einen RemoteServiceListener um Instanzen von
          JUnitTestCorrector
      //im Netzwerk lokal verfügbar zu machen (nutzt SLP)
      context.registerService( RemoteServiceListener.class.getName
          (), this, map );

      try{
        //direkte Verbindung zu einer anderen Instanz von OSGi
        this.ros = (RemoteOSGiService) context.getService(
            remoteserviceref );
        URI uri = new URI( remoteUrl );
        logger.log("[CorrectorProxy] connecting to "+uri);
        ros.connect( uri );
      }catch(RemoteOSGiException ex) {
        logger.log("No remote connection: ");
        ex.printStackTrace();
      }
    }
  }
...
  /**
   * Einzige Methode von JUnitTestCorrector.
   */
  public JUnitTestResult runUnitTest( String testInterfaceDef,
          String classUnderTest,
          String junitTestClass,
          long timeOut ) {
```

```
    //Service lokal vorhanden? (lokales Bundle oder Proxy von R-
       OSGi)
    //Frage den ServiceTracker.
    JUnitTestCorrector tester = (JUnitTestCorrector) getService();
      if( tester == null ) {
        tester = service;
      }
      if( tester == null ) {
        logger.log("[CorrectorProxy] no corrector service present...
            ");
        return null;
      }
      else
        //nutze Korrekturservice
        return tester.runUnitTest( testInterfaceDef, classUnderTest
           , junitTestClass, timeOut );
  }
...
```

Derselbe Mechanismus kann auch für Loadbalancing-Szenarien verwendet werden. Da `CorrectorServiceTrackerProxy` immer eine beliebige vorhandene Instanz des Service `JUnitTestCorrector` nutzt können mehrere im Netzwerk vorhandene Instanzen je nach Auslastung angesprochen werden. Die daraus resultierende Skalierbarkeit ermöglicht auch Anwendungen mit einer hohen Auslastung des Korrekturservices, etwa Programmierwettbewerbe.

4.5 Workflow für Übungsaufgaben

Um etwa bei Programmieraufgaben ein unmittelbares Feedback zu ermöglichen ohne die gesamte Komplexaufgabe abgeben zu müssen wurde die Möglichkeit eingefügt `SubTasklets` direkt korrigieren zu lassen. Dazu war eine Erweiterung der Darstellungsschicht sowie kleinere Änderungen an den entsprechenden `Struts-Actions` des `taskmodel-core-view` notwendig, etwa im Code zur Speicherung einer Aufgabenseite:

```
...
  private void processInteractiveTasklets(
    ComplexTasklet ct,
    List<SubTasklet> subtasklets,
    List<SubmitData> submitDatas,
```

```
    HttpServletRequest request) {

    for ( int i = 0; i < subtasklets.size(); i++ ) {
      SubTasklet subTasklet = subtasklets.get(i);

      //wurde der Knopf für interaktives Feedback gedrückt?
      if( request.getParameterMap().containsKey(
          "doAutoCorrection_" + subTasklet.getVirtualSubtaskNumber()
              ) ) {

        if( !subTasklet.isInteractiveFeedback() )
          //illegaler Versuch, etwa durch manipulierte Requests
          throw new SecurityException( ... );

        ct.doInteractiveFeedback(subTasklet,submitDatas.get(i));
      }
    }
  }
...
```

Listing 7: Erweiterung von ExecuteAction.java

Eine Bespieldarstellung des Feedbacks ist im Anhang in der Abbildung 26 zu sehen. Gut zu erkennen ist in dieser Darstellung die modale Darstellung. Der Student hat die Möglichkeit unmittelbar die Antwort der automatischen Korrektur zu sehen und entsprechende Änderungen an seiner Lösung vorzunehmen ohne dazu die bearbeitete Aufgabe abgeben zu müssen. Abbildung 22 zeigt zusätzlich ein vergleichendes Sequenzdiagramm des Workflows für das Speichern einer Seite mit Aufgaben.

4.6 Erweiterungen am Buildprozess

Die Erweiterungen am Aufgabenframework ließen sich in den aktuellen Buildprozess einfügen. Es gibt eine neue Bean in der Springkonfiguration mit dem Namen „OSGiFramework“, die das verwendete OSGi-Framework kapselt.

Wesentlich wichtiger sind die Änderungen, die nötig waren um die neuen Aufgabentypen kompilieren und zu Bundles packen zu können. Um ein OSGi-kompatibles Bundle zu erstellen ist die Erstellung einiger notwendiger Einträge in die Manifest.MF des jar-Archivs notwendig. Diese Einträge können je nach Umfang der verwendeten Bibliotheken und eigenen Pakete eine große Zahl erreichen.

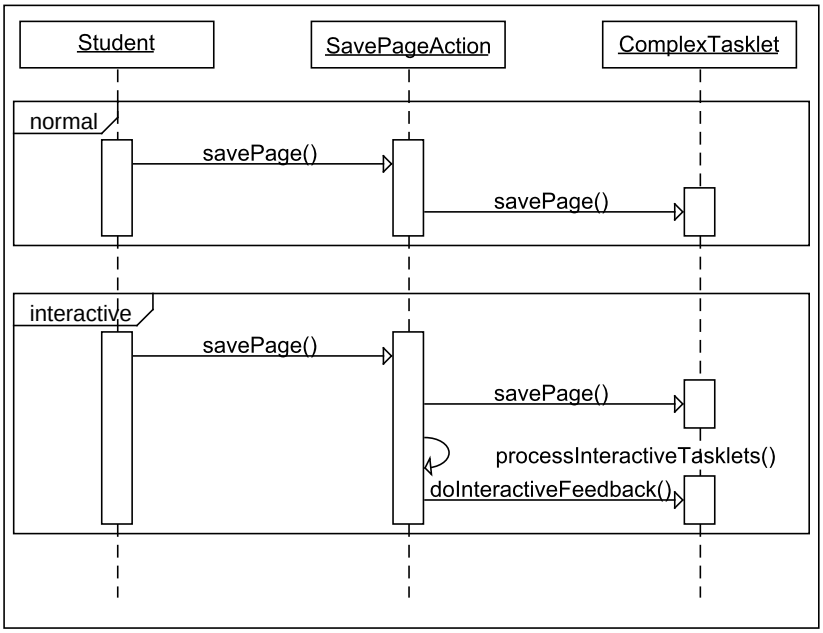


Abbildung 22: Workflow im Vergleich - bisher und interaktiv

Hier hat sich der Verwendung des von Peter Kriens entwickelte Bnd[27] als sehr nützlich erwiesen. Diese Software besteht sowohl aus einer Kommandozeilenanwendung zum Generieren von Bundles als auch einem Plugin für das Buildtool Ant[28].

Wegen des Einsatzes von Maven in der Version 1.0.2 konnte leider das durch das Apache Felix bereitgestellte Plugin zur Erstellung von OSGi-Bundles nicht genutzt werden, es setzt mindestens Version 2.0 voraus. Stattdessen wurde eine wiederverwendbare Erweiterung für Maven 1.0.2 geschrieben. Diese ruft Bnd mit der zum Aufgabentyp gehörigen Definitionsdatei für die Manifesteinträge als Parameter auf und kopiert, wenn notwendig, verwendete Bibliotheken in das Bundle. Das so entstandene Jar-Archiv kann dann problemlos im OSGi-Framework verwendet werden:

```
...
<goal name="build" prereqs="java:compile,osgi:copyjars">
  <ant:delete file="${maven.build.dir}/${maven.final.name}.jar"/>
  <ant:taskdef name="bnd" classname="aQute.bnd.ant.BndTask">
    <ant:classpath>
```

[27] http://www.aqute.biz/Code/Bnd
[28] http://ant.apache.org/

```
      <ant:pathelement
         location="${plugin.getDependencyPath('biz.aqute:bnd')}"/>
      <ant:path refid="maven.dependency.classpath"/>
    </ant:classpath>
  </ant:taskdef>
  <ant:bnd classpath="${maven.build.dir}/classes"
    eclipse="false"
    failok="false"
    exceptions="true"
    files="junittask.bnd"
    output="${maven.build.dir}/${maven.final.name}.jar"/>
 <echo message="Done building bundle via BND."/>
</goal>

<goal name="osgi:copyjars"
  description="Copy all dependent JARs">

  <ant:echo>
    Copy dependent JARs into ${maven.build.dir}/lib
  </ant:echo>
  <ant:mkdir dir="${maven.build.dir}/lib"/>
  <j:forEach var="lib" items="${pom.artifacts}">
    <j:set var="dep" value="${lib.dependency}"/>
    <j:if test="${dep.getProperty('osgi.bundle')=='true'}">
      <ant:copy todir="${maven.build.dir}/lib" file="${lib.path}"/>
    </j:if>
  </j:forEach>
</goal>
...
```

Listing 8: Ausschnitt aus dem Buildskript

Beim Beispiel des Java™-Aufgabentypes sieht die verwendete Eingabedatei für Bnd folgendermaßen aus:

```
Bundle-Name: junit-addon
Bundle-Activator: junittask.activator.Activator
Bundle-SymbolicName: junit-addon
Extension-Name: junit-addon
Bundle-ManifestVersion: 2
Private-Package: junittask.*
Import-Package: *;resolution:=optional
Bundle-ClassPath: .,junit-3.8.1.jar, groovy-all-1.1-BETA-2.jar
```

```
Include-Resource: target/lib/
```

Listing 9: Bnd-Definitionen für die Java™-Aufgabe

Die wichtigsten Einträge sind dabei *Bundle-Activator* als Aktivator, der die Services des Bundles im Framework registriert und *Bundle-ClassPath*, der die privat verwendeten Bibliotheken auflistet. Diese Datei ist deutlich kleiner und aussagekräftiger als die resultieren Manifest.MF, welche insgesamt 72 Einträge umfasst.

5 Zusammenfassung

Die vorliegende Arbeit bietet einen Überblick über die wichtigsten Softwarewerkzeuge zur Durchführung von vorlesungsbegleitenden Übungen und Prüfungen an der Universität Leipzig und dem weiteren deutschen Hochschulraum. Es wurden aufbauend auf den Anforderungen, die im elatePortal verarbeitet wurden, weitere lehrstuhlspezifische Anwendungsfälle für Testwerkzeuge beleuchtet und daraus Anforderungen entwickelt, die diese Fälle beschreiben. Die Literaturrecherche ergab, dass eine adäquate Architektur für eine Integration von verschiedenen Aufgabentypen nach aktuellem Erkenntnisstand auf den Prinzipien der Serviceorientierung basieren sollte. Diese Erkenntnisse mündeten in wesentliche Erweiterungen des bestehenden Aufgabenframeworks des elatePortal.

Aufbauend auf dieser Architektur wurden drei neue Aufgabentypen entworfen und implementiert: Zeichenaufgaben und Programmieraufgaben für Autotool bzw. Java™. Diese demonstrieren die Verwendung der geschaffenen Schnittstellen zur Einbindung von sowohl neuen als auch bestehenden Testwerkzeugen als Aufgabentypen. Darüber hinaus decken Sie konkrete vorhandene Anwendungsfälle ab und werden teilweise schon in Prüfungen eingesetzt bzw. für einen Einsatz im Übungsbetrieb evaluiert.

Die entstandene Architektur erlaubt den Einsatz des Aufgabenframeworks des elatePortals auf einer wesentlicher breiteren Basis und für bedeutend mehr Anwendungsfälle als bisher.

5.1 Ausblick

5.1.1 Weitere Einsatzszenarien für OSGi

Eine generelle Einführung eines OSGi-Frameworks als Plattform für das Binden der einzelnen Komponenten des elatePortal zur Startzeit ist wegen der Vielzahl der im Kapitel 3.4 vorgestellten Funktionen von OSGi erstrebenswert. Wegen der bisherigen Verwendung von Spring[29] bietet sich für die Migration die Verwendung des Projekts „Spring Dynamic Modules“ an. Es verbindet die Möglichkeiten des Spring-Frameworks mit der dynamischen Laufzeitumgebung von OSGi. Die daraus resultierende Erweiterung der Konfigurierbarkeit der Plattform stellt eine technologisch hochinteressante Ergänzung der produktlinienorientierten Entwicklung des elatePortals dar. Die Austauschbarkeit von einzelnen Komponenten und Services wäre besonders für die Anforderung nach hoher Verfügbarkeit einer

[29] http://www.springframework.org/

solchen Plattform interessant. Die Verteilung von Services auf verschiedene Computer ermöglicht neben einer erhöhten Sicherheit (z.B. durch Bereitstellung eines Services, der Prüfungsdaten verwaltet auf dem Rechner des Dozenten statt auf dem Server) auch Möglichkeiten für eine bessere Skalierbarkeit. Weitere Untersuchungen zum Thema „Clustering von OSGi-Services“ wären hier interessant.

5.1.2 Verbesserung der Java™-Aufgaben

Die Implementierung der Java™-Aufgaben ist in der vorliegenden Form nur bedingt den tatsächlichen Anforderungen angemessen. Ein großes Problem ist etwa das unerwünschte Duplizieren und Austauschen von Lösungen zwischen Studenten. Wünschenswert wären also Erweiterungen zur Parametrisierung der Aufgabenstellungen, analog zu den Templates in QTI. Eine weitere Möglichkeit ist die Verwendung von Werkzeugen zur Plagiatskontrolle, die halbautomatisch syntaktisch äquivalente Lösungen erkennt. Auch der Einsatz von Werkzeugen wie etwa CheckStyle zur Sicherstellung einer gewissen Mindestnorm der Lösungen wäre eine interessante und praxisrelevante Ergänzung dieses Aufgabentyps.

A Fragebogen für die Erfassung des Ist-Zustandes von Übungen und Prüfungen

A.1 Übungsbetrieb

- Verwenden Sie begleitend zu Vorlesungen Übungsaufgaben?
- Wie werden diese Aufgaben gestellt und ausgegeben?
- In welcher Form werden die Lösungen eingereicht?
- Werden die Aufgaben korrigiert? In welcher Form werden die Korrekturen zurückgegeben?
- Welche Aufgabentypen verwenden Sie?
- Ließen sich die Fragestellungen mit MC-, Zuordnungs- oder Lückentextaufgaben ausdrücken?
- Ließen sich die Lösungen der Aufgaben formell spezifizieren, so daß eine automatische Korrektur denkbar wäre?
- Welche Aufgabentypen sind spezifisch für Ihren Fachbereich? (Beweistextaufgaben in der Mathematik, Programmieraufgaben in der Informatik, Zeichnen von Diagrammen etc.)?
- Würden Sie eine entsprechend angepasste technische Unterstützung des Übungsbetriebes in Ihrem Fachbereich einsetzen? Wenn nicht, welche Anforderungen sprächen dagegen?
- Gibt es Software, die Sie in der Lehre oder im Übungsbetrieb einsetzen? Auf welche Weise unterstützt diese den Übungsbetrieb? Hat diese Software Einfluss auf den Aufwand der Aufgabenerstellung bis -korrektur?

A.2 Prüfungen

- Wie organisieren Sie Ihre Prüfungsabläufe? Termine, Aufgabenerstellung, Korrektur?
- Wie hoch schätzen Sie den aktuellen Aufwand für die genannten Punkte, speziell für die Korrektur?
- Lassen sich die Aufgabenstellungen für spätere Prüfungen weiterverwenden?

- Sind die Aufgabestellungen parametrisierbar?
- Wie werden die Prüfungsergebnisse weiterverarbeitet und veröffentlicht? Welche elektronische Unterstützung ist dafür vorhanden?
- Lassen sich die Prüfungsfragen analog zu Übungsaufgaben mit MC-, Zuordungs- oder anderen automatisch korrigierbaren Fragetypen formulieren?
- Welche Bedingungen müßten erfüllt sein, damit Sie Prüfungen am PC durchführen würden?
- Worin sehen Sie Vor- und Nachteile einer rein elektronischen Prüfung?
- Hätten Sie Zugang zu ausreichenden technischen Ressourcen, um eine elektronische Prüfung durchführen zu können? Würden Sie eine Prüfung verteilt auf verschiedene Termine durchführen?

B elatePortal - Weitere Diagramme

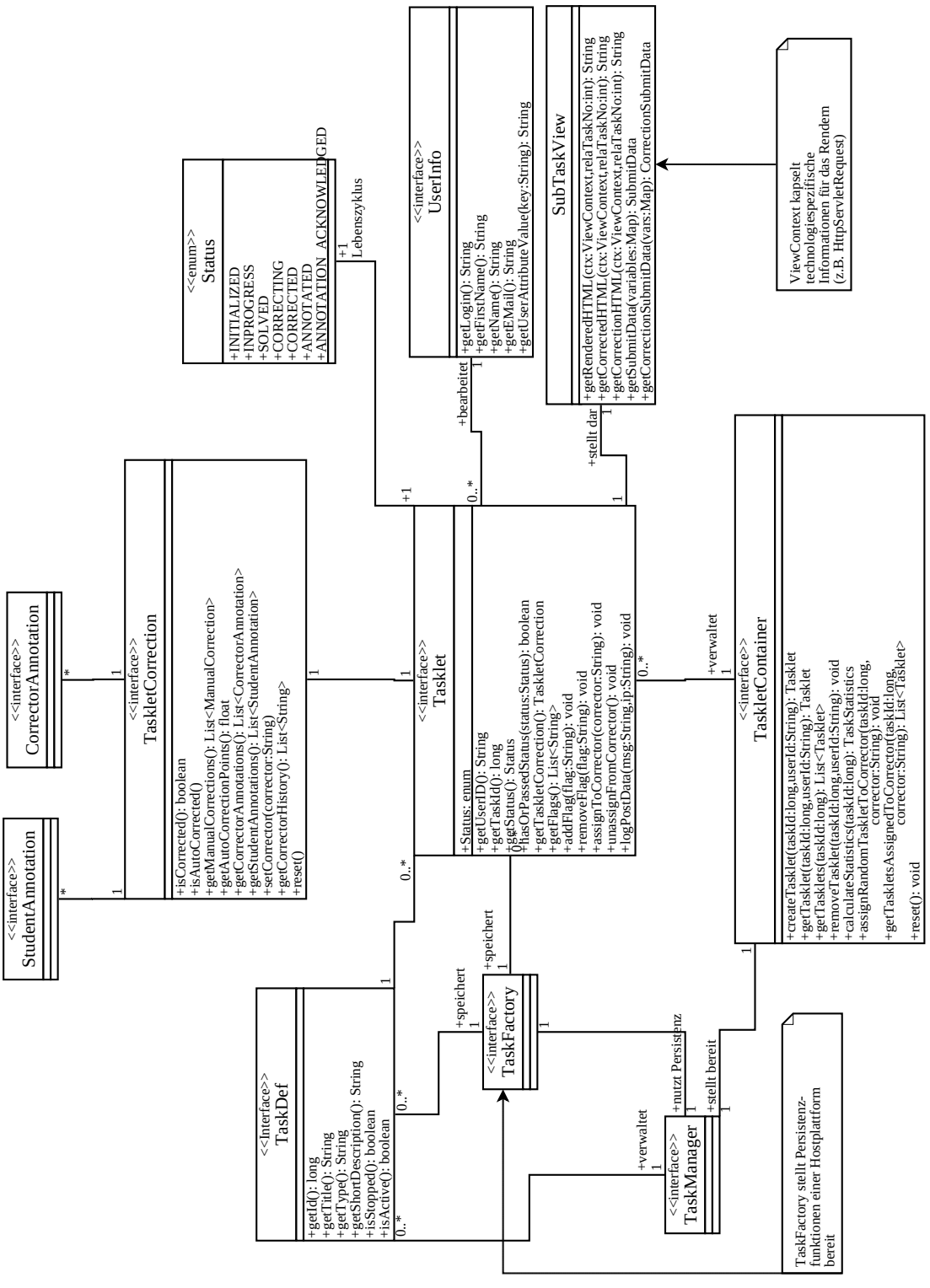


Abbildung 23: Ausschnitt der Gesamtübersicht der API

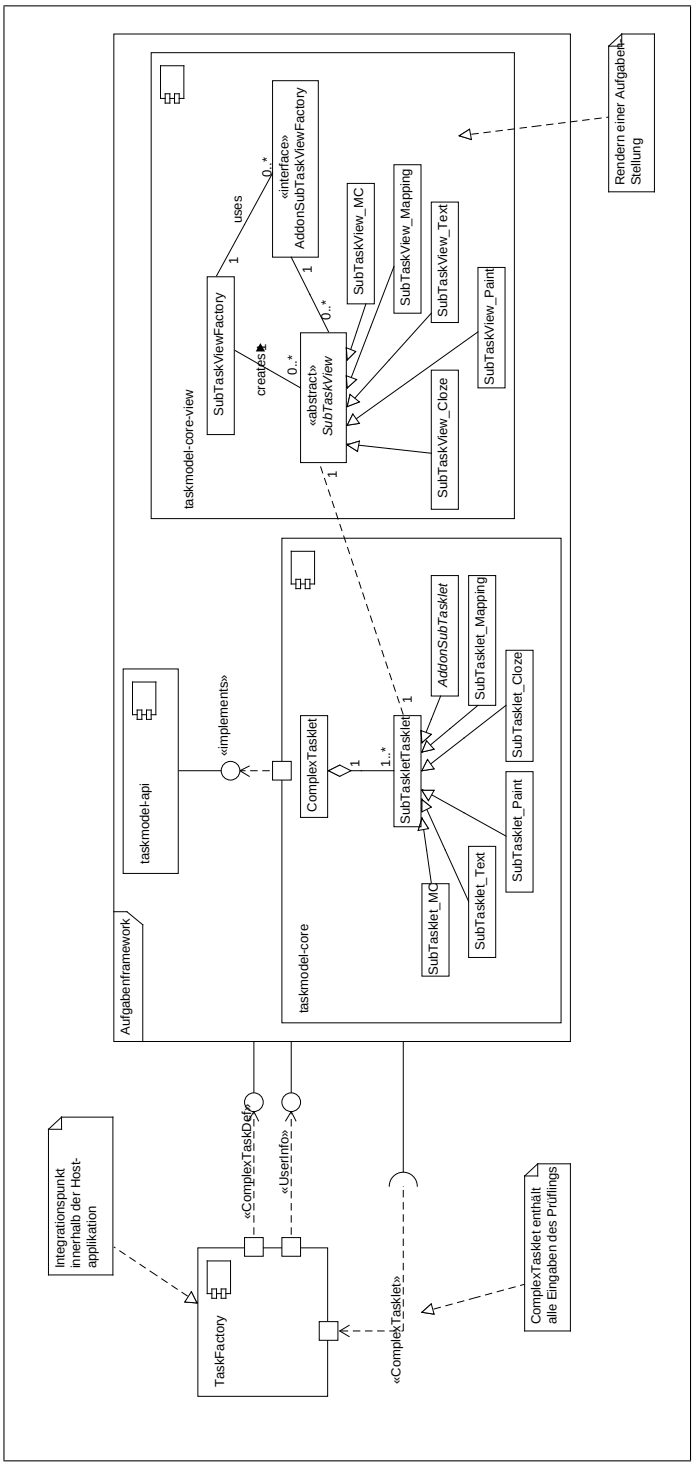


Abbildung 24: Gesamtübersicht des Aufgabenframeworks

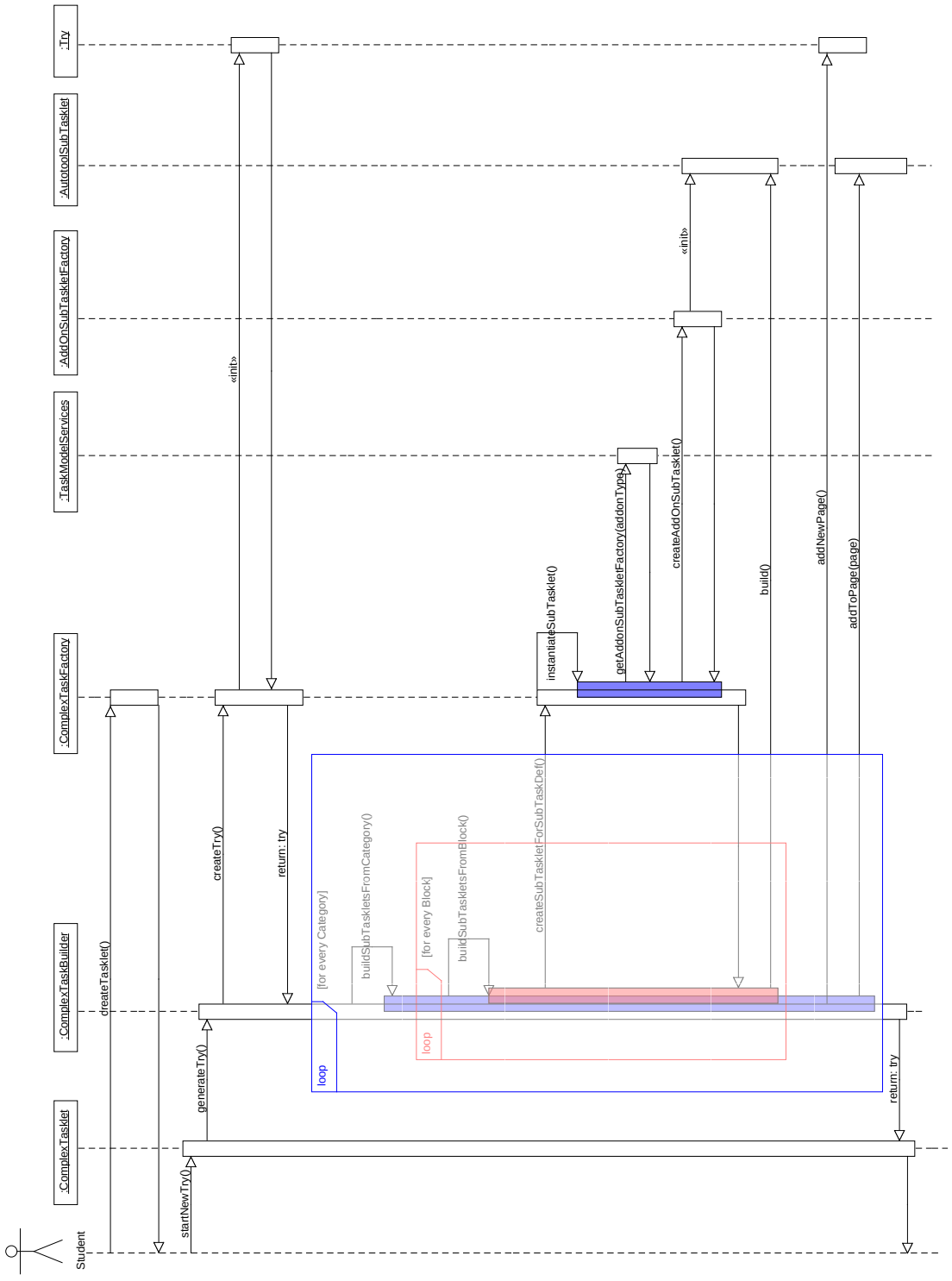


Abbildung 25: Sequenzdiagramm „Bearbeitungsversuch“am Beispiel einer Autotoolaufgabe

C Beispielkommunikation mit Autotool

Ein Beispiel für den Aufruf der Methode `autotool.grade` zur Bewertung eines Lösungsversuches. Zu sehen ist sowohl der Aufruf als auch die Antwort. Aus Übersichtsgründen wurden die ausführlichen Textelemente entfernt.

```
<methodCall>
    <methodName>autotool.grade</methodName>
    <params>
      <param>
        <value><struct><member>
              <name>contents</name>
              <value>Equiv-Quiz</value>
            </member></struct></value>
      </param>
      <param>
        <value><struct><member>
              <name>contents</name>
              <value><struct><member>
                    <name>contents</name>
                    <value>
                    ...
                    von Autotool zu bewertendes Konstrukt
                    ...
                    </value>
                  </member>
                  <member>
                    <name>tag</name>
                    <value>Equiv</value>
                  </member>
                </struct></value></member>
            <member>
              <name>signature</name>
              <value>-448176354758034</value>
            </member>
          </struct></value>
      </param>
      <param>
        <value><struct><member>
              <name>contents</name>
              <value>[ [ ( 1 , 2 , 'a' ) ,
               ( 3 , 4 , 'b' ) ] ,
```

```
            [ ( 1 , 4 , 'a' ) ] ]</value>
        </member></struct></value>
    </param>
  </params>
</methodCall>
<methodResponse>
  <params>
    <param>
      <value><struct><member>
           <name>contents</name>
           <value><struct><member>
                 <name>first</name>
                 <value>
                   <boolean>0</boolean>
                 </value></member>
               <member>
                 <name>second</name>
                 <value>
                   <double>0.0</double>
                 </value></member></struct></value></member>
         <member>
           <name>documentation</name>
           <value>
             <string>
             ...
             ausführliche Dokumentation der Bewertung durch
                Autotool
             ...
             </string>
           </value></member></struct></value>
    </param>
  </params>
</methodResponse>
```

D Darstellungen der neuen Aufgabentypen

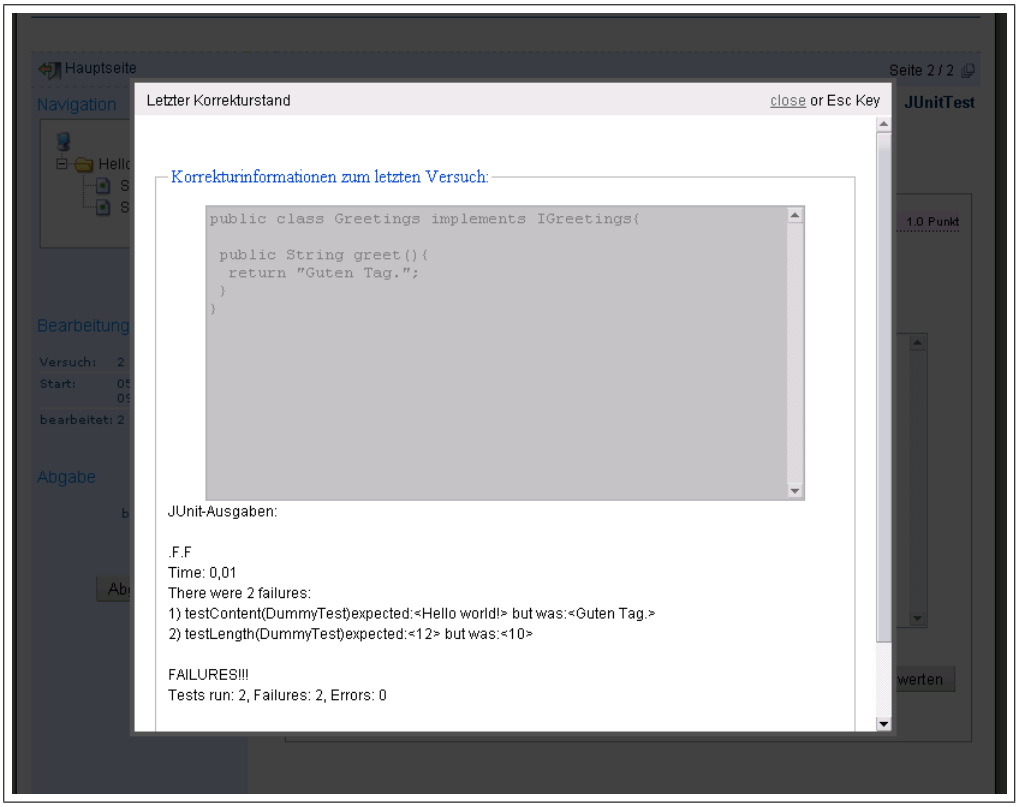


Abbildung 26: Feedback der Javabeispielaufgabe

Abbildung 27: Präsentation des Zeichenaufgabenapplets

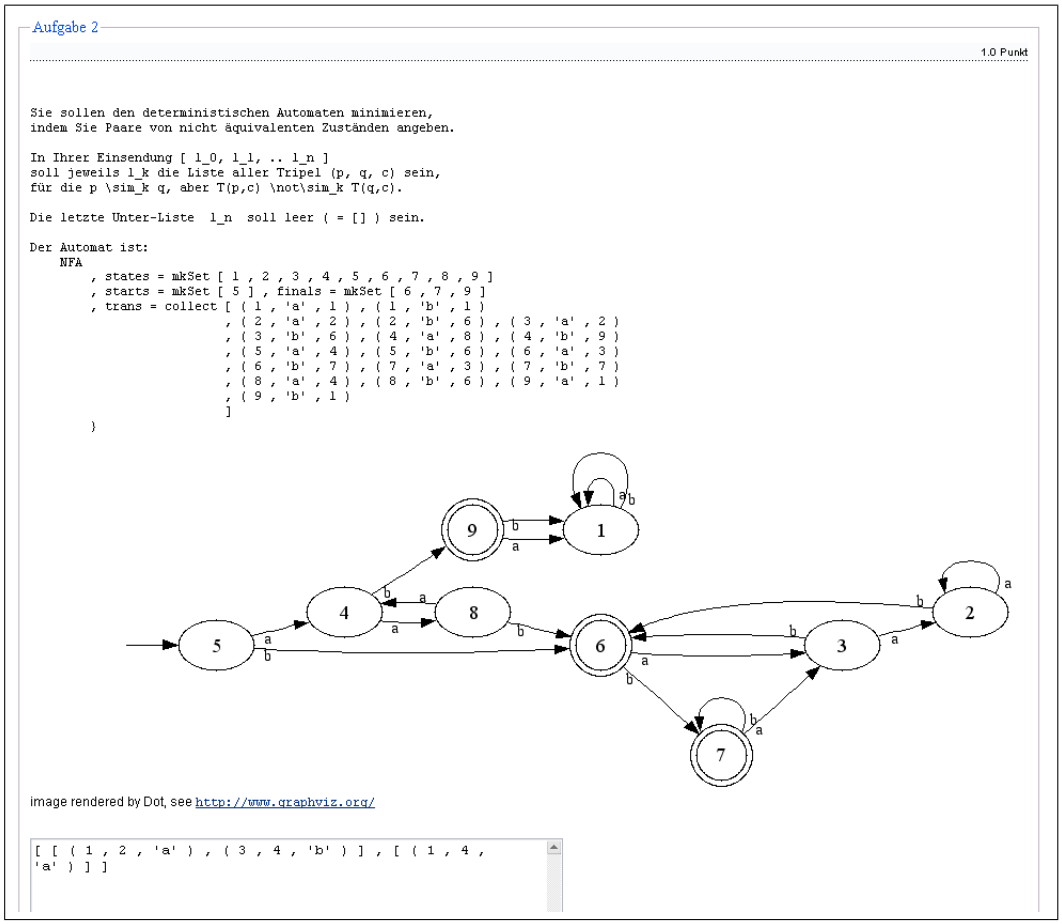


Abbildung 28: Präsentation einer Autotoolaufgabe

Abkürzungen

API	application programming interface. 55
CDC	Connected Device Configuration. 42
CMS	Content Management System. 14
CORBA	Common Object Request Broker Architecture. 38
FREMA	Framework Reference Model for Assessment. 29, 30, 38
IAF	IMS Abstract Framework. 26, 27, 29, 31, 38
IMS	Instructional Management Systems. 26, 27
IPC	inter-process communication. 41
JDK	Java Development Kit. 71
JISC	Joint Information Systems Committee. 29
JVM	Java Virtual Machine. 40, 41
LCMS	Learning Content Management System. 13
OSGi	Open Services Gateway initiative. 25, 40, 82
QTI	Question & Test Interoperability Specification. 11, 26, 27, 33, 50
RMI	Remote Method Invocation. 49, 62
SAP	Service Access Point. 27
SLP	Service location protocoll. 49, 62
SOA	Service Oriented Architecture. 25, 38
SOAP	Simple Object Access Protocol. 29
SPL	Software Product Line. 55
SSO	Single Sign On. 27
UPnP	Universal plug and play. 48

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[al01] ET AL, Marko R. *In2Math - Interaktive Mathematik- und Informatikgrundausbildung*. http://www.mathematik.hu-berlin.de/~in2math/; besucht am 27.08.2007. 2001

[All03] ALLIANCE, Osgi: *OSGi Service Platform: The OSGi Alliance*. 1. Aufl. IOS Press,US, 2003. – 604 S

[All04] ALLIANCE, OSGi. *Listeners Considered Harmful: The „Whiteboard“ Pattern*. www.osgi.org/documents/osgi_technology/whiteboard.pdf; besucht am 15.09.2007. 2004

[Ber07a] BERGER, Thorsten. *Das elatePortal*. http://elateportal.de/; besucht am 14.08.2007. 2007

[Ber07b] BERGER, Thorsten: *Softwareproduktlinien-Entwicklung - Domain Engineering: Forschungsstand, Probleme und Lösungsansätze*, Universität Leipzig, Diplomarbeit, 2007

[BMMW04] BLINCO, Kerry; MASON, Jon; MCLEAN, Neil; WILSON, Scott: Trends and Issues in E-learning Infrastructure Development. In: *Proceedings of alt-i-lab 2004, IMS Global Learning Consortium*, 2004

[BRS06] BÖHME, T.; RAHM, E.; SOSNA, D.: LOTS - Online-Training an der Universität Leipzig. In: *Workshop on e-Learning 2006, HTWK Leipzig*, 2006

[DMW+06] DAVIS, Hugh; MILLARD, David E.; WILLS, Gary; GILBERT, Lester; HOWARD, Yvonne; SHERRATT, Robert; JEYES, Steve; SCLATER, Niall; TULLOCH, Iain; YOUNG, Rowin. *FREMA final report*. http://www.jisc.ac.uk/media/documents/programmes/elearningframework/fremafinalreport_v6.pdf, besucht am 22.02.2008. 2006

[Eck05] ECKARDT, Jens: *Konzeption und Implementierung eines offenen, ontologiebasierten Übungssystems*, Universität Leipzig, Diplomarbeit, 2005

[ES98] UND E. SEKERINSKI, L. M.: A Study of the Fragile Base Class Problem. In: *Lecture Notes in Computer Science 1445, Springer-Verlag*, 1998, S. 355–382

[GHJV94] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John M.: *Design Patterns: Elements of Reusable Object-Oriented*

Software (Addison-Wesley Professional Computing Series). Addison-Wesley Professional, 1994. – ISBN 0201633612

[GJSB05] Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad: *Java(TM) Language Specification, The (3rd Edition) (The Java Series)*. 3rd edition. Prentice Hall PTR, 2005. – ISBN 0321246780

[GPVD99] Guttman, E.; Perkins, C.; Veizades, J.; Day, M.: Service Location Protocol, Version 2 / Internet Engineering Task Force. 1999 (2608). – RFC. – 54 S

[IBM05] IBM. *IBM SOA Foundation - An Architectural Introduction and Overview*. http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/; besucht am 22.10.2007. 2005

[IBM06] IBM. *JSR 291: Dynamic Component Support for Java SE*. http://jcp.org/en/jsr/detail?id=291; besucht am 04.06.2007. 2006

[IMS03a] IMS Global Learning Consortium: *Abstract Framework*. 1.0. 2003

[IMS03b] IMS Global Learning Consortium: *Abstract Framework: Applications, Services, and Components*. 1.0. 2003

[IMS06] IMS Global Learning Consortium: *IMS Question and Test Interoperability Assessment Test, Section, and Item Information Model*. v2.1. 2006

[Kön05] König, T.: *Erstellung einer in das System eClaus integrierten Plagiaterkennung für abgegebene Programme*, Universität Stuttgart, Diplomarbeit, 2005

[Kön07] König, Dierk: *Groovy in action*. 1. Aufl. Manning Publications Co., 2007. – 694 S

[Laf07] Laforge, Guillaume. *JSR 241: The Groovy Programming Language*. http://jcp.org/en/jsr/detail?id=241;besucht am 18.07.2007. 2007

[LG03] Li Gong, Mary D.: *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*. 2nd Edition. Prentice Hall PTR, 2003. – ISBN 0201787911

[LS07] Lehrstuhl Softwaresysteme, Universität P. *Praktomat Homepage und Quelltexte*. http://www.fim.uni-passau.de/de/fim/fakultaet/lehrstuehle/softwaresysteme/forschung/praktomat.html; besucht am 27.08.2007. 2007

[NK05] Niemeyer, Pat; Knudsen, Jonathan: *Learning Java*. 2005. – ISBN 0596008732

[OAS07] OASIS. *Reference Model for Service Oriented Architecture 1.0*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm; besucht am 22.10.2007. 2007

[PBL05] Pohl, Klaus; Böckle, Günter; van der Linden, Frank J.: *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005. – ISBN 3540243720

[PDET07] Prof. Dr. Erwin Tschirner, Prof. Dr. Gerhard H. *Hochschulsprachtest*. http://www.uni-leipzig.de/hochschulsprachtest/index.html; besucht am 14.09.2007. 2007

[RAR07] Rellermeyer, Jan S.; Alonso, Gustavo; Roscoe, Timothy: R-OSGi: Distributed Applications through Software Modularization. In: *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference*, 2007

[SR03] Sosna, D.; Rahm, E.: Web-basiertes SQL-Training im Bildungsportal Sachsen. In: *Proc. BTW-Workshop „Datenbanken und E-Learning“, Leipzig, Feb. 2003*, 2003

[Sun05] Sun. *JSR 218: Connected Device Configuration (CDC) 1.1*. http://jcp.org/en/jsr/detail?id=218; besucht am 15.01.2007. 2005

[SUN07] SUN. *Java SE Security Documentation*. http://java.sun.com/javase/technologies/security/; besucht am 12.07.2007. 2007

[Wan07] Wanka, Christian: *Entwicklung einer XML-RPC-Schnittstelle für das Autotool mit clientseitiger Implementierung durch das ZOPE-Produkt AutoI*, HTWK Leipzig, Diplomarbeit, 2007

[WBR04] Wilson, Scott; Blinco, Kerry; Rehak, Daniel: Service-Oriented Frameworks: Modelling the infrastructure for the next generation of e-Learning Systems. In: *Proceedings of alt-i-lab 2004, IMS Global Learning Consortium*, 2004

[Wie06] Wielicki, Tom: Integrity of Online Testing in E-Learning: Empirical Study. In: *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)*, 2006

[Wil06a] Wills, Dr. G.: R2Q2: Rendering and Reponses Processing for QTIv2 Question Types. In: *Proceedings of 10th International CAA Conference*, 2006

[wil06b] WILLS, Dr. G. *R2Q2: Rendering and Response processing services for QTIv2 questions*. http://www.r2q2.ecs.soton.ac.uk/overview/; besucht am 27.08.2007. 2006

[Win99] WINER, Dave. *XML-RPC Specification*. http://www.xmlrpc.com/spec; besucht am 04.08.2007. 1999

[WR02] WALDMANN, Johannes; RAHN, Mirko: The Leipzig autotool System for Grading Student Homework. In: *International Workshop on Functional and Declarative Programming in Education, Technical Report No. 0210*, 2002, S. 155

[YHDN05] Y, Davies WM H.; HC, Davis; DE, Millard; N, Sclater: Aggregating Assessment Tools in a Service Orientated Architecture. In: *Proceedings of 9th International CAA Conference*, 2005

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe

- - - - - - - - - - - - - - - - - - - -

Leipzig, den 17. März 2007 Steffen Dienst