Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BM-2006-31

Bachelorarbeit

im Studiengang „Angewandte Informatik“

Testen der Web Services eines Lehrevaluationssystems

Dennis Neumann

am Institut für
Informatik

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

30. September 2006

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel.	+49 (5 51) 39-1 44 14
Fax	+49 (5 51) 39-1 44 15
Email	office@informatik.uni-goettingen.de
WWW	www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 30. September 2006

Danksagungen

Ich möchte mich besonders bei Edith Werner bedanken, die mich bei der Bearbeitung der Bachelorarbeit immer tatkräftig unterstützte und immer nützliche Tipps parat hatte. Außerdem bedanke ich mich bei Helmut Neukirchen, der mir bei technischen Problemen zur Hilfe stand.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADB	Authorisierungsdatenbank/Authorization Database
ANT	Another Neat Tool
AXIS	Apache Extensible Interaction System
BASH	Bourne Again Schell
FDB	Fragebogendatenbank/Forms Database
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JSP	Java ServerPages
MTC	Main Test Component
MVC	Model View Controller
ONEVA	Online Evaluationssystem
POP	Post Office Protocol
SMTP	Simple Mail Transfer Protocol
SOAP	früher: Simple Object Access Protocol/Service Oriented Architecture Protocol; heute: keine Abkürzung
SQL	Structured Query Language
SUT	System Under Test
TTCN-3	Testing and Test Control Notation 3
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XML	Extensible Markup Language

1. Einleitung

Das Testen von Software wird insbesondere bei größeren Projekten immer wichtiger. Es ist heute üblich, dass ungefähr genauso viel Zeit und Aufwand in die Tests wie in die Implementierung einer Anwendung investiert wird. Ausführliche Tests sollten bereits bei der Planung eines Projekts berücksichtigt werden. Wenn man sich Modelle ansieht, die zur Herstellung moderner Software verwendet werden, erkennt man, dass Programmtests nicht unterschätzt werden dürfen. Das prominenteste Beispiel ist das allgemeine V-Modell, bei dem bereits einzelne Klassen mit dem Komponententest getestet werden und verschiedene Testmethoden für alle Entwicklungsphasen vorgesehen sind.

Diese Arbeit basiert auf den beiden Bachelorarbeiten [11] und [12]. In [11] wurde das Online-Evaluationssystem entworfen und in [12] komplett mit den zugehörigen Web Services implementiert. Diese Arbeit beschäftigt sich mit einigen Erweiterungen der Anwendung und dem Testen der darin integrierten Web Services.

Zunächst werden theoretische Grundlagen zu Web Services und zum Softwaretesten gegeben. Die benutzte Testsprache und das Testwerkzeug zur Ausführung der Tests werden beschrieben. Danach wird die Funktionsweise und die innere Struktur des Online-Evaluationssystems kurz erläutert (für Einzelheiten sollten [11] und [12] hinzugezogen werden).

Nach diesen Einführungskapiteln werden die praktischen Tätigkeiten beschrieben, die während der Arbeit durchgeführt wurden. Zuerst werden die Funktionserweiterungen der Applikation ausführlich erläutert. Danach wird auf die angewendeten Methoden des Testens und die konkrete Umsetzung der Testfälle eingegangen. Schließlich werden die Ergebnisse der Testarbeiten präsentiert und erläutert.

2. Grundlagen

2.1 Web Services

„Bei Web Services handelt es sich um Softwarebausteine, die auf verschiedenen Netzwerkrechnern laufen und über das Internet zu einer Anwendung verbunden werden“ [1, S.2].

Diese kurze Definition sagt schon Vieles über die Web Services aus. Ähnlich einem Objekt in objektorientierten Programmiersprachen ist ein Web Service ein Programmteil, in dem Operationen und Daten gekapselt werden. Im Unterschied zu Objekten in einer bestimmten Programmiersprache sind Web Services sprach- und plattformunabhängig, das heißt sie können prinzipiell in jeder Sprache programmiert und auf jedem Betriebssystem ausgeführt werden [2, S. 500-502]. Sie sind außerdem speziell mit dem Ziel entwickelt worden, von überall im Internet leicht über einen URL (*Uniform Ressource Locator*, eindeutige Lokalisationsnummer) zugreifbar zu sein. Auf diese Weise lassen sich komplexe Webanwendungen leicht auf mehrere Rechner verteilen, wobei die einzelnen Web Services als abgeschlossene Einheiten auch von anderen Anwendungen benutzt werden können.

Web Services können mit einfachen Internetseiten-Abfragen verglichen werden. Auch hier wird eine Anfrage (*request*) an einen Web Server geschickt, woraufhin eine Datei als Antwort (*responce*) zurückgesendet wird. Im Falle eines Seitenaufrufs beinhaltet die Antwort eine HTML-Datei, die in einem Browser für einen Benutzer dargestellt wird. Dagegen kann ein Programm mit einer HTML-Datei nichts anfangen, denn es sind keine semantischen Informationen darin enthalten, die eine automatische Verarbeitung ermöglichen würden. Stattdessen antwortet der Server mit einer XML-Datei, die ein wohldefiniertes und standardisiertes Format aufweist. Diese Datei kann leicht maschinell analysiert (*parsing*) und die zurückgegebenen Daten können weiterverarbeitet werden. Web Services erweitern also die Funktionen eines Web Servers insofern, dass im Internet bereitgestellte Informationen nicht nur von Menschen, sondern auch von Anwendungen benutzt werden können. [2, S.499]

Abbildung 1 zeigt die beiden Instanzen „Browser“ und „Applikation“, welche die Funktionen eines Web Servers nutzen. Hierbei ist noch anzumerken, dass im Falle des Browsers nur die Antwort eine HTML-Datei enthält. Bei einem Web Service-Aufruf werden sowohl in der Anfrage, als auch in der Antwort XML-Dateien ausgetauscht.

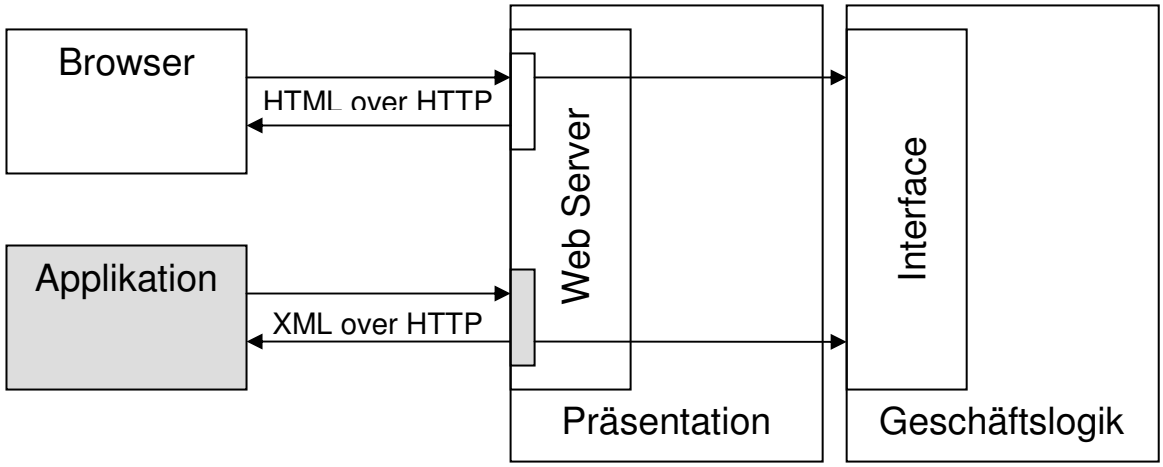


Abbildung 1: Kommunikation eines Browsers und einer Applikation mit einem Web Server

2.1.1 XML (Extensible Markup Language)

XML ist ein Datenformat, mit dem sich auf eine einfache Weise komplexe Datenstrukturen darstellen lassen [3]. In Web Services wird XML benutzt um Nachrichten zu codieren und Schnittstellen zu definieren. Die Nachrichten werden im so genannten SOAP-Protokoll (Simple Object Access Protocol [7]) versendet. Damit das Benutzerprogramm (*client*) ermitteln kann, welche Parameter ein bestimmter Web Service erwartet und welchen Rückgabewert dieser zurückliefert, werden Schnittstellen als WSDL-Dateien (Web Service Definition Language [8]) bereitgestellt. Diese Schnittstellen sind ebenso wie der Web Service selbst über einen URL erreichbar.

Um einen Web Service anhand seiner Schnittstellenbeschreibung zu finden, existieren entsprechende Verzeichnisdienste. Diese werden in UDDI (Universal Description, Discovery and Integration), ebenfalls einem XML-Standard, beschrieben.

2.1.2 SOAP (Simple Object Access Protocol)

SOAP [7] ist ein leichtgewichtiges Protokoll, dessen Zweck der Austausch strukturierter Informationen in einer dezentralisierten verteilten Umgebung ist [4, S. 177]. In der Web Services-Technologie wird es als standardisiertes, auf XML basierendes Nachrichtenaustauschformat benutzt.

Das allgemeine Format von SOAP sieht wie folgt aus [5]:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"">

        <soapenv:Header>
        ...
        </soapenv:Header>

        <soapenv:Body>
        ...
        </soapenv:Body>

</soapenv:Envelope>
```

Die gesamte Nachricht ist somit ein „Briefumschlag“ (*envelope*), der wie ein HTML-Dokument einen Kopf (*header*) und einen Rumpf (*body*) einschließt. Der Nachrichtenkopf dient zur Angabe von so genannten Meta-Informationen, also Mitteilungen darüber, wie die Nachricht verarbeitet werden soll und nicht was zu verarbeiten ist. Hier kann zum Beispiel eine Zugehörigkeit der Nachricht zu einer Transaktion oder die mögliche Verschlüsselung des Rumpfes angegeben werden. Im Nachrichtenrumpf werden alle Daten aufgeführt, die zwischen dem Client und dem Server ausgetauscht werden [4, S.179f].

Ein Beispiel aus [4] zeigt wie eine Flugreservierung als Rumpf einer SOAP-Nachricht aussehen könnte:

```
<env:Body>

    <r:reservierung xmlns:r="http://www.web-air.de/reservierung">
        <r:abflugort>Frankfurt</r:abflugort>
        <r:ankunftsort>Los Angeles</r:ankunftsort>
        <r:abflugdatum>2006-12-14</r:abflugdatum>
        <r:abflugzeit>10:25</r:abflugzeit>
        <r:sitzPraeferenz>Fenster</r:sitzPraeferenz>
    </r:reservierung>

</env:Body>
```

An diesem Beispiel sieht man, dass der Inhalt des Rumpfes komplett benutzerspezifisch ist. Um Verwirrungen zu vermeiden, wird häufig ein eigener Namensraum (*namespace*) definiert. Damit so eine Nachricht vom Web Server (oder genauer: Web Service Container) des Reisebüros empfangen und verarbeitet werden kann, müssen alle benutzerspezifischen Elemente wie „`reservierung`“ und „`abflugort`“ genau definiert worden sein. Das heißt sowohl der Verfasser der Nachricht (z.B. ein Programm auf dem Rechner des Reisenden), als auch der Empfänger müssen die Grammatik der Nachricht verstehen. Diese Grammatik wird in der WSDL- Definition eines Web Service festgelegt (siehe nächstes Kapitel).

SOAP-Nachrichten werden meist über das HTTP-Protokoll (Hypertext Transfer Protocol) übertragen. Unterstützt werden jedoch auch die beiden Email-Protokolle SMTP (Simple Mail Transport Protocol) und POP (Post Office Protocol) sowie FTP (File Transfer Protocol) und einige andere weniger bekannte Protokolle [5].

2.1.3 WSDL (Web Services Definition Language)

Wie bereits kurz erwähnt, stellt eine WSDL-Datei [8] eine plattformunabhängige Schnittstellen-Spezifikation für einen Web Service dar. Sie enthält [2, S.516]:

- die öffentlich verfügbaren Funktionen und deren Signaturen, d.h. die Namen, die erlaubten Übergabeparameter und den Rückgabetyp der Funktionen
- Informationen über die unterstützten Transport-Protokolle, wie HTTP, SMTP usw.
- die URL-Adresse des Web Services
- die Typ-Schemata für den Datenaustausch, vor allem Definitionen eigener komplexer Datentypen

Diese Informationen werden in so genannten Beschreibungselementen codiert [6]. In Abbildung 2 sind die wichtigsten Elemente aufgelistet.

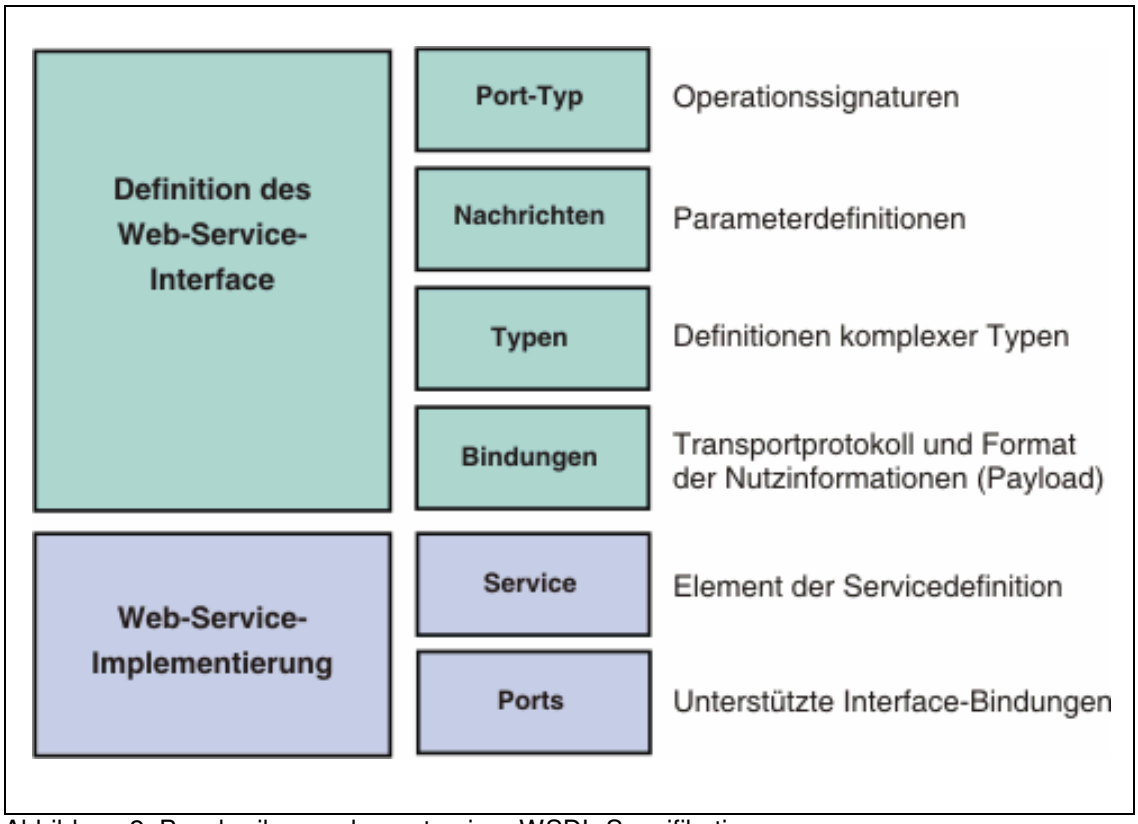


Abbildung 2: Beschreibungselemente einer WSDL-Spezifikation

Ein Port-Typ stellt eine abstrakte Schnittstelle des Web Services dar. Er kapseln eine oder mehrere Nachrichten, die ihrerseits aus Typen zusammengesetzt sind (s. Abbildung 2). In den Nachrichten können entweder XMLSchema-Datentypen oder selbstdefinierte Typen verwendet werden. In den Bindungen werden spezifische Netzwerk-Informationen angegeben. Beispielsweise könnte ein Web Service als ein Prozedurfernaufruf (*remote procedure call*) über das HTTP-Protokoll erreichbar sein.

Ein Service definiert die konkrete Schnittstelle, über die auf den Web Service zugegriffen werden kann. Außer den Ports, welche die Port-Typen konkretisieren, enthält ein Service immer die genaue URL-Adresse des Web Services.

2.1.4 UDDI (Universal Description, Discovery and Integration)

Mit SOAP und WSDL sind nur statische Aufrufe von Web Services möglich. Das heißt der Benutzer muss die genaue URL-Adresse und die Funktionen eines Web Services kennen, um ihn richtig aufrufen zu können. UDDI [13], ein Verzeichnisdienst für Web Services, ermöglicht es Programmen, zur Laufzeit nach Web Services mit bestimmten Kriterien zu suchen und sie zu benutzen.

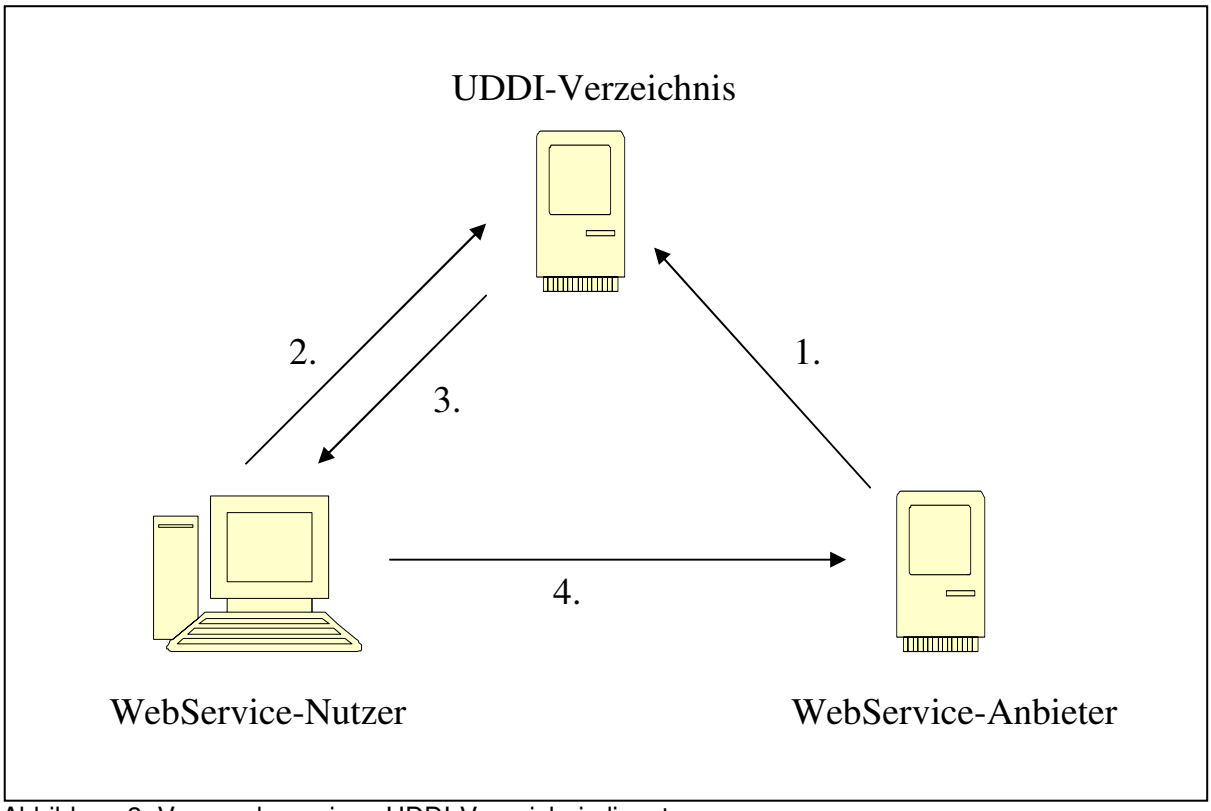


Abbildung 3: Verwendung eines UDDI-Verzeichnisdienstes

Abbildung 3 zeigt wie das UDDI-Verzeichnisdienst vom Nutzer und vom Anbieter eines Web Services verwendet wird:

1. Der Anbieter, meist ein Unternehmen, veröffentlicht einen Web Service im Verzeichnis.
2. Der Nutzer (ein Programm) schlägt im Verzeichnis nach.
3. Der Nutzer erhält als Antwort den URL des gefundenen Web Services.
4. Nun kann der Nutzer mit Hilfe der URL-Adresse den Web Service des Anbieters verwenden.

2.2 Softwaretesten

Die Hauptziele des Softwaretestens sind [vgl. 9, S.9]:

- Es werden Fehlerwirkungen im Programm nachgewiesen.
- Die Qualität des Programms wird bestimmt.
- Das Vertrauen in das Programm soll erhöht werden.

Im Unterschied zur Fehlerkorrektur (*debugging*) werden beim Softwaretesten nur Wirkungen von Fehlern entdeckt und nicht die fehlerverursachenden Defekte im Programm. Die Suche nach diesen Fehlerwirkungen kann meistens nur stichprobenartig erfolgen, da bereits kleine Programme eine sehr große Menge an Eingabe- und Ausgabedaten besitzen [10].

Abbildung 4 zeigt die einzelnen Schritte, die beim Testen einer Anwendung (oder eines Teils der Anwendung) durchgeführt werden müssen.

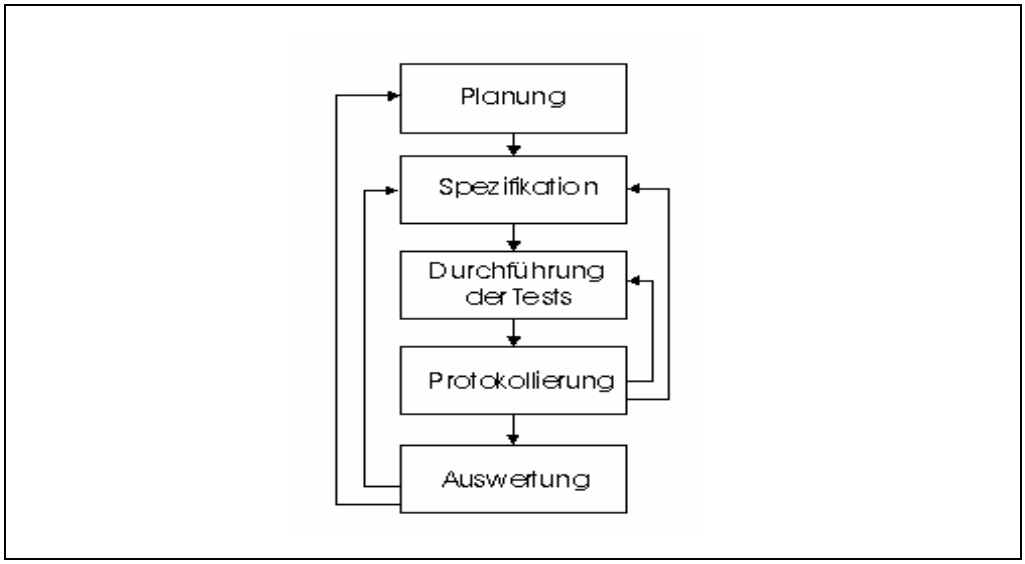


Abbildung 4: Die einzelnen Schritte beim Test einer Software

2.2.1 Testplanung

Vor der eigentlichen Erstellung der Tests sollte in einem Testplan genau festgehalten werden, was und wie getestet werden soll. Das Testobjekt, wie z.B. ein Programm oder nur eine Klasse, ist zu benennen und zu spezifizieren. Die Zielsetzung, die beim Testen angestrebt wird, ist ebenfalls anzugeben.

Des Weiteren muss ein zum Testobjekt passendes Testverfahren ausgewählt werden. Hier stehen grundsätzlich zwei Verfahren zur Auswahl [9]:
- statischer Test: Die Quellcode-Dateien und andere Dokumente, die zu einer Anwendung gehören, werden (in der Regel werkzeuggestützt) systematisch untersucht.
- dynamischer Test: Das Verhalten des Testobjekts wird während einer Ausführung beobachtet und protokolliert.

Die beiden allgemeinen Verfahren lassen sich in speziellere aufteilen. Beim dynamischen Test gibt es beispielsweise die Ausprägungen Blackbox-Test und Whitebox-Test. Beim Blackbox-Test wird nur die Funktion des Testobjekts geprüft, wohingegen der Whitebox-Test die innere Struktur der getesteten Anwendung miteinbezieht.

2.2.2 Testspezifikation

Die Spezifikation der Testfälle (logische Einheiten bei einem Testdurchlauf) erfolgt nach einem gewählten Schema (Testmethode). Beim Blackbox-Test gibt es beispielsweise die Äquivalenzklassenbildung oder die zustandsbezogene Methode [vgl. 9, Kap. 5.1]. Auf der Basis einer solchen Testmethode und der Spezifikation des Testobjektes werden zunächst logische Testfälle erstellt, die in Tabellen oder als Diagramme dargestellt werden. Die logischen Testfälle sind im Vergleich zu den konkreten Testfällen noch abstrakt gehalten. So werden z.B. keine konkreten Eingabewerte angegeben, sondern nur Wertebereiche.

Um die erwarteten Ergebnisse auf bestimmte Eingaben zu erhalten, wird ein Testorakel „befragt“ [9, S. 24]. Ein solches „Orakel“ kann die Testobjektspezifikation oder auch der Programmcode sein. Der Tester muss daraus ermitteln, welche Wirkungen durch welche Eingabedaten verursacht werden. Außer der erwarteten Ausgabedaten gehören auch persistente Datenänderungen in einer Datenbank oder in Dateien dazu.

Außer den gültigen Testfällen, die das „Orakel“ liefert, sollten auch Testfälle mit Eingabewerten erstellt werden, die möglicherweise eine Fehlersituation verursachen und für die es keine Ausnahmebehandlungen (*exception handling*) existieren.

Die konkreten Testfälle, die die tatsächlichen Eingabeparameter enthalten, können gleich in einer Programmiersprache implementiert werden. Ähnliche Testfälle sollten aus Übersichtsgründen zu Testszenarien gruppiert werden.

2.2.3 Durchführung und Auswertung der Tests

Die Ausführung der Tests kann manuell, sollte jedoch mit Hilfe geeigneter Werkzeuge durchgeführt werden. Gegebenenfalls muss dafür eine Testumgebung eingerichtet werden. Die Testumgebung umfasst alle Software- und Hardware-Komponenten, die zur Testdurchführung nötig sind. Die wichtigste Komponente, die oft zusätzlich zu den Testfällen implementiert werden muss, ist der Testadapter. Dieser bildet eine Schnittstelle zwischen den Testfällen und dem Testobjekt. Falls das Testobjekt und die Testfälle in verschiedenen Programmiersprachen geschrieben sind, sorgt der Testadapter für die Transformation der Datentypen, die ausgetauscht werden.

Damit die Testergebnisse auch für unbeteiligte Personen nachvollziehbar sind, müssen die Testdurchläufe protokolliert werden [9, S. 27ff]. Wenn die Tests mit Hilfe spezieller Werkzeuge durchgeführt werden, werden meist automatisch Protokolle erstellt, die verwendet werden können. Bei der manuellen Testdurchführung sollten nicht nur die möglicherweise gefundenen Fehlerwirkungen, sondern auch alle Testfälle protokolliert werden, die fehlerfrei durchlaufen.

Falls Fehlerwirkungen gefunden werden, sollte genau geprüft werden, ob diese tatsächlich im Testobjekt auftreten. Unterschiedliche Ist- und Soll-Ergebnissen können auch andere Ursachen haben [vgl. 9, S. 27]:

- ungenaue oder falsche Testobjektspezifikation.
- Fehler in der Spezifikation der logischen oder konkreten Testfälle.

- fehlerhafte Testinfrastruktur, z.B. Probleme beim Aufruf des Testobjektes über Netzwerk.
- fehlerhaft implementierter Testadapter, vor Allem bei der Transformation der Datentypen zwischen den Testfällen und dem Testobjekt.

Zum Abschluss der Testarbeiten sollte eine Testauswertung in Form eines kurzen Berichtes erstellt werden, die alle Ergebnisse basierend auf den Testprotokollen zusammenfasst.

2.3 Blackbox-Test

Das Prinzip des Blackbox-Tests ist sehr einfach: In Abhängigkeit von Eingabedaten werden die Ausgabedaten auf ihre Richtigkeit geprüft (Abbildung 5). Dabei ist nur wichtig, was ausgegeben wird und nicht wie das Ergebnis zustande kommt. Der Programmcode ist unbekannt, das Testobjekt wird also als eine schwarze Kiste angesehen, in die man nicht hineinsehen kann [vgl. 9, S. 108]. Die Position des Beobachters (PoO, *Point of Observation*) befindet sich außerhalb der Kiste. Die Steuerung des Testobjektes (PoC, *Point of Control*) ist ebenfalls nur von außen durch die Variierung der Eingabeparameter möglich.

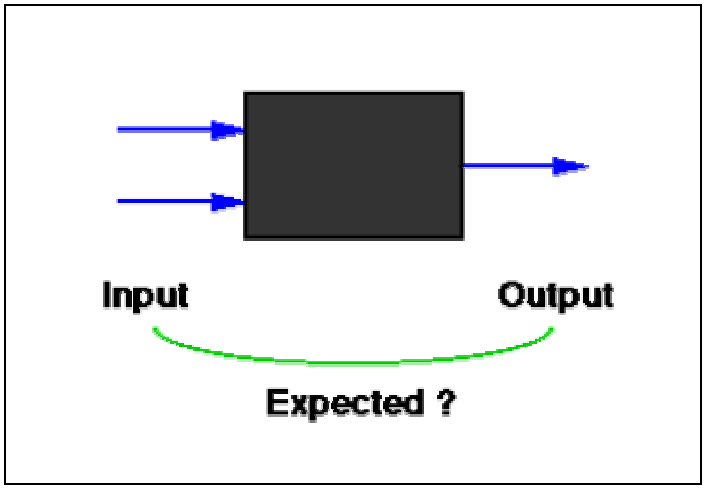


Abbildung 5: Der Blackbox-Test

2.3.1 Zustandsbezogener Test

Der zustandsbezogene Test ist eine Variante des Blackbox-Tests, die immer dann angewendet wird, wenn ein interner Zustand im Testobjekt von Bedeutung ist. Der Zustand könnte beispielsweise in externen Dateien, in einer Datenbank oder auch nur zur Laufzeit im Testobjekt gespeichert werden. Der zustandsbezogene Test sollte also dann durchgeführt werden, wenn die Ausgabedaten des Testobjektes nicht nur von den Eingabedaten, sondern auch vom internen Zustand abhängig sind. Idealerweise wird in diesem Fall jede Operation des Testobjekts bei jedem möglichen Zustand getestet. In der Realität ist dies jedoch aus Komplexitätsgründen fast nie möglich. Deshalb sollten außer den „alltäglichen“ Kombinationen „Zustand – Operationsaufruf“ (d.h. denjenigen, die in der Spezifikation des Testobjekts vorgesehen sind), zumindest auch potentiell kritische Kombinationen getestet

werden. Dazu gehören beispielsweise Operationsaufrufe mit einer leeren Datenbanktabelle oder mit einer schreibgeschützten Datei.

Operationsaufrufe und die dadurch verursachten Zustandsübergänge werden in einem so genannten Übergangsbaum modelliert. Zur Verdeutlichung wird ein Beispiel aus [9] erläutert.

Die Beispielanwendung ist eine Implementierung des Stapels (*stack*). Ein neuer Stapel wird mit dem Kommando `init(max)` erstellt, wobei `max` die maximale Höhe des Stapels angibt. Danach können Elemente durch `push` hinzugefügt und durch `pop` wieder vom Stapel entfernt werden. Außerdem gibt es die Operation `top`, die das oberste Element zurückgibt, ohne es von dem Stapel zu nehmen. Auf einen vollen Stapel können keine Elemente mehr gelegt werden. Falls der Stapel leer ist, kann er schließlich mit Hilfe von `delete` gelöscht werden.

Bei diesem Stapel lassen sich drei verschiedene Zustände identifizieren: „leer“, „gefüllt“ und „voll“. Außerdem werden vollständigkeitshalber die beiden Zustände „initial“ und „gelöscht“ definiert, bei denen der Stapel nicht existiert und die deswegen im Test keine Rolle spielen.

In verschiedenen Stapelzuständen reagieren die Operationen teilweise auf verschiedene Weisen. So geben `pop` und `top` in den Zuständen „gefüllt“ und „voll“ ein Element zurück, im Zustand „leer“ aber nicht. Deshalb müssen Operationsaufrufe in Abhängigkeit vom Zustand durchgeführt werden. In einem Übergangsbaum werden die Zustände als Knoten und die Operationsaufrufe als Kanten abgebildet. Ausführungen von Operationen verursachen Zustandsübergänge, die durch Pfeile angedeutet sind. Abbildung 6 zeigt einen möglichen Baum für das Stapel-Beispiel.

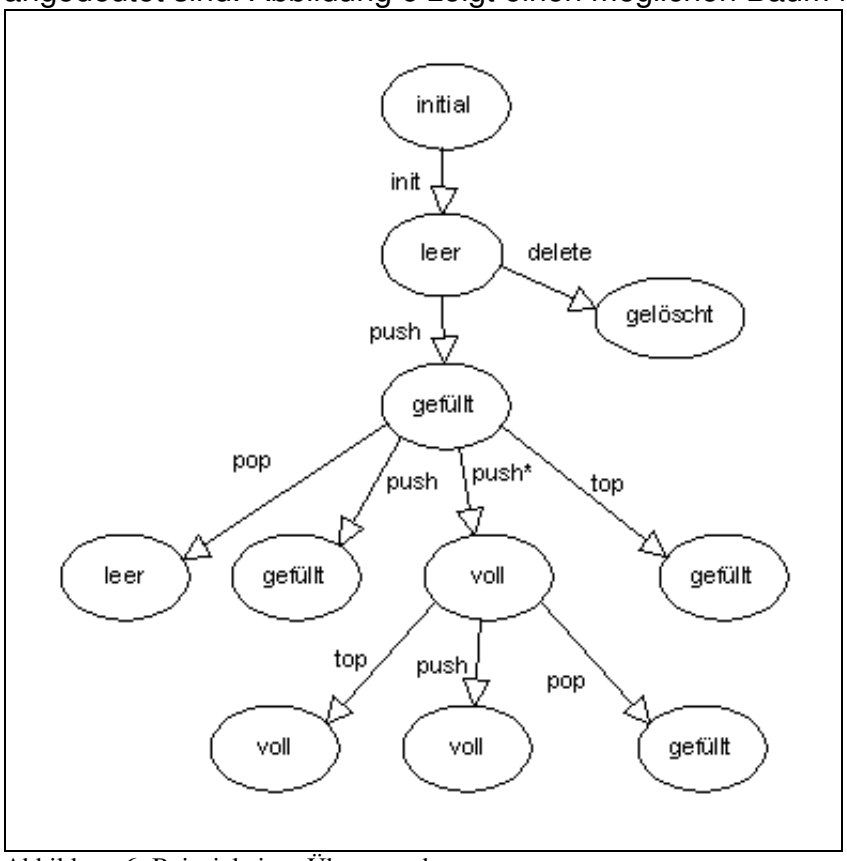


Abbildung 6: Beispiel eines Übergangsbaums

In diesem Übergangsbaum sieht man die erlaubten, d.h. die in der Spezifikation angegebenen Kombinationen „Zustand – Operationsaufruf“. Dies sind die grundsätzlichen Funktionsabläufe, die wahrscheinlich keine Fehler hervorrufen. Zusätzlich dazu müssen auch die möglicherweise zu Fehlerwirkungen führende Abläufe (s. Abbildung 7) getestet werden[1].

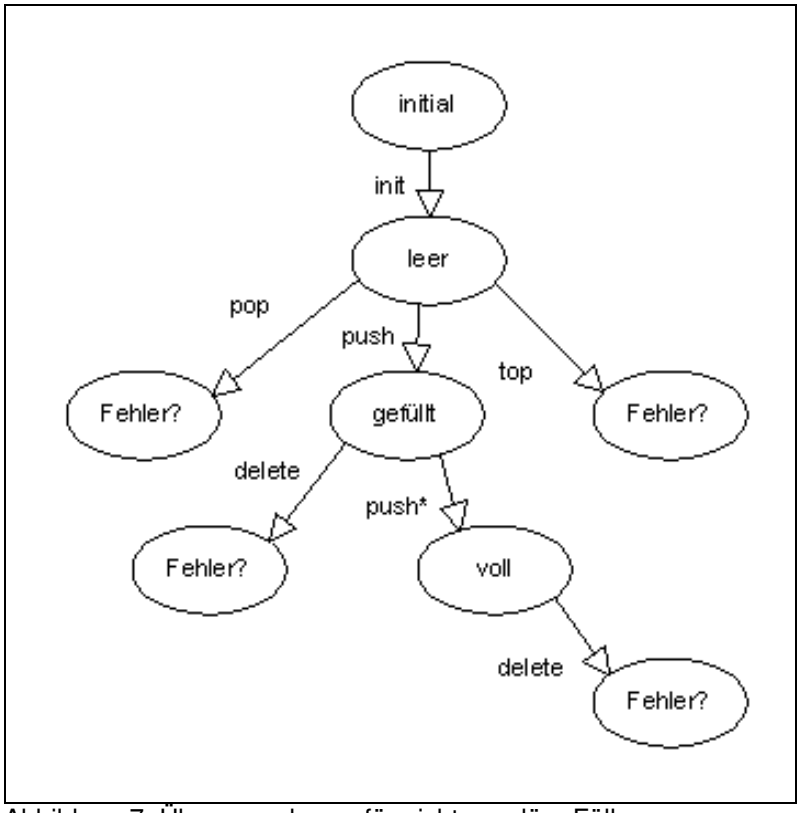


Abbildung 7: Übergangsbaum für nicht-reguläre Fälle

Aus diesen Bäumen können direkt Testfälle ermittelt werden. Jeder Pfad von der Wurzel zu einem Blatt stellt einen Testfall dar. Der Tester muss dafür sorgen, dass vor jeder Testfallausführung der Stapel gelöscht und neu initialisiert wird.

2.3.2 Andere Blackbox-Testmethoden

Wenn das Testobjekt keinen internen Zustand besitzt oder wenn die Operationen des Testobjekts in jedem Zustand gleich reagieren, werden andere Testmethoden angewandt. Die am häufigsten benutzten Methoden sind die Äquivalenzklassenbildung und die Grenzwertanalyse.

Bei der Äquivalenzklassenbildung wird die Menge aller möglichen Eingabewerte in Klassen unterteilt. Diese Äquivalenzklassen beinhalten jeweils Werte, von denen ausgegangen wird, dass sie zum gleichen Ergebnis führen. Der Test nur eines Wertes aus jeder Klasse wird dann als ausreichend angesehen [9, Kap. 5.1.1]. Aus der Spezifikation des Testobjekts werden gültige Äquivalenzklassen hergeleitet.

[1] Aus Übersichtsgründen wurden die spezifizierten und die kritischen Programmabläufe in zwei Bäume aufgeteilt. Eine andere Möglichkeit wäre, alles in einem Baum darzustellen.

Ungültige Klassen enthalten Eingabewerte, die möglicherweise Fehlerwirkungen auslösen.

Die Grenzwertanalyse kann als eine Ergänzung zur Äquivalenzklassenbildung verwendet werden [9, Kap. 5.1.2]. Häufig gibt es Missverständnisse in der Spezifikation, was die Grenzwerte zwischen den Äquivalenzklassen angeht. Beispielsweise ist bei einer Formulierung wie „die Funktion nimmt nur positive Zahlen an“ unklar, ob die Null auch zum akzeptierten Bereich gehört oder nicht. Deswegen sollten Werte, die nahe an solchen Bereichsgrenzen liegen, besonders intensiv getestet werden.

2.4 TTCN-3 als Testsprache

TTCN-3 [14] (*Testing and Test Control Notation, Version 3*) ist eine speziell für Software- und Hardwaretesten entwickelte Programmiersprache. Ursprünglich vorwiegend für das Testen von Telekommunikationsanlagen eingesetzt, wird TTCN-3 heute auch häufig für das Testen von verteilten Web-Anwendungen benutzt.

TTCN-3 ist eine modulare Programmiersprache. Der Code wird also nicht in Klassen, sondern in Modulen gekapselt. Genau wie eine Klasse enthält ein Modul Operationen und Attribute, von einem Modul können aber keine Instanzen gebildet werden. Die Bindung zwischen Modulen ist daher immer nur statisch, d.h. sie erfolgt zur Kompilationszeit. In TTCN-3 wird die Bindung durch Importierung beliebiger Datentypen, Funktionen und Ports aus anderen Modulen realisiert (Schlüsselwort: `import`).

Außer einfachen Variablen und Funktionen beinhalten TTCN3-Module einige weitere Elemente, die beim Testen von verteilten Systemen hilfreich sind und die in den nächsten Abschnitten näher beschrieben werden (vgl. im Folgenden [15]).

2.4.1 Datentypen und Datenschablonen

Um möglichst viele Systeme (*SUT, System under Test*) zu unterstützen, bietet TTCN-3 eine breite Palette an Datentypen [15]. Neben den eingebauten (*built-in*) Typen wie `integer` oder zusammengesetzten Typen wie `record` sind spezifische Typen wie `verdicttype` und `address` vorhanden, die in anderen Programmiersprachen nicht zu finden sind. Außerdem lassen sich sehr flexibel neue Datentypen definieren, indem die vorhandenen Typen eingeschränkt werden (*subtyping*).

Zur Darstellung von Zahlen und Zeichenketten gibt es in TTCN-3 die üblichen Typen `integer`, `float` und `charstring`. Im Unterschied zu anderen bekannten Programmiersprachen besitzen diese Typen keine Wertebereiche. Die maximalen und minimalen Größen der Werte werden auf das jeweilige System angepasst, das getestet wird. Aus diesem Grund sind Datentypen wie `double`, die den Wertebereich erweitern, überflüssig. Um verschiedene Arten von Zeichenketten darzustellen, bietet TTCN-3 weitere String-Datentypen an (z.B. `universal string` zur Darstellung von Unicode-Zeichen oder `hexstring` zur Darstellung von hexadezimalen Zeichenketten).

Zur Erzeugung komplexer Datentypen besitzt TTCN-3 zwei Container-Typen: `record` und `set`. Beide dienen zur Schachtelung beliebiger Datentypen, sich selbst eingeschlossen. Im Unterschied zu `set` stellt `record` eine geordnete Liste von Werten dar.

Zur Repräsentation von Listen mit Elementen gleichen Datentyps bietet TTCN-3:

- `array` zur Darstellung eines statischen Feldes.
- `record of` zur Darstellung einer dynamischen und geordneten Liste.
- `set of` zur Darstellung einer dynamischen und ungeordneten Liste.

Um spezielle Funktionen anzubieten, die beim Testen nützlich sind, stellt TTCN-3 eine Reihe besonderer Datentypen. Der wichtigste und am häufigsten gebrauchte ist `verdicttype`. Damit wird der Zustand eines Testfalls festgelegt. Die möglichen Werte für diesen Datentyp sind: `pass` (Test bestanden), `fail` (nicht bestanden), `error` (fehlerhaft), `inconc` (von *inconclusive*, Entscheidung nicht möglich) und `none` (noch keine Angabe).

Mit dem Typ `anytype` lassen sich generische Nachrichten erstellen, die später Werte beliebigen Datentyps enthalten können. Der Typ `address` kann zum Speichern von Adressen zu entfernten Systemen benutzt werden. So kann beispielsweise die Sender-Adresse einer empfangenen Nachricht dynamisch ermittelt und so eine Antwort-Nachricht an die richtige Adresse gesendet werden.

Oft ist es nützlich, Wertebereiche für Datentypen einzugrenzen. Angenommen, ein Protokoll hätte die drei Anfragemethoden `get`, `post` und `put`. Dann kann in TTCN-3 folgendermaßen ein passender Datentyp definiert werden:

```
type charstring requestMethod („get“, „post“, „put“);
```

Der neue Typ ist ein Untertyp von `charstring`, hat den Namen `requestMethod` und kann nur die drei Werte in den Klammern annehmen. Außer der besseren Lesbarkeit werden durch diese Einschränkung mögliche Fehler bereits im TTCN3-Code vermieden, weil keine falschen Anfragemethoden aufgerufen werden können.

Für Zahlendatentypen können zusätzlich zu den oben genannten Aufzählungen von Werten auch Wertintervalle angegeben werden. Speziell für Zeichenketten und Listen-Datentypen kann außerdem die minimale und maximale Länge mit dem Schlüsselwort `length` eingestellt werden.

Konkrete Werte werden in TTCN-3 entweder als Variablen oder als Datenschablonen (*templates*) dargestellt. Während Variablen lediglich zur lokalen Zwischenspeicherung von Werten verwendet werden, müssen alle gesendeten und empfangenen Nachrichten als Datenschablonen definiert werden.

Schablonen werden mit dem Schlüsselwort `template` definiert. Eine Schablone zum oberen Beispiel-Datentyp könnte so aussehen:

```
template requestMethod := „post“;
```

Der Name „Schablone“ kommt daher, dass diese nicht nur konkrete Werte enthalten kann, sondern auch reguläre Ausdrücke. Dadurch können empfangene Nachrichten, deren Inhalt nicht genau bekannt ist, mit einem Muster verglichen werden. Für Listentypen lassen sich außerdem optionale Felder angeben, die in der Nachricht vorhanden sein können, aber nicht müssen.

2.4.2 Ports, Timer und Testkomponenten

Nachrichten, die in Form von Datenschablonen vorliegen, werden über Ports an das getestete System gesendet und von dort empfangen. Timer werden dazu benutzt, um eine bestimmte Zeit auf eine Nachricht zu warten. Testkomponenten enthalten alle Ports und Timer, die für bestimmte Testfälle notwendig sind.

Bevor ein Port in einer Testkomponente benutzt werden kann, muss er wie ein Datentyp definiert werden. Dabei werden die Typen von ein- und ausgehenden Nachrichten angegeben, die der Port empfangen und verschicken kann. Beispiel:

```
type port httpPort message {
     out request;
     in response;
}
```

Hier wird ein Port definiert, der HTTP-Anfragen versenden und HTTP-Antworten empfangen kann. Die beiden Nachrichten-Typen `request` und `response` sind benutzerdefinierte Datentypen, deren Obertyp z.B. `record` sein könnte. Die ausgehenden Nachrichten werden mit dem Befehl `send` sofort verschickt. Die ankommenden Nachrichten werden dagegen in einer Warteschlange zwischengespeichert. Sie müssen explizit mit dem Befehl `receive` abgeholt werden.

Eine Testkomponente ist eine Art Behälter für Ports, Timer, Konstanten und Variablen, die für eine Reihe von Testfällen nötig sind. Der folgende Code-Abschnitt definiert eine Testkomponente.

```
type component ServerTester {
     port httpPort port1;
     port httpPort port2;
     const requestMethod method := „get“;
     timer tmr := 3.0;
}
```

Zunächst werden zwei Ports vom Porttyp `httpPort` definiert, die zwei verschiedene Web Server ansprechen könnten. Die Konstante `method` könnte in den Testfällen dazu verwendet werden, die Anfragemethode zu ermitteln. Ein Timer wird meistens dafür verwendet, nur die angegebene Zeit (in Sekunden) auf eine Nachricht zu warten. Falls nämlich in einer bestimmten Zeit keine Nachricht ankommt, muss

davon ausgegangen werden, dass entweder das Testobjekt oder die Testinfrastruktur fehlerhaft sind. In diesem Fall wird der Testfall abgebrochen und der Teststatus (*verdict*) auf `fail` oder `error` gesetzt.

In einem TTCN3-Testfall gibt es immer eine Haupttestkomponente (MTC, *main test component*). Falls nur eine Komponente definiert wird, ist diese automatisch die Haupttestkomponente. In diesem Fall ist die Schnittstelle zu dem getesteten System komplett durch die Ports der Haupttestkomponente beschrieben. Falls das getestete System aus mehreren Teilsystemen besteht, können mehrere Komponenten definiert werden. Wie eine Komponente in einem Testfall verwendet wird, wird in Kapitel 2.4.4 näher erläutert.

2.4.3 Funktionen und Alternativ-Blöcke

Wie in anderen Programmiersprachen kapseln TTCN3-Funktionen Programmierlogik, die von anderen Stellen im Programm aufgerufen werden kann. Eine Funktion besitzt Parameter und einen Rückgabewert. Ein einfaches Beispiel für eine Funktion in TTCN-3:

```
function func (in integer int1, out integer int2)
     runs on ServerTester
     return float
{
     int2 := 1;
     return 1.2;
}
```

Statt Referenzen oder Zeigern können die Übergabeparameter mit `out` als ausgehende Parameter deklariert werden. D.h. wenn sie in der Funktion verändert werden, ändert sich ihr Wert auch außerhalb der Funktion. Eine Besonderheit der TTCN3-Funktionen ist der optionale `runs on` Befehl. Damit wird eine Testkomponente an die Funktion gebunden, was bedeutet, dass alle Ports und andere Definitionen der Komponente in der Funktion verwendet werden können. Dies wird benötigt, wenn in der Funktion Nachrichten verschickt oder empfangen werden sollen.

Ein wichtiges Konzept in TTCN-3 sind so genannte Alternativ-Blöcke (*alt statements*). Damit lassen sich Fehler beim Empfangen von Nachrichten komfortabel abfangen. Zunächst ein Beispiel:

```
port1.send(myRequest);
tmr.start;
alt {
     [] port1.receive(myResponse) {
          setverdict(pass);
          tmr.stop;
     }
     [] port1.receive {
          setverdict(fail);
          tmr.stop;
     }
     [] tmr.timeout {
          setverdict(error);
     }
}
```

Hier wird eine Nachricht verschickt und dann eine Antwort erwartet. Zuerst wird die Datenschablone `myRequest` über den Port `port1` an das getestete System gesendet. Danach wird sofort ein Timer gestartet. Im alt-Block sind alle Ereignisse aufgeführt, die nun eintreten können. Falls eine Nachricht ankommt, die sich mit der Schablone `myResponse` deckt, wird der Status auf „bestanden“ gesetzt und der Timer wird gestoppt. Wenn eine andere Nachricht als erwartet empfangen wird, dann wird der Status entsprechend auf „nicht bestanden“ gesetzt. Bei einem Timeout wird der Status schließlich auf „Fehler“ gesetzt. Tritt eines dieser Ereignisse auf, wird der zugehörige Teil-Block ausgeführt und die Ausführung des Programms nach dem ganzen Alternativ-Block fortgesetzt.

Ähnlich einer Funktion kann ein einzelner Teil-Block eines Ereignisses aus dem `alt`-Block ausgelagert werden. Dies wird mit einem so genannten Alternativ-Schritt (*altstep*) gemacht. Beispielsweise kann man den mittleren Teil-Block als einen Alternativ-Schritt so schreiben:

```
altstep receiveAny() runs on ServerTester {
     [] port1.receive {
          setverdict(fail);
          tmr.stop;
     }
}
```

Jetzt lässt sich der ganze Teil-Block durch seinen Namen aufrufen. Der Vorteil davon ist, dass solche Standard-Schritte in verschiedenen `alt`-Blöcken gestartet werden können.

Eine weitere Vereinfachung kann durch eine Aktivierung von `alt`-Schritten erreicht werden:

```
activate(receiveAny);
activate(timedOut);
```

Die aktivierten Alternativ-Schritte werden nun in allen nachfolgenden Alternativ-Blöcken implizit aufgerufen, so dass nur das Ereignis aufgeführt werden muss, bei dem eine richtige Nachricht empfangen wird.

2.4.4 Testfälle

Die definierten Funktionen und Alternativ-Schritte müssen in die richtige Reihenfolge gebracht und aufgerufen werden. Dies ist die Aufgabe eines Testfalls. Ein Testfall ist im Prinzip nichts Anderes als eine Funktion, die aber von außen, also von der Testumgebung aufgerufen werden kann. Ein Testfall wird wie folgt definiert:

```
testcase tc () runs on ServerTester {
     // Aufrufe von Funktionen
     // alt-Blöcke mit alt-Schritten
     // andere Logik (if, while,…)
}
```

Im Gegensatz zu normalen Funktionen und Alternativ-Schritten muss ein Testfall eine Testkomponente einbinden, die automatisch die Haupttestkomponente ist. Zur Benutzung von mehreren Komponenten in einem Testfall siehe [15, Kap. 5].

Optional können die Testfälle in einem Steuerungsteil (`control`) zur Ausführung gebracht werden. In TTworkbench [17] lassen sich die Testfälle jedoch auch direkt in die Testumgebung laden (siehe nächsten Abschnitt).

2.5 TTworkbench als Testumgebung

TTCN3-Dateien sind keine lauffähigen Programme. Um mit TTCN-3 beschriebene Testszenarien tatsächlich durchführen zu können, müssen diese mit einem geeigneten Compiler in ausführbaren Code übersetzt werden. Außerdem wird eine Testumgebung benötigt, damit die übersetzten Testfälle mit dem zu testenden System (*system under test*) kommunizieren können. Diese Funktionalitäten bietet die auf Eclipse [16] basierende Entwicklungsumgebung TTworkbench (Demo- und Studenten-Versionen unter [17] kostenlos erhältlich).

TTworkbench bietet zwei TTCN3-Perspektiven (Zusammenstellungen aus zusammengehörigen Fenstern): eine für die Entwicklung und Kompilation von Testfällen und die andere für die Ausführung und Protokollierung derselben. Die Entwicklungsperspektive (*TTCN-3 development perspective*) ist eine erweiterte Java-Perspektive von Eclipse. Hier lassen sich also auch die in Java programmierten Dateien der Testumgebung wie der Testadapter und der Codec bearbeiten und kompilieren.

Die Kommunikation zwischen den Testfällen und dem Testadapter wird dadurch ermöglicht, dass die TTCN3-Module in Java-Klassen übersetzt werden. Aus den Haupttestkomponenten, Ports und Datentypen werden Klassen generiert, die im Testadapter in Form von Objekten benutzt werden können. Genauer gesagt sind diese Klassen JavaBeans, d.h. sie besitzen Eigenschaften, die durch `set`- und `get`-

Methoden gesetzt und gelesen werden können. So können Daten aus dem TTCN3-Code, wie z.B. die Namen der Tetkomponenten oder der Ports, im Testadapter leicht ermittelt und benutzt werden. Für einen standardisierten Zugriff implementieren die generierten Klassen Schnittstellen (*interfaces*), die in TTworkbench-Bibliotheken enthalten sind. Um Standardfunktionen anzubieten, werden sie außerdem von generischen Klassen abgeleitet. Das Klassendiagramm in Abbildung 8 zeigt diesen Sachverhalt am Beispiel der beiden TTCN3-Datentypen `integer` und `charstring`.

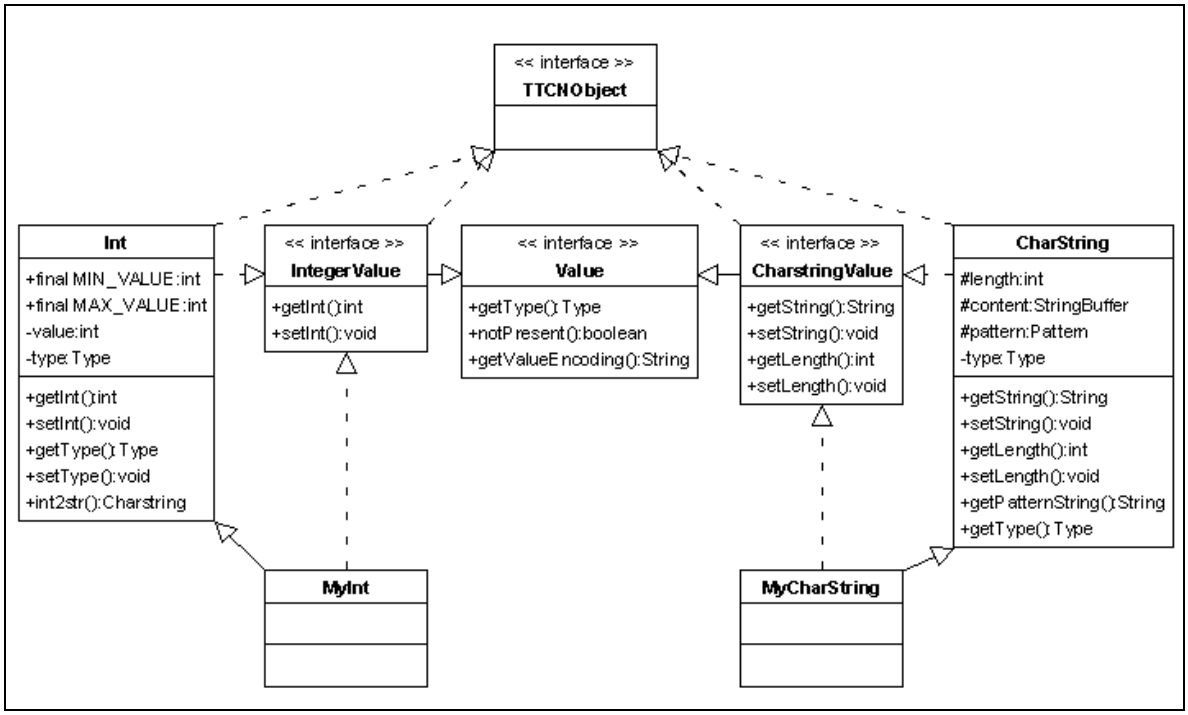


Abbildung 8: Klassendiagramm der in TTworkbench eithaltenen Repräsentationen der TTCN3-Datentypen `integer` und `charstring`

Alle Schnittstellen und Klassen außer `MyInt` und `MyCharString` sind in TTworkbench vorhanden und können benutzt werden. Die Schnittstelle `TTCNObject` enthält keine Methoden und wird wahrscheinlich von der Testumgebung zu Identifikationszwecken verwendet. `Value` ist die oberste Schnittstelle für alle TTCN3-Typen. Diese Tatsache ist für den Codec wichtig. Dieser bekommt ein `Value`-Objekt, entscheidet zur Laufzeit, von welchem konkreten Typ das Objekt ist und kann somit den übergebenen Wert aus dem Objekt rausholen und codieren (s. Kapitel 5.3.2).

Die Schnittstellen `IntegerValue` und `CharstringValue` erweitern `Value` und besitzen schon datentypspezifische Methoden[2]. `Int` und `CharString` sind schließlich konkrete Implementationen der TTCN3-Standard-Datentypen. Falls der Tester im TTCN3-Code erweiterte Datentypen verwendet, werden automatisch benutzerspezifische Klassen erstellt, die von den Hauptklassen abgeleitet werden.

Die Ausführungsperspektive (*TTCN-3 execution management perspective*) stellt Werkzeuge zur Verfügung, mit deren Hilfe die Testfälle ausgeführt und protokolliert werden können. Zuvor müssen die Testfälle mit Hilfe einer XML-Beschreibungsdatei (s. Abbildung 9) geladen werden. Die Datei enthält alle zur Testausführung

[2] Die im Klassendiagramm abgebildeten Methoden und Attribute stellen nur einen kleinen Teil der tatsächlich vorhandenen dar.

benötigten Informationen, wie beispielsweise die Namen und Positionen des Moduls, des Testadapters und der Testfälle. Diese Datei wird automatisch generiert, sollte jedoch unbedingt auf Richtigkeit geprüft und gegebenenfalls angepasst werden.

```
- <moduleloader>
  - <module File="Test01_Users.jar" Name="Test01_Users" Package="generated_ttcn">
      <testadapter Name="com.testingtech.ttcn.tri.ONEVAadapter" File="lib/ONEVAtests_TA.jar"/>
    - <testcase Name="control" Selection="true" Verdict="none" Status="stopped" Module="">
        <description>The control part of module Test01_Users</description>
      </testcase>
    - <testcase Module="Test01_Users" Name="tc1_1" Selection="true" Status="stopped" Verdict="none">
        <description/>
      </testcase>
    - <testcase Module="Test01_Users" Name="tc1_2" Selection="true" Status="stopped" Verdict="none">
        <description/>
      </testcase>
    - <testcase Module="Test01_Users" Name="tc1_3" Selection="true" Status="stopped" Verdict="none">
        <description/>
      </testcase>
    - <testcase Module="Test01_Users" Name="tc1_4" Selection="true" Status="stopped" Verdict="none">
        <description/>
      </testcase>
    </module>
  </moduleloader>
```

Abbildung 9: XML-Beschreibungsdatei Test01_Users.clf

Alle geladenen Testfälle werden im `Management View` aufgelistet und können von dort mit einem Klick auf das Männchen-Symbol gestartet werden. Der Status des aktuellen Testfalls wird im `Properties View` angezeigt.

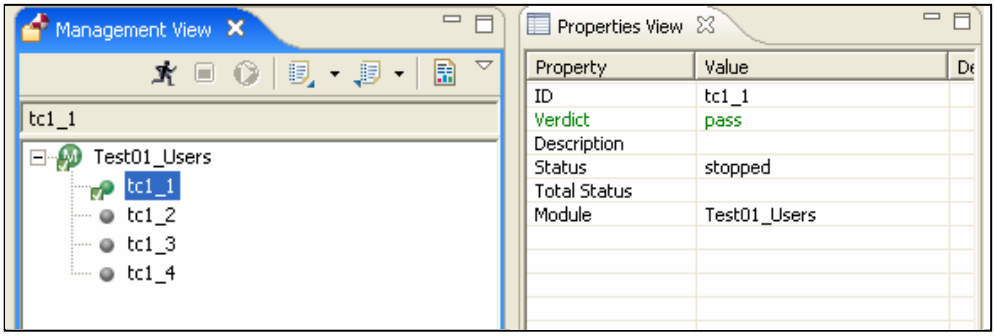


Abbildung 10: Informationen zu den geladenen Testfällen

Bei der Ausführung eines Testfalls werden alle Schritte mitprotokolliert. Es gibt zwei Ansichten: eine textuelle und eine graphische (siehe Abbildungen 11 und 12).

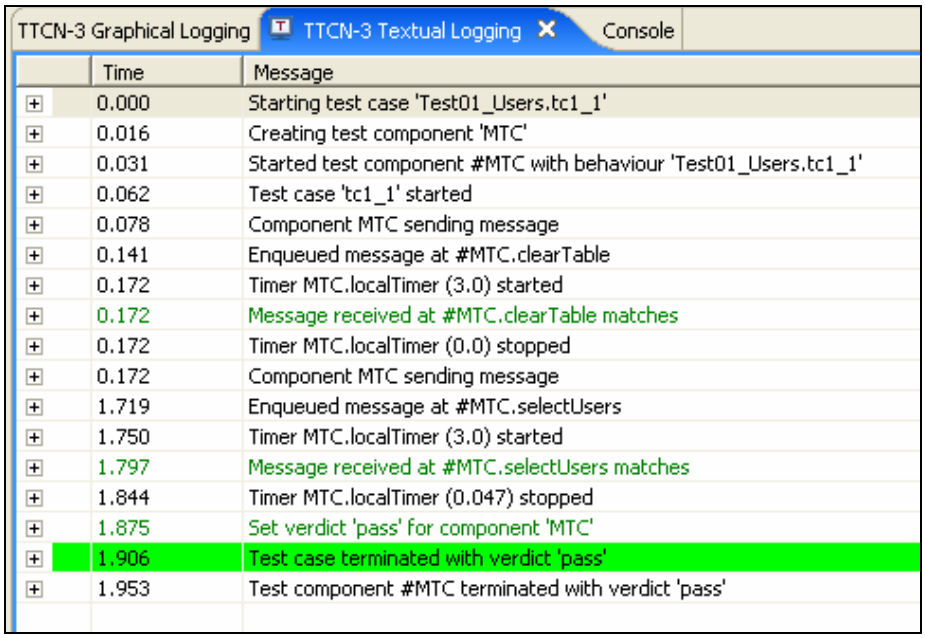


Abbildung 11: textuelles Protokoll einer Testfallausführung

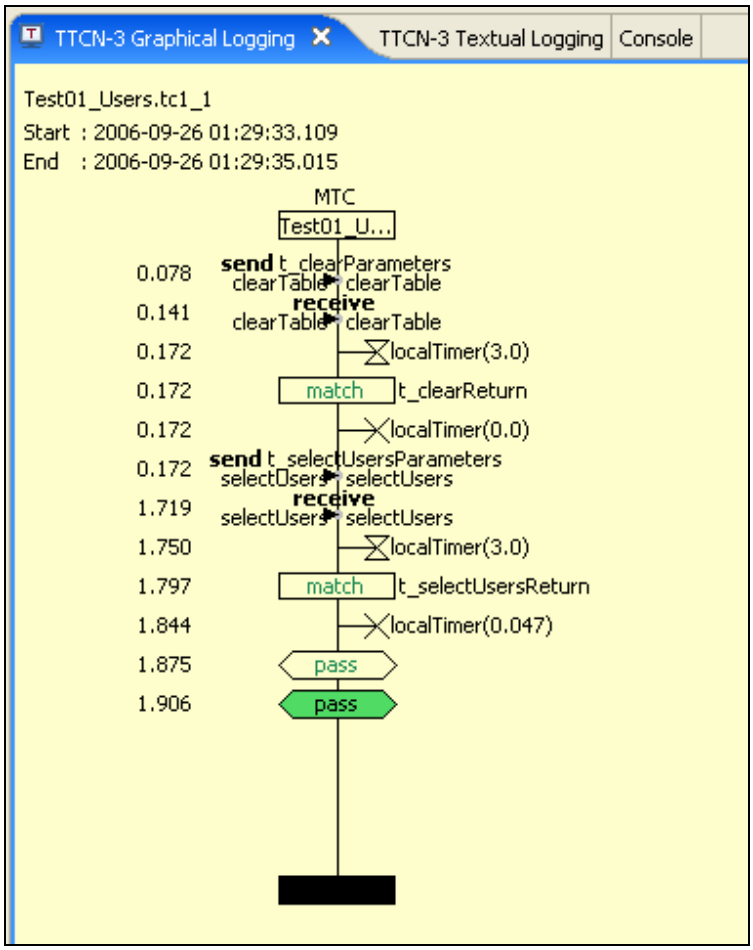


Abbildung 12: graphisches Protokoll einer Testfallausführung

3. Die Web-Applikation ONEVA

Das Online-Evaluationssystem (ONEVA) ist eine Anwendung, mit deren Hilfe Vorlesungen evaluiert werden können. Dieses Kapitel beschreibt kurz die wichtigsten Aspekte des Programms (für Details siehe [11] und [12]).

3.1 Die Benutzeroberfläche

Mit ONEVA können Lehrveranstaltungen von Studenten bewertet und die Bewertungen dann von Dozenten analysiert werden. Ein Administrator hat Zugriff auf alle Daten des Systems und kann es dadurch erweitern und pflegen.

Ein Student, der eine Veranstaltung evaluieren möchte, muss sich zunächst im System mit seiner Matrikelnummer anmelden. Danach hat er Zugriff auf die Fragebögen aller Lehrveranstaltungen und kann diese ausfüllen. In Abbildung 13 ist ein Ausschnitt eines Fragebogens dargestellt.

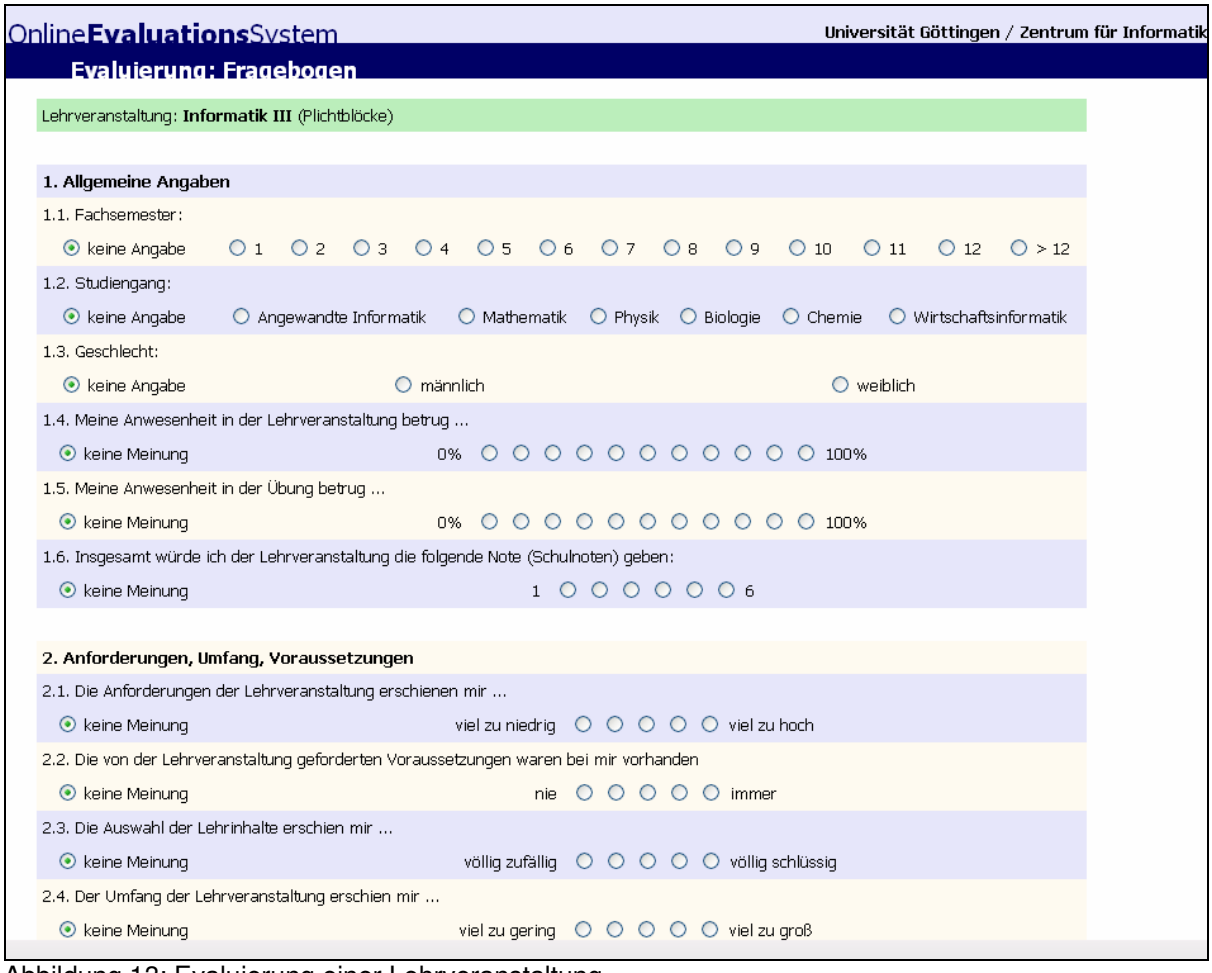


Abbildung 13: Evaluierung einer Lehrveranstaltung

Die ausgefüllten Fragebögen werden mitsamt der Matrikelnummer in einer Datenbank gespeichert, so dass ein Student eine bestimmte Vorlesung nur einmal evaluieren kann.

Ein Dozent kann Bewertungen zu Lehrveranstaltungen analysieren, die ihm zugeordnet sind. Die Ergebnisse der Evaluierungen werden in Grafiken dargestellt. Dabei kann ausgewählt werden, ob Daten für alle oder nur für ein bestimmtes Semester angezeigt werden sollen. Außerdem lassen sich die Ergebnisse durch Klicks auf die einzelnen Auswahlmöglichkeiten filtern. Beispielsweise können die Bewertungen aller männlichen Erstsemestler, die Mathematik studieren, angesehen werden. Abbildung 14 zeigt ein Beispiel eines Evaluierungsergebnisses.

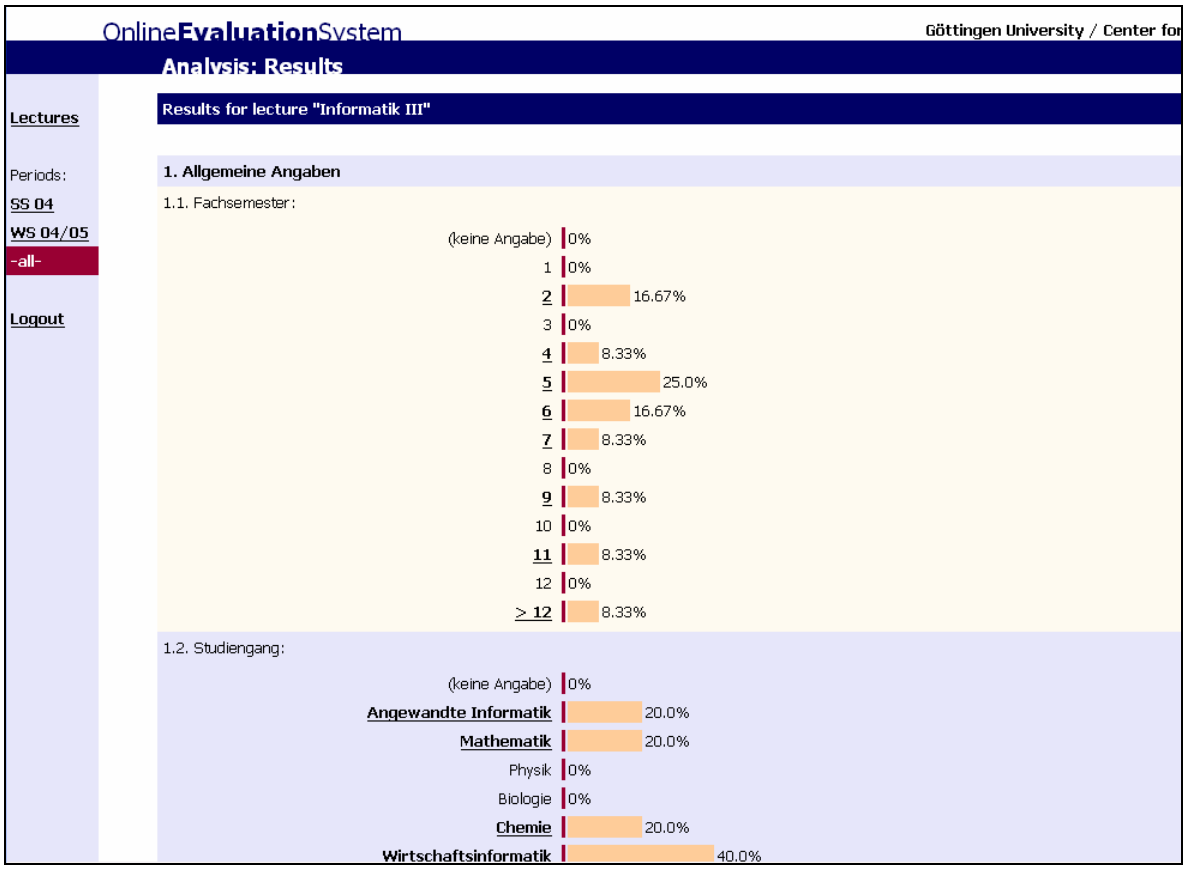


Abbildung 14: Teil eines Evaluierungsergebnisses

Die Aufgabe des Administrators ist es, alle Daten zu verwalten. Er kann unter Anderem Benutzer, Vorlesungen und Fragebögen neu einfügen, bearbeiten und löschen. Abbildung 15 zeigt, wie eine Lehrveranstaltung bearbeitet wird.

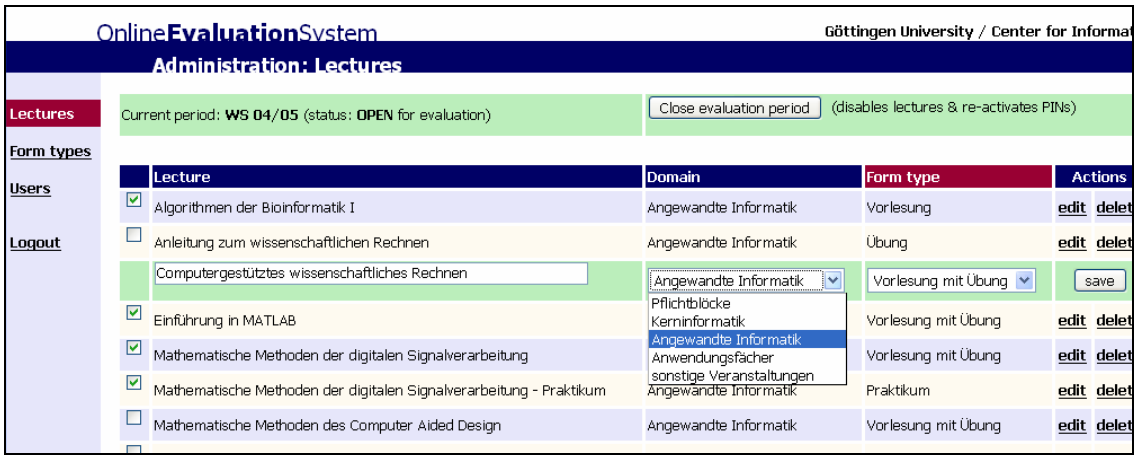


Abbildung 15: Bearbeitung einer Lehrveranstaltung

3.2 Technische Umsetzung

Das Online-Evaluationssystem wurde mit der JSP/Servlets-Technologie [18] nach dem MVC-Paradigma (*model view controller*) realisiert.

Das Modell besteht aus Web Services und einem MySQL-Datenbankserver [19]. Die Web Services greifen über die JDBC-Schnittstelle (Java Database Connectivity) auf die Datenbanktabellen und ändern deren Inhalt. ONEVA enthält die folgenden drei Web Services:

- `ADBService`: greift auf die Authorisierungsdatenbank (ADB) zu. Damit lassen sich alle Benutzerdaten und die Pins der Studenten verwalten.
- `FDBService`: greift auf die Fragebogendatenbank (FDB) zu. Hier sind alle Fragen, Fragebogentypen, Lehrveranstaltungen sowie die ausgefüllten Fragebögen gespeichert.
- `FDBtoADBService`: dient zur Synchronisation der Lehrveranstaltungstabellen in den beiden Datenbanken. Die Daten zu den Lehrveranstaltungen werden nämlich sowohl in der ADB als auch in der FDB gebraucht.

Die Web Services sind in einem AXIS-Container (*Apache eXtensible Interaction System*) [20] installiert, der auf einem Apache Tomcat-Server [21] läuft. Tomcat ist ein Web Server mit einer Umgebung zur Ausführung von Servlets und zur Übersetzung von JSP-Seiten in Servlets. Dadurch wird die dynamische Generierung von Internetseiten ermöglicht. AXIS ist eine aus Servlets bestehende Anwendung, die WSDL-Beschreibungen für Java-Klassen generiert und ankommende SOAP-Nachrichten in Java-Datentypen (und umgekehrt) umwandelt. Dadurch sehen die Java-Klassen von außen wie Web Services aus. In Tomcat wird die Web Service-Kommunikation dadurch gewährleistet, dass alle HTTP-Anfragen, die SOAP-Nachrichten enthalten, an AXIS weitergeleitet werden.

Die JSP-Seiten bilden die Benutzeroberflächenschicht (*view*). Wie bereits erwähnt, werden die JSP-Seiten in Servlets, also spezielle Java-Klassen, übersetzt und ausgeführt. Das Ergebnis der Ausführung ist eine HTML-Datei, die an den anfragenden Client (z.B. einen Web-Browser) in einer HTTP-Antwort (*response*) gesendet wird. Die HTML-Seiten werden vom Browser interpretiert und, wie im vorigen Kapitel beschrieben, für den Benutzer dargestellt.

Der Kern der Steuerungsschicht (*controller*) sind die drei Servlets `Evaluation`, `Analysis` und `Administration`. Sie kümmern sich um die Navigation zwischen den JSP-Seiten. Außerdem sind sie für den Zugriff auf die Web Services und damit auf die Daten der Datenbanken zuständig. Dazu benutzen sie die drei Java-Klassen `ADBServiceClient`, `FDBServiceClient` und `FDBtoADBServiceClient`. Diese Klassen rufen die Web Services auf und leiten die zurückgegebenen Daten an die Servlets zurück.

Weitere Details zur technischen Realisierung des Online-Evaluationssystems können in [11] und [12] nachgelesen werden.

4. ONEVA-Funktionserweiterungen

4.1 Portierung auf Linux

Das Online-Evaluationssystem wurde mit Hilfe der Entwicklungsumgebung Eclipse [16] auf einem Windows-Rechner erstellt. Da es zukünftig auf einem oder mehreren Linux-Servern laufen soll, musste die gesamte Anwendung auf dieses Betriebssystem portiert werden. Dazu wurden auf einem frisch installierten Linux-System eine Java-Umgebung, ein Tomcat-Webserver [21] mit einem AXIS-Container [20] sowie ein MySQL-Datenbankserver [19] eingerichtet. Damit die Anwendung in der neuen Umgebung laufen konnte,

- mussten in einigen XML-Dateien alte Windows-Pfade geändert werden.
- musste die Klassenpfad-Variable um alle benötigten Bibliotheken erweitert werden.
- mussten die Web Services auf dem AXIS-Container installiert werden (*deployment*) [vgl. 12, S. 30].
- wurden die gesicherten Datenbank-Daten auf den neuen MySQL-Server eingespielt.
- wurden einige Aktualisierungen im Programmcode gemacht:
  - die JDBC-Verbindung mit dem richtigen Passwort eingestellt
  - einige Java Beans, die per SOAP verschickt werden, enthielten null-Werte, wodurch Ausnahmen ausgelöst wurden. Deswegen wurden diese Werte im Programmcode explizit auf Standardwerte des jeweiligen Datentyps gesetzt.
- wurden einige kleine Fehler im Programmcode behoben.

Des Weiteren wurde die JSP-Seite `start.jsp` als eine Startseite (*welcome page*) definiert und der Port für den Zugriff auf ONEVA von 8080 zu 80 geändert. Somit hat sich der URL von
`http://<hostname>:8080/Evaluation/evaluation/start.jsp` zu
`http://<hostname>/Evaluation` verkürzt.

4.2 Erhöhte Sicherheit für die Web Services

Ein Sicherheitskonzept war in ONEVA bisher nicht implementiert [12]. Beim Login wurden lediglich der Benutzername und das Passwort in der Datenbank nachgeschlagen und bei korrekten Eingaben wurde auf die entsprechende Seite navigiert. Bei Administrator-Arbeiten bzw. bei der Analyse der Fragebögen konnte dann ohne Benutzerrechte auf die Web Services und dadurch auf die Datenbank zugegriffen werden. Jeder, der die URL-Adresse zu einer Administrator- oder Dozenten-Seite kennt oder errät, kann diese Seite ansehen. In diesem Falle hatte man ursprünglich Zugriff auf den kompletten Funktionsumfang der Applikation ohne sich einloggen zu müssen.

Eine einfache boolesche Variable in der Session (ob man angemeldet ist oder nicht) war als Lösung dieses Problems nicht ausreichend. Die Web Services können per

Definition von jedem Nutzer über das Internet gestartet werden und müssen deshalb von innen geschützt werden und nicht nur von der Benutzeroberfläche aus.

Um die beiden Probleme zu lösen wurden die Web Services um eine Sicherheitsfunktion erweitert. Jede Operation der Services erhält einen zusätzlichen Parameter `hiddenPasswd`, d.h. ein verstecktes Passwort welches dem Benutzer verborgen bleibt. Bei erfolgreicher Anmeldung im ONEVA-System wird ein solches Passwort automatisch in der Session gespeichert. Da es im Java-Code eingebunden ist, merkt der Benutzer nichts davon und arbeitet mit dem Programm wie gewohnt. Im Hintergrund wird das versteckte Passwort bei jedem Zugriff auf ein Web Service aus der Session gelesen und an die Web Service-Operation neben den anderen Parametern übergeben. In der Operation wird dieses Passwort auf Richtigkeit überprüft und nur bei einem korrekten Wert wird die Operation weiter ausgeführt.

Ohne die Kenntnis dieses Passworts kann auf die Web Services also nicht mehr von außen zugegriffen werden. Das andere Problem wurde dadurch auch gelöst: Wenn jemand direkt ohne Anmeldung auf eine JSP-Seite geht, dann enthält die Session kein verstecktes Passwort. Dadurch wird der Zugriff auf die Services nicht gewährt und die Daten in der Datenbank werden geschützt.

Es existieren drei verschiedene versteckte Passwörter: für Administratoren, Dozenten und Studenten. Auf diese Weise wird zusätzlich zu der Schutzfunktion ein rollenbasierter Zugriff auf die Web Services realisiert.

4.3 Skripte zur automatischen (De-) Installation und Kompilation

.
Das Online-Evaluationssystem besteht aus vielen Dateien, die in mehr als zehn Ordnern untergebracht sind. Um das gesamte Programm auf einem Rechner korrekt zu installieren muss man sich mit Web Servern und der JSP-Technologie gut auskennen. Aus diesem Grunde wurde eine Installationsroutine erstellt. Diese sollte es einem Benutzer möglich machen, die gesamte Anwendung leicht und schnell auf einem Rechner zu installieren. Außerdem sollte man ONEVA von einem Rechner genauso leicht wieder entfernen können. Dabei sollte der Benutzer ebenfalls nicht unbedingt ein Computer-Experte sein müssen.

Um die Applikation später bequemer weiterentwickeln zu können, sollte schließlich ein Skript zur schnellen Kompilation geschrieben werden. Der Entwickler soll ohne eine spezielle Entwicklungsumgebung wie z.B. Eclipse [16] in der Lage sein, alle zu ONEVA gehörenden Java-Quelldateien mit einem Befehl zu kompilieren und in die richtigen Pakete zu platzieren.

4.3.1 Installation

Da das System voraussichtlich nur auf Linux-Rechnern laufen wird, ist die Installationsroutine als ein Linux-Bashskript [22] implementiert. Ein Bashskript ist im Prinzip nichts Anderes als eine Folge von Shell-Befehlen wie `mkdir` oder `cp`. Hinzu kommen Kontrollflussanweisungen, also Schleifen und `if`-Bedingungen. Durch den `echo`-Befehl lassen sich Dialoge mit dem Benutzer leicht realisieren.

Viele Installationsschritte erfordern Administratorrechte. Deswegen prüft das Skript zuallererst, ob der aktuelle Benutzer über Root-Rechte verfügt. Ist das nicht der Fall, wird das Skript mit einer Fehlermeldung beendet. Ein großer Teil des Installationsskripts beschäftigt sich mit der Überprüfung des vorhandenen Systems. Um Fehler zu vermeiden ist es wichtig, dass alle für ONEVA benötigten Programme bereits installiert sind. So ist ein Apache Tomcat Web Server [21] zwingend erforderlich, genauso wie ein MySQL Datenbank-Server [19]. Eine Java Laufzeitumgebung muss ebenfalls installiert und korrekt eingerichtet sein. Wird eine dieser Anwendungen nicht gefunden, bricht die Installation sofort mit einem entsprechenden Hinweis ab.

Wenn die Systemprüfung erfolgreich abgeschlossen ist, werden zunächst die beiden Datenbanken ADB (*authorization database*) und FDB (f*orms database*) erstellt. Alle nötigen Tabellen werden automatisch angelegt und mit Daten gefüllt. Dazu werden SQL-Sicherungs-Dateien (*dumps*) ausgeführt, die in demselben Verzeichnis liegen wie das Installationsskript selbst. Die SQL-Dateien enthalten entweder Testdaten wie sie zurzeit vorliegen oder später (bei einer möglichen Inbetriebnahme) Daten aus vorhergehenden Installationen.

Falls noch nicht vorhanden, wird der Web Container Apache AXIS [20] installiert, wobei die benötigten Dateien und Bibliotheken in die richtigen Ordner im Tomcat-Pfad kopiert werden. Dieser Container ist für die Ausfuhrung der Web Services zuständig.

Der ganze Ordner mit den ONEVA-Dateien wird ebenfalls auf den Tomcat-Server kopiert. Die Dateien sind in diesem Ordner bereits richtig angeordnet, so dass die Anwendung sofort gestartet werden kann. In dem Fall, dass Daten oder Dateien schon vorhanden sind, wird der Benutzer immer nachgefragt, ob überschrieben werden soll oder nicht.

Um den Zugriff auf die Datenbank zu ermöglichen, werden schließlich die Web Services `ADBService`, `FDBService` und `FDBtoADBService` auf dem AXIS-Container installiert (*deployment*). Dies geschieht mit Hilfe des AXIS-Programms `AdminClient` und der bereits vorhandenen WSDD-Dateien (*Web Service Deployment Descriptor*). Nach einem Neustart von Tomcat steht ONEVA über die Internet-Adresse `http://<hostname>/Evaluation` zur Verfügung.

4.3.2 Deinstallation

Bei der Deinstallationsroutine, die ebenfalls als ein Bash-Skript implementiert ist, werden alle ONEVA-Programmteile von der Festplatte gelöscht. Bevor alle Daten aus der ADB und der FDB entfernt werden, speichert das Deinstallationsskript die Daten in SQL-Sicherungen (*dumps*) ab. Diese Dateien werden dazu verwendet, bei der nächsten Installation (beispielsweise auf einem anderen Rechner) die beiden Datenbanken wieder herzustellen. Auf diese Weise kann ONEVA leicht und schnell auf einen anderen Rechner transportiert werden.

Bevor die ONEVA-Dateien gelöscht werden können, wird der Benutzer nach den Pfaden zu der Java- und der Tomcat-Installation gefragt. Dieser Schritt entfällt, wenn die beiden Umgebungsvariablen `JAVA_HOME` und `TOMCAT_HOME` richtig gesetzt sind.

Der Pfad zu Java wird für das Entfernen der Web Services (*undeployment*) benötigt. Mit dem Pfad zu Tomcat kann die ONEVA-Applikation im Dateisystem gefunden werden.

Nachdem die Datenbanken gesichert und gelöscht sind, werden die Web Services aus dem AXIS-Container entfernt. Dazu verwendet das Skript wieder WSDD-Dateien. Schließlich wird das komplette Verzeichnis, in dem ONEVA liegt, und optional der AXIS-Container gelöscht.

4.3.3 Kompilation

Die Kompilation der Java Quelldateien ist als ein ANT-Skript implementiert. ANT (*Another Neat Tool* [23]) ist eine Alternative zu den klassischen `make`-Dateien. Es wird in XML codiert, ist leichter zu erlernen und ist nicht so fehleranfällig.

Das ANT-Skript ist sehr kurz und einfach. Es werden nur die Quell- und Zielordner sowie zwei benötigte Java Archive für den Klassenpfad (*classpath*) angegeben. Alles Andere wird automatisch von ANT erledigt.

```
- <project name="oneva" basedir=".">
    <description>ONEVA build file</description>
    <property name="src" location="JavaSource" />
    <property name="build" location="WEB-INF/classes" />
  - <target name="compile" description="Compile from JavaSource into WEB-INF/classes">
      <!--  Compile the java code from ${src} into ${build}   -->
    - <javac srcdir="${src}" destdir="${build}">
      - <classpath>
          <pathelement location="WEB-INF/lib/axis.jar" />
          <pathelement location="WEB-INF/lib/j2ee.jar" />
        </classpath>
      </javac>
    </target>
  </project>
```

Abbildung 16: build.xml

5. Testfälle

Dieser Abschnitt beschreibt die Erstellung und Ausführung der Testfälle sowie die Resultate, die bei der Ausführung beobachtet werden.

Bevor die Testfälle implementiert werden können, müssen diese methodisch hergeleitet werden. Mit Hilfe eines Testplans werden zunächst logische Testfälle erstellt. Diese verschaffen einen ersten Überblick, es werden aber noch keine konkreten Werte bzw. Übergabeparameter für die Web Services angegeben. Im nächsten Schritt werden die konkreten Testfälle in der Programmiersprache TTCN-3 [14] (Testing and Test Control Notation) implementiert. Um diese auszuführen, wird eine Testumgebung innerhalb von TTworkbench [17] programmiert.

5.1 Testplan

Der Testplan besteht aus mehreren Teilen, die nachfolgend aufgeführt werden. Dies sind erste Überlegungen zu der Art und Weise, wie die Testfälle hergeleitet werden.

5.1.1 Testobjekte und Testziele

Es werden die Web Services `FDBService`, `ADBService` sowie `FDBtoADBService` getestet (Spezifikation im Anhang). Bei diesen handelt es sich um einen Bestandteil des „Online-Evaluationssystems“ (ONEVA), welches bereits implementiert ist. Die drei Web Services haben die Aufgabe, auf zwei Datenbanken (Authorisierungs- und Fragebogendatenbank) zuzugreifen, d.h. Daten in die Datenbanken zu schreiben und aus den Datenbanken zu lesen. Jeder Datenbank-Zugriff ist durch eine Funktion in den Web Services festgelegt.
Die Web Services sind als Java-Klassen implementiert und werden in einem AXIS-Container auf einem Tomcat Web Server ausgeführt.

Hauptsächlich wird die Funktionalität der Web Services geprüft. Das bedeutet es wird sichergestellt, dass die Services das in der Spezifikation festgelegte Verhalten fehlerfrei und vollständig aufweisen. Zusätzlich wird durch einen Test auf Robustheit das Verhalten der Services auf fehlerhafte Eingabedaten geprüft. Dies sind Sonderfälle, die nicht von der Spezifikation abgedeckt sind, die aber keinen Programmabsturz o.ä. hervorrufen dürfen.

5.1.2 Testverfahren

Die Web Services werden „in Aktion“ untersucht, d.h. sie werden auf einem Rechner ausgeführt und beobachtet. Aus diesem Grund wird ein dynamischer Test durchgeführt und kein statischer. Des Weiteren ist die Struktur der einzelnen Funktionen uninteressant. Es ist unwichtig, wie die Web Services ihre Aufgaben verrichten, nur die Ergebnisse (der Rückgabewert oder der geänderte Zustand der Datenbank) sind von Bedeutung. Die Web Services als Middleware beruhen auf dem Konzept, eine Funktionalität durch eine Schnittstelle für andere (möglicherweise entfernte) Prozesse bereitzustellen. Dabei ist es sogar unwichtig, in welcher Programmiersprache ein Web Service implementiert ist. Allein seine Funktionalität

zählt. Aus diesen Gründen wird hier der Blackbox-Test gewählt und nicht der Whitebox-Test, bei dem die Struktur eines Testobjektes untersucht wird. An einigen Stellen wird jedoch die innere Struktur des Systems dazu benutzt, Testfälle zu erstellen. So kann zum Beispiel die maximale Länge eines String in einer bestimmten Tabelle durch das Anschauen des Datenbank-Schemas ermittelt werden. Außerdem wird auf Datenbanktabellen direkt von außen zugegriffen, ohne dass die Web Services aufgerufen werden. Insgesamt könnte man das Verfahren also einen Greybox-Test nennen.

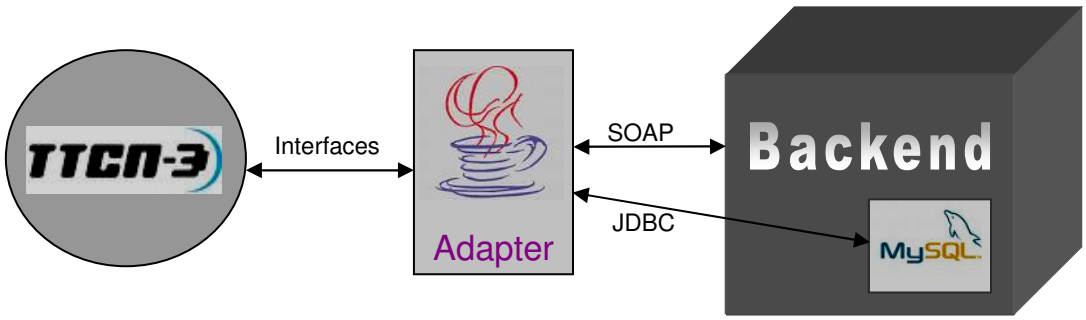


Abbildung 17: Funktionsweise der Testausführung

Abbildung 17 verdeutlicht das benutzte Testverfahren. Die Web Services sind ein Bestandteil der Modell-Schicht von ONEVA (*backend*). TTCN3-Testfalldefinitionen greifen auf die Web Services mit Hilfe des in Java implementierten Testadapters zu. Neben dem Zugriff über vordefinierte WSDL-Schnittstellendefinitionen kann der Datenbankzustand auch direkt geändert werden. Dies wird dazu benutzt, um die Datenbanktabellen für die Testfälle zu initialisieren.

5.1.3 Testpriorisierungen

Priorisiert werden „reguläre“ Testfälle, das heißt solche, die der Spezifikation und der üblichen Anwendung des Systems entsprechen. Dabei werden nur „passende“ Parameter übergeben, z.B. Strings, die nur aus Buchstaben und Zahlen bestehen. Solche Fälle sind nicht besonders fehleranfällig, sind aber entscheidend für das Funktionieren des Programms. Da für das Speichern und Ändern fast aller Daten der Administrator verantwortlich ist, wird davon ausgegangen, dass nur sinnvolle Daten eingetragen werden, d.h. keine Benutzernamen mit Sonderzeichen o.ä.

Schließlich werden „nicht-reguläre“ Testfälle erstellt, also mit Parametern wie Null-Werte, zu lange Strings und Strings mit Sonderzeichen. Hier wird davon ausgegangen, dass alle Funktionen, die Daten in die Datenbank eintragen, mit solchen Parametern gleiche Ergebnissen liefern. Schließlich sind sie sehr ähnlich implementiert und benutzen dieselbe Datenbank. Auch die Funktionen, die Daten aus der Datenbank entfernen, haben einen ähnlichen Aufbau. Aus diesen Gründen werden für die nicht-regulären Testfälle nur die Funktionen `addUser`, `deleteUser` und `selectUsers` benutzt (letztere zur Bestätigung der Zustandsänderung).

5.1.4 Testmethode

Das Verhalten der hier zu testenden Web Services hängt von dem internen Zustand der Datenbank ab. „Interner Zustand“ bedeutet hier die Menge aller gespeicherten Daten in der Datenbank. Für die meisten zu testenden Funktionen reicht es jedoch, nur den Inhalt einer oder einiger Tabellen zu betrachten, auf die diese Funktionen zugreifen. Beispielsweise kann in einem bestimmten Zustand mittels `addLecture` eine neue Vorlesung hinzugefügt werden, wodurch sich der Zustand verändert. Ein zweiter Aufruf derselben Funktion mit denselben Parametern bewirkt jedoch keine Zustandsveränderung mehr. Das bedeutet, dass Funktionen bei verschiedenen Zuständen getestet werden müssen, weshalb ein zustandsbasierter Test als Testmethode gewählt wird.

Um einen gewünschten Zustand in einer Tabelle zu erreichen, müssen oftmals mehrere Funktionen aufgerufen werden. Erst dann kann eine zu testende Funktion bei dem richtigen Zustand aufgerufen werden. Funktionen, die also zur Initialisierung von Zuständen benötigt werden, sollten in vorhergehenden Testfällen schon getestet worden sein. Dadurch wird vermieden, dass unklar ist, welche Funktion gegebenenfalls einen Fehler verursacht hat.

Die Zustandsübergänge werden in einem Zustandsbaum modelliert. Die Knoten des Baumes repräsentieren die einzelnen Zustände. Die Kanten sind Funktionen, welche die entsprechenden Zustandsübergänge verursachen. Der Zustand „leer“ bedeutet, dass die Tabellen, auf die die Funktion zugreift, keine Daten enthalten. Um die Übersicht zu behalten, werden alle anderen Zustände „gefüllt“ oder „1 User“ o.ä. genannt. „Plus 1 User“ bedeutet, dass zum ursprünglichen Zustand ein Benutzer hinzugefügt wurde. Abbildung 18 zeigt ein einfaches Beispiel.

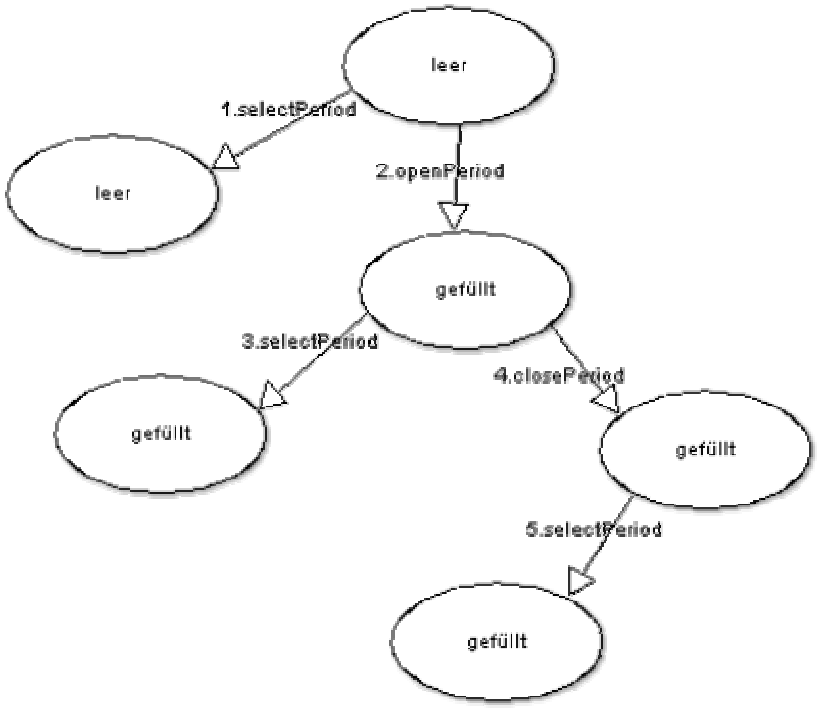


Abbildung 18: Zustandsbaum zur Manipulation der Tabelle `period`

Ein Zustandsbaum wird aus zusammengehörigen Funktionen konstruiert, d.h. aus Funktionen, die auf dieselben Tabellen zugreifen und voneinander abhängen. So kann ein Benutzer nicht gelöscht werden, solange er noch nicht in der Datenbank gespeichert wurde. Um die Funktion `deleteUser` aufzurufen, muss also direkt davor `addUser` mit entsprechenden Parametern aufgerufen werden. Als Beispiel sind die folgenden Funktionen in einem Baum zusammen zu bringen: `addUser`, `selectUsers`, `saveUser` und `deleteUser`.

Einige Funktionen haben keinen Rückgabewert. Also lässt sich nicht direkt überprüfen, ob diese korrekt ablaufen und möglicherweise einen Zustandswechsel hervorrufen. Deswegen muss oftmals eine `select…()`-Funktion am Ende eines Testfalls aufgerufen werden, um den Zustandswechsel zu bestätigen.

Aus einem Zustandsbaum werden direkt die Testfälle abgeleitet. Ein Testfall entspricht genau einem Pfad von der Wurzel bis zu einem Blatt des Baums. In den meisten Fällen wird in einem Testfall nur eine Funktion getestet. Die restlichen Funktionen (die bestenfalls schon getestet wurden) dienen zur Initialisierung und zum Nachweis der richtigen Zustände. Allerdings werden dadurch auch anwendungsfall-ähnliche Abläufe erreicht. Das bedeutet, dass Abläufe wie „Benutzer hinzufügen“ –> „Benutzer editieren“ –> „alle Benutzer anzeigen“ ebenfalls getestet werden.

Um die Datenbank-Zustände besser beeinflussen zu können, gehen alle Testfälle von leeren Tabellen aus (Startzustand). Einige Datensätze lassen sich jedoch durch die Web Services nicht mehr löschen. Aus diesem Grund muss bei den konkreten Testfällen eine Hilfsfunktion implementiert werden, die direkt auf die Datenbank zugreift und den Startzustand herstellt (d.h. alle Daten aus den Tabellen entfernt).

Aus Übersichtsgründen und Platzmangel sieht man in den Bäumen nur die Namen der Funktionen. Diese werden nummeriert und in separaten Tabellen detaillierter aufgeführt. Hier können die übergebenen Parameter und die Rückgabewerte der Funktionen abgelesen werden. Da hier zunächst logische Testfälle definiert werden, sieht man keine konkreten Übergabeparameter, sondern nur allgemeine Beschreibungen. „Regulärer String“ bedeutet z.B. eine String-Variable mit lateinischen Buchstaben und/oder Zahlen 0-9.

Testfall #	*Operation*	*Parameter*	*erw. Rückgabewert*
8_1	1.selectPeriod	-	leeres Period-Objekt
8_2	2.openPeriod	regulärer String	-
-	3.selectPeriod	-	Period-Objekt mit open==true
8_3	4.closePeriod	-	-
-	5.selectPeriod	-	Period-Objekt mit open==false

Die Zustandsbäume und die dazugehörigen Tabellen sind im Anhang aufgeführt.

5.2 Implementierung der Testfälle in TTCN-3

Wie oben bereits beschrieben ist TTCN-3 eine speziell für das Testen von Software und Hardware entwickelte Programmiersprache. Damit lassen sich sehr bequem Testszenarien beschreiben, entweder als eine Referenz oder (wie hier) als ausführbare Tests.

Im Folgenden wird geschildert wie die konkreten Testfälle in Modulen untergebracht und implementiert sind. Danach wird ein Beispiel-Testfall gezeigt und mit Hilfe einiger Codeausschnitte erläutert.

5.2.1 Definition der Module

Ein Testfall ist nichts Anderes als eine Folge von Funktionsaufrufen wie sie in den Zustandsbäumen definiert sind (siehe Kapitel 5.1.4).

Die zuvor erstellten logischen Testfälle dienen als Grundlage für die TTCN3-Implementierungen. Aus jedem Zustandsbaum wird ein TTCN3-Modul erstellt, in dem die einzelnen Testfälle sowie alle zu versendeten Nachrichten definiert sind.

Jedes Modul beinhaltet Tests zu zusammengehörigen Funktionen. In dem Modul `Users` werden beispielsweise Testfälle zu den Funktionen `selectUsers`, `addUser`, `saveUser` und `deleteUser` definiert. Auf diese Weise sind alle Funktionen der ONEVA-Web Services auf insgesamt neun Module verteilt.

Im Vergleich zu der Benutzung nur eines Moduls wird durch diese Modularisierung vor Allem die Übersicht über die Testfälle verbessert. Der Nachteil davon ist, dass viele Definitionen von Datentypen, Schablonen (*templates*) usw. in mehreren verschiedenen Modulen benötigt werden. Um diese Redundanz zu verringern, werden ähnliche Definitionen in Gruppen (TTCN-3-Schlüsselwort `group`) zusammengefasst, so dass diese in anderen Modulen leicht importiert werden können. Die Funktion, zu der die jeweilige Gruppe gehört, ist an dem Namen der Gruppe erkennbar.

Eine Gruppe wird folgendermaßen innerhalb eines Moduls definiert:

```
group g_addUser {
     // Definitionen zur Funktion „addUser“
}
```

Die einzige Funktion einer Gruppe ist die Strukturierung von Code, wodurch ganze Code-Abschnitte in andere Modulen leicht importiert werden können.

Beispiel:
Die Funktion `addUserLecture` ordnet eine Lehrveranstaltung (*lecture*) einem Dozenten (*user*) zu. Diese Funktion wird in dem Modul `Lectures` getestet. Die Typ- und Portdefinitionnen zum Einfügen eines Dozenten sind bereits in dem Modul `Users` vorhanden und können von dort importiert werden. Da diese bereits in einer

Gruppe zusammengefasst sind, kann die Gruppe, wie in Abbildung 19 gezeigt, importiert werden (`group g_addUser`).

```
module Test03_Lectures {
    import from Test01_Users {
        const adminPasswd;
        group g_clearTables;
        group g_addUser;
    }
```

Abbildung 19: Die `import`-Funktionalität in einem TTCN3-Modul

Die importierten und die im selben Modul definierten Gruppen enthalten Typdefinitionen und Schablonen (*templates*). Typdefinitionen sind TTCN3-Repräsentationen der Java-Datentypen, die in den Web Services verwendet werden. Komplexe Datentypen wie Felder, Vectoren oder JavaBeans werden in TTCN-3 als `record` oder `record of` repräsentiert. Übergabeparameter für die Funktionen der Web Services werden ebenfalls als `record` gekapselt, damit sie zusammen in einer Nachricht verschickt werden können.

Die Schablonen enthalten konkrete Werte, die gesendet und empfangen werden sollen. In jedem `group`-Block befinden sich meist mehrere Schablonen zum Senden (verschiedene Übergabeparameter). Falls die zu testende Funktion einen Rückgabewert hat, sind auch eine oder mehrere Schablonen definiert, die empfangen werden sollen.

Für jede Funktion ist in der Haupttestkomponente (*main test component*) ein Port zum Senden und Empfangen der Nachrichten definiert. Die Ports haben dieselben Namen wie die jeweiligen Funktionen, damit die richtige Funktion in der Testumgebung leicht ermittelt werden kann. Der Vorteil liegt darin, dass der Name der Funktion nicht extra in der Nachricht neben den Übergabeparametern geschickt werden muss.

Jede zu testende Web Service-Funktion ist auf eine TTCN3-Funktion abgebildet. Diese TTCN3-Funktionen kümmern sich um das Verschicken und Empfangen der Schablonen über die Ports, das heißt um den Aufruf der konkreten Web Service-Funktionen. Außerdem setzen sie den Teststatus (*verdict*) auf „bestanden“ (*pass*) oder „nicht bestanden“ (*fail*), je nachdem, ob der erwartete Rückgabewert empfangen wird oder nicht. Eine Zusammenstellung der TTCN3-Funktionen in bestimmter Reihenfolge mit bestimmten Schablonen ergibt einen Testfall, der ausgeführt werden kann.

5.2.2 Beispiel eines Testfalls

In diesem Abschnitt wird beispielhaft ein Testfall gezeigt und erklärt.

Abbildung 20 zeigt einen einfachen Testfall.

```
testcase tc5_1() runs on FDBTester {
    f_init();
    f_selectFormtypes(a_selectFormtypesAll, a_emptyFormtypesVector);
}
```
Abbildung 20: Ein Testfall, der alle Fragebogentypen auswählt

Der Name der Komponente (*component*) `FDBTester` deutet zunächst darauf hin, dass der `FDBService` aufgerufen werden soll. In der Testumgebung wird der Name der Komponente dazu benutzt, den richtigen Web Service zu ermitteln.

Dieser Testfall macht nichts Anderes als zu versuchen, alle Fragebogentypen auszuwählen, die in der FDB gespeichert sind. Da die Datenbanktabelle `formtypes` kurz davor in der Funktion `f_init()` gelöscht wird (s.u.), sollte ein leerer Vector (in TTCN3 ein `record of`) als Ergebnis zurückgeliefert werden. Bei allen anderen Ergebnissen wird der Testfall als „nicht bestanden“ (*fail*) angesehen.

Die Funktion `f_init()` wird als Erstes in jedem Testfall aufgerufen. Sie erledigt einige Initialisierungsaufgaben, wie in Abbildung 21 zu sehen.

```
function f_init() {
    activate(alt_timeout());
    activate(alt_receiveAny());

    f_clearTable("formtypes");
}
```
Abbildung 21: Initialisierungsfunktion eines Testfalls

Durch den `activate`-Befehl werden zwei Alternativschritte (*altsteps*) aktiviert, d.h. sie werden in jeden `alt`-Block implizit hinzugefügt. Dies ist deshalb sinnvoll, weil beim Empfangen jeder Nachricht die beiden möglichen Fehler abgefangen werden müssen:
1. Die Nachricht kommt innerhalb einer (längeren) Zeit nicht an (`alt_timeout`).
2. Es kommt eine andere Nachricht an als erwartet (`alt_receiveAny`).

Die Funktion `f_clearTable()` löscht alle Daten einer bestimmten Datenbank-Tabelle. Diese Funktion ist nicht in ONEVA implementiert, sondern wird nur als Hilfe für das Kontrollieren des Datenbankzustandes benutzt. Die Testumgebung greift dabei direkt auf die Datenbank zu und entfernt alle Daten aus der angegebenen Tabelle.

Die Funktion `f_selectFormtypes` in Abbildung 20 ist ein Stellvertreter für die Funktion `selectFormtypes` in dem `FDBService`. Der erste Parameter ist eine Schablone, die an den Web Service gesendet wird. Die zweite Schablone ist der erwartete Rückgabewert, in diesem Fall also ein leerer Java-Vector, bzw. ein leerer `record of` in TTCN-3.

5.2.3 Nicht-reguläre Testfälle

Testfälle mit Sonderzeichen werden nur in einem Modul durchgeführt, weil die Funktionen in den anderen Modulen mit hoher Wahrscheinlichkeit genauso reagieren würden. Es werden 35 verschiedene Strings an die beiden Funktionen `addUser` und `deleteUser` übergeben (Abbildung 22).

```
const Irr irregulars := ("", " ", "^", "°", "²", "§", "$", "½", "&", "/", "?", "ß",
    "´", "`", "\\", "\\\\", "\r", "\t", "\d", "\"", "+", "*", "'", "#", "~", "-",
    ";", "|", "<", ">", "µ", "@", "€", null, "12345678901234567890123456789012");
```

Abbildung 22: Code-Ausschnitt aus Test01_Users.ttcn3, Array mit potenziell fehlerträchtigen Parametern

5.3 Testumgebung

Die Testumgebung ist ein Bindeglied zwischen den Testfällen und den Web Services. Der wichtigste Bestandteil ist der Testadapter, der einen Codec benutzt um die ausgehenden Nachrichten zu kodieren sowie die ankommenden Nachrichten zu dekodieren. Abbildung 23 zeigt stark vereinfacht einen Web Service-Aufruf aus einem Testfall heraus.

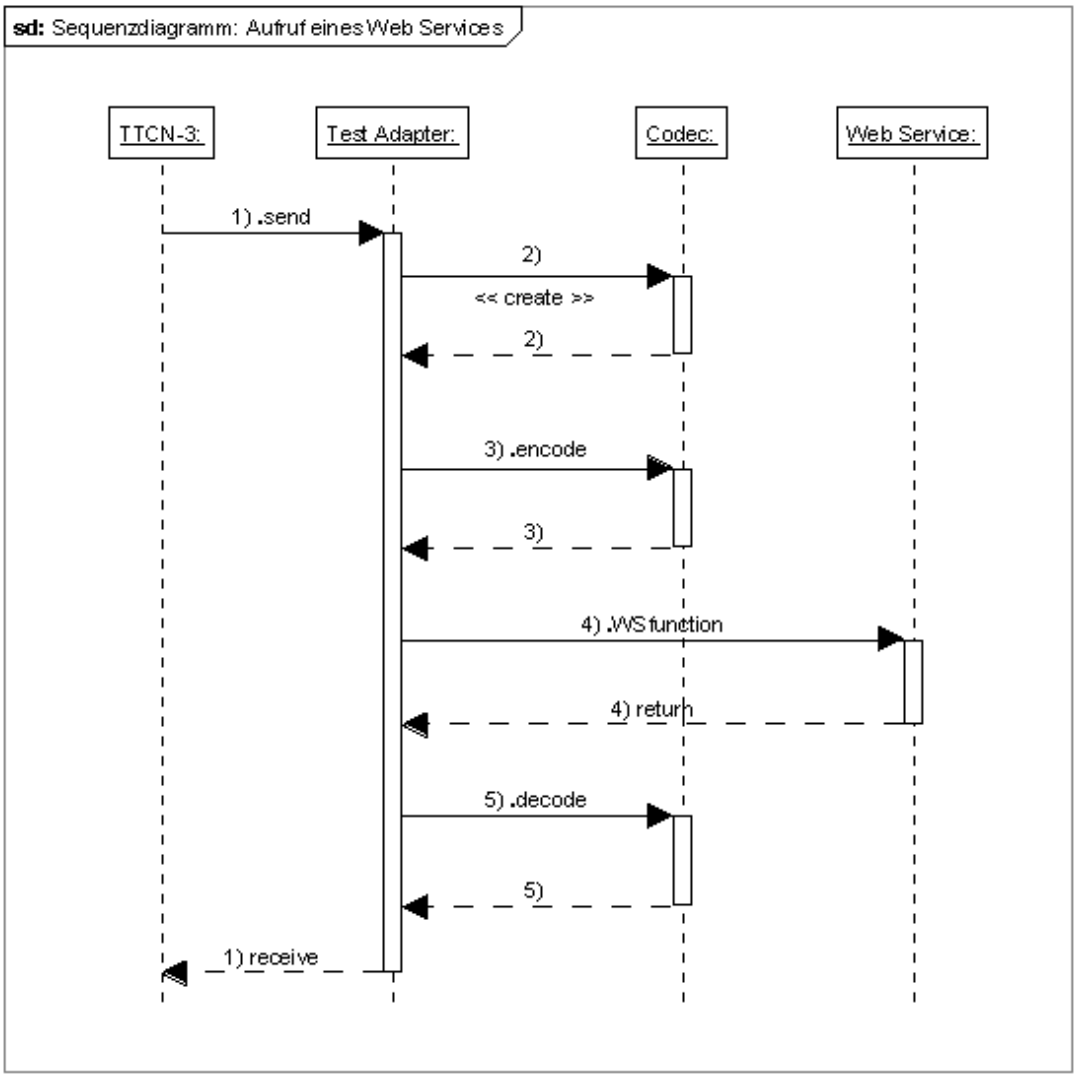


Abbildung 23: Ablauf beim Aufruf eines Web Services

Die folgenden Schritte werden ausgeführt:

1. Im TTCN3-Code wird die Funktion `send` mit einer Nachricht aufgerufen. Die Nachricht enthält Parameter für die Funktion `WSfunction`, die ein Teil des `Web Service` ist. Die Parameter sind TTCN3-Werte von Typen wie `integer` oder `charstring`. Der Testadapter, der dem aktuellen Testfall in einer XML-Datei zugewiesen ist, wird automatisch gestartet und erhält die Nachricht.

2. Der Testadapter instanziiert den benötigten Codec und legt ihn in einer Datenstruktur ab, damit bei späteren Testfällen ein schnellerer Zugriff erfolgen kann.

3. Der Testadapter reicht die Nachricht an den Codec weiter. Der Codec wandelt die TTCN3-Werte in Java-Werte um und gibt diese an den Adapter zurück.

4. Der Adapter stellt eine Verbindung zu dem Web Service her und ruft die Methode `WSfunction` auf. Als Parameter für die Funktion werden die kodierten Werte benutzt. Der Rückgabewert (*return*) wird zunächst in einer Variablen zwischengespeichert.

5. Damit der Rückgabewert in TTCN-3 benutzt werden kann, muss er dekodiert werden (*de*kodiert aus der Sicht von TTCN-3). Dazu wird die Methode `decode` des Codecs benutzt. Der dekodierte Wert wird in eine Warteschlange eingereiht, die von TTCN-3 aus zugreifbar ist.

6. (Im Diagramm der Rückgabepfeil von 1.) Durch den Aufruf von `receive` im TTCN3-Code wird die angekommene Nachricht aus der Warteschlange gelesen und kann nun auf ihre Richtigkeit überprüft werden.

Außer dem Codec und dem Testadapter muss eine Testumgebung einige weitere Funktionen vorweisen. Beispielsweise müssen die TTCN3-Dateien zunächst zu ausführbaren Programmen kompiliert werden. Es müssen Timer implementiert sein, die Tests müssen ausgewertet und die Resultate müssen angezeigt werden. Dies und alles Andere was bei der Ausführung von Testfällen nötig ist, bietet das Programm TTworkbench (siehe Kap. 2.4). Der Testadapter und der Codec müssen allerdings vom Tester selber programmiert werden, damit die Nachrichten richtig kodiert und versendet werden können.

Nach dieser allgemeinen Einführung werden in den folgenden Abschnitten der Testadapter und der Codec, die speziell für ONEVA implementiert wurden, genauer betrachtet.

5.3.1 Testadapter

Die Bibliothekensammlung von TTworkbench enthält bereits generische Klassen und Schnittstellen (*interfaces*) für Testadapter und Codecs. Abbildung 24 bietet eine vereinfachte Übersicht als ein Klassendiagramm.

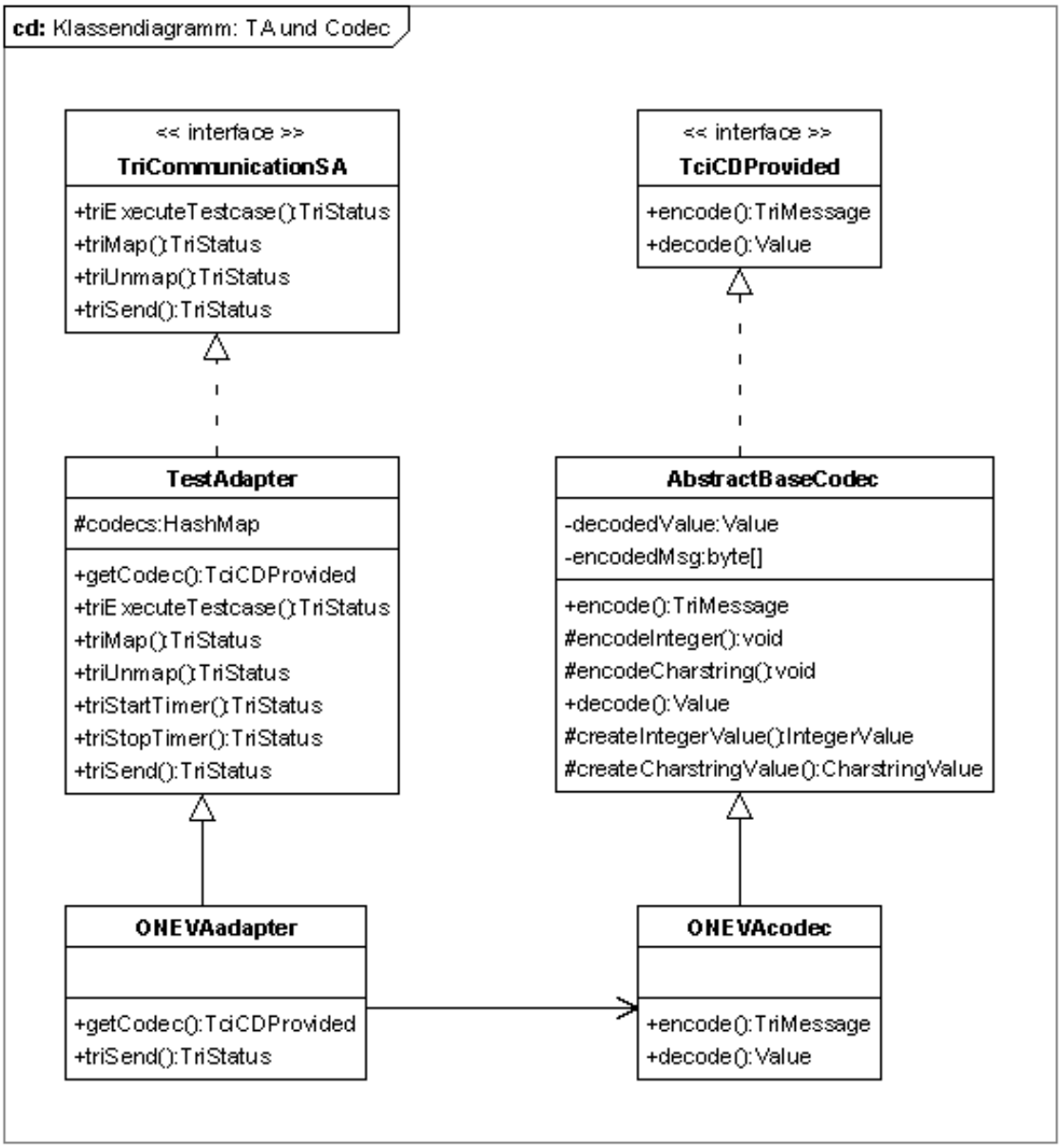


Abbildung 24: Generische und selbsterstellte Testadapter- und Codec-Klassen

Die Methoden des Testadapters werden vom Framework automatisch zu richtigen Zeitpunkten ausgeführt. Wenn im TTCN3-Code beispielsweise ein Testfall ausgeführt wird, wird die Methode `triExecuteTestcase` aufgerufen.

Viele der (generischen) Methoden des bereitgestellten Adapters können für den Test der ONEVA-Web Services verwendet werden. So werden zum Beispiel die Methoden für die Manipulation des Timers ohne Änderungen benutzt. Lediglich die beiden Methoden `getCodec` und `triSend`, die im ONEVAadapter überschrieben werden, müssen neu implementiert werden.

`getCodec(String encodingName):`

Wie der Name schon sagt, liefert diese Methode einen Codec. Falls nicht anders eingestellt, wird als Parameter immer `null` übergeben. Der Name des Codecs wird als Erstes in der Methode auf den richtigen Wert gesetzt.

```
if ((encodingName == null) || encodingName.equals("")) {
     encodingName = "ONEVAcodec";
}
```

Danach wird ein Codec mit diesem Namen gesucht. Dazu wird die Methode der Oberklasse aufgerufen, die in der Hash-Tabelle `codecs` nachschaut. Bei Erfolg wird der gefundene Codec sofort zurückgegeben.

```
TciCDProvided codec = super.getCodec(encodingName);
if (codec != null) {
     return codec;
}
```

Falls der Codec noch nicht vorhanden ist, wird er neu erstellt und in die Hash-Tabelle hinzugefügt.

```
if (encodingName.equals("ONEVAcodec")) {
     codec = new ONEVAcodec(RB);
     codecs.put(encodingName, codec);
}
return codec;
```

`triSend(..., TriMessage sendMessage)`:

Diese Methode wird beim Senden einer Nachricht aufgerufen. Sie erledigt nicht nur das Weiterleiten der Nachrichten an die Web Services, sondern auch das Zurückgeben der Rückgabewerte an den gerade ausgeführten TTCN3-Testfall.

Der Codec, der vor dieser Methode automatisch aufgerufen wird, codiert die Nachricht, serialisiert sie in ein byte-Array und speichert sie im Objekt `sendMessage`. Das Objekt wird an die Methode `triSend` übergeben, in der die Nachricht wieder deserialisiert und an einen Web Service gesendet wird. Der Name des Web Services wird aus dem Namen der Hauptkomponente und die aufzurufende Funktion aus dem Namen des Ports ermittelt. Im Falle, dass die Nachricht über den Port `clearTable` gesendet wurde, wird der Name der Tabelle aus dem byte-Array gelesen und alle Daten aus der Tabelle direkt über die JDBC-Schnittstelle entfernt.

Falls die aufgerufene Funktion des Web Services einen Rückgabewert liefert, wird dieser wieder in ein byte-Array gesteckt und mit Hilfe der Methode `triEnqueueMsg` in eine Warteschlange eingereiht. Das Framework von TTworkbench ruft wieder automatisch den Codec auf, um den Rückgabewert zu decodieren, damit im TTCN3-Code darauf zugegriffen werden kann.

5.3.2 Codec

Die Methode `encode` bekommt ein Objekt vom Typ `Value` (siehe Kapitel 2.5) und gibt die codierte Nachricht in einem `TriMessage`-Objekt zurück. Da `Value` ein generischer Typ für alle TTCN3-Datentypen ist, werden mittels mehrerer geschachtelter `switch-case`-Blöcke alle TTCN3-Typen der einzelnen Übergabeparameter ermittelt, die in der Nachricht (im `Value`-Objekt) enthalten sind. Bei komplexen Typen wie `record` werden dabei alle Felder durchlaufen und ebenfalls die Typen ermittelt. Anhand des Typs kann der Wert aus dem jeweiligen Parameter z.B. mit `getInt()` oder `getCharstring()` gelesen und gleich in ein byte-Feld serialisiert werden. Am Ende wird das Feld in ein `TriMessage`-Objekt geschrieben und an den Testadapter zurückgegeben.

Die Methode `decode` macht die umgekehrte Prozedur: sie bastelt aus Java-Werten JavaBean-Objekte, die eine Repräsentation für TTCN3-Werte darstellen. Bei einem Rückgabewert eines einfachen Typs wie `boolean` oder `int` wird ein passendes Bean (beispielsweise vom Typ `IntegerValue`) instanziiert, der Wert aus der Nachricht deserialisiert und in das Bean mit z.B. setInt() geschrieben. Das Bean wird schließlich an den Testadapter zurückgegeben.

Viele Rückgabewerte der ONEVA-Web Services sind Java-Vektoren, die Objekte mit Daten aus der Datenbank enthalten. Die Methode `selectUsers` beispielsweise liefert einen Vektor mit User-Objekten, die Passwörter und Namen aller Benutzer enthalten. Ein solcher Vektor wird in eine `record of`-Datenstruktur umgewandelt, die ihrerseits aus `record`-Objekten besteht. Ein solches `record`-Objekt muss die selben Felder haben, wie das zugehörige Datenobjekt. Z.B. muss ein `record`, der für einen User steht, ein `username`- und ein `password`-Feld enthalten. Da es in ONEVA viele solcher Datenobjekte gibt (`User, Lecture, Formtype, Period`, usw.), wird der Typ und die Felder zur Laufzeit mit Hilfe von Java Reflection ermittelt.

5.4 Testausführung und –auswertung

Die Testfälle werden mit TTworkbench ausgeführt (wie in Kapitel 2.5 beschrieben). Die meisten regulären Testfälle laufen fehlerfrei durch. Ein Beispielprotokoll für die Funktion `selectUsers` ist in den Abbildungen 11 und 12 zu sehen. Drei der insgesamt 46 regulären Testfälle laufen mit `fail` durch, verursachen also Fehlerwirkungen (alle logischen Fälle sind im Anhang aufgeführt).

Testfall 6_5:
Hier wird zunächst ein Fragebogentyp erstellt und dessen ID-Nummer zur Erstellung von zwei Lehrveranstaltungen verwendet. Dabei wird die Eigenschaft `enabled` der Veranstaltungen automatisch auf `true` gesetzt (d.h. sie werden im aktuellen Semester angeboten). Mit `toggleLecture` wird eine der Veranstaltungen auf `enabled=false` gesetzt. Schließlich werden alle zur Zeit angebotenen Veranstaltungen aus der Datenbank ausgewählt und es wird auch erwartungsgemäß die mit `enabled=true` zurückgegeben. Allerdings ist ihr `enabled`-Wert auf `false` gesetzt, was die Fehlerwirkung herbeiführt. Abbildung 25 zeigt einen Teil des grafischen Protokolls, in dem der Fehler zu sehen ist.

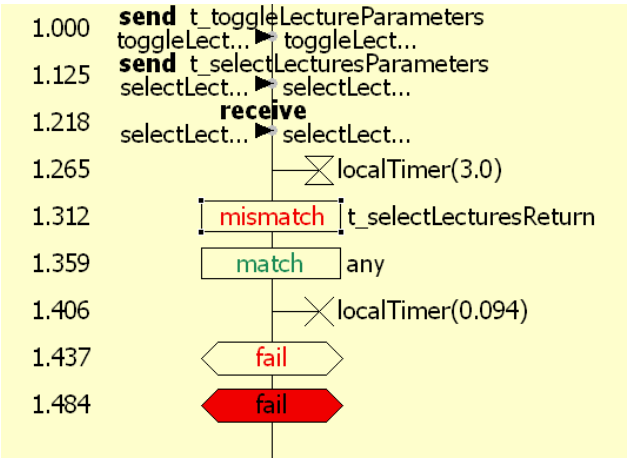


Abbildung 25: Eine nicht erwartete Nachricht verursacht eine Fehlerwirkung

Im `Test Data View` werden die erwartete und die tatsächlich empfangene Nachrichten genau angegeben.

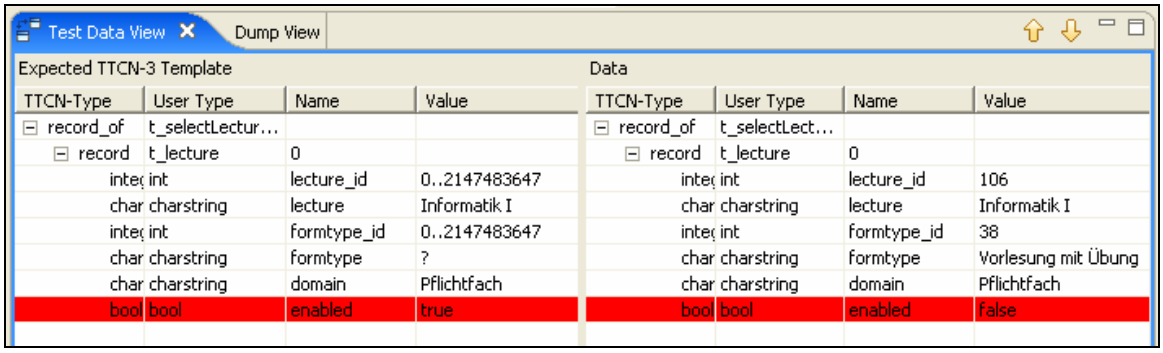


Abbildung 26: Die erwartete und die empfangene Nachricht bei Testfall 6_5

Bei Betrachtung des Programmcodes kann man feststellen, dass der Wert von `enabled` gar nicht gesetzt wird, `false` also als `default`-Wert übergeben wird. Der Wert wird nur dann gesetzt, wenn alle und nicht nur die „eingeschalteten“ Vorlesungen ausgewählt werden. Bei „eingeschalteten“ Vorlesungen ist eigentlich klar, dass der Wert auf `true` gesetzt sein muss und wird wahrscheinlich deswegen ignoriert. Es ist also kein Fehler im eigentlichen Sinne, vorausgesetzt der `enabled`-Wert wird nicht im Programm verwendet. Vorsichtshalber wird dies jedoch korrigiert, so dass immer der richtige Wert zurückgegeben wird.

Testfall 7_4:
Es wird ein Fragebogentyp mit einer Frage erstellt. Nachdem die Frage gelöscht wird, sollte die Funktion `selectQuestions` einen leeren Vector zurückgeben. Allerdings wird ein Vector empfangen, der die Frage immer noch enthält.

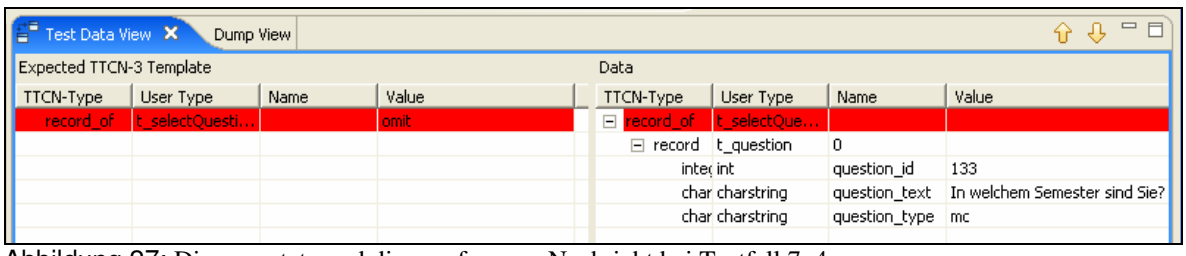


Abbildung 27: Die erwartete und die empfangene Nachricht bei Testfall 7_4

Die Methode `deleteQuestion` im `FDBService` prüft, ob bereits Antworten auf diese Frage in der Datenbank gespeichert sind. Falls nicht, soll die Frage gelöscht werden. Durch einen logischen Fehler im Code (es fehlt ein !, logisches *nicht*) wird dies genau umgekehrt gehandhabt.

Testfall 8_1:
Wenn die Funktion `selectPeriods` aufgerufen wird, während die Datenbanktabelle `periods` leer ist, wird eine Ausnahme verursacht und es kommt keine Nachricht an. Der Timer läuft nach drei Sekunden ab und der Testfall endet mit dem Status `fail`.

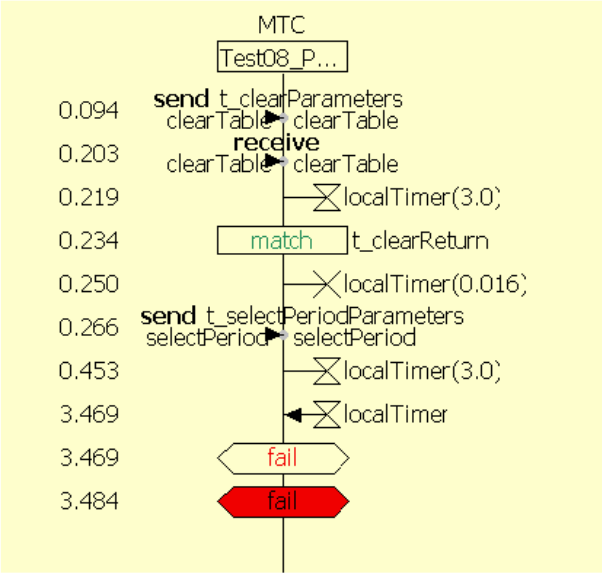


Abbildung 28: Ein Timeout wird verursacht, wenn keine Nachricht ankommt

Der Fehler wird dadurch verursacht, dass ein leeres `Period`-Objekt zurückgegeben wird. Die Attribute des Objekts sind auf `null` gesetzt, wodurch eine `SAXParserException` ausgelöst wird. Wenn man die Werte im Code explizit auf Standardwerte setzt (z.B. eine Zeichenkette auf einen leeren String), tritt der Fehler nicht mehr auf.

Irreguläre Testfälle (siehe Kap. 5.2.3):
Von insgesamt 70 irregulären Testfällen deckten 11 davon Fehlerwirkungen auf. 8 sind bei der Funktion `addUser` und 3 bei der Funktion `deleteUser` aufgetreten. Folgende Zeichenketten verursachten Fehlerwirkungen:

- „`\\`“: bei `addUser` und `deleteUser`[3]
- „`\\\\`“: bei `addUser`
- „`\r`“: bei `addUser`
- „`\d`“: bei `addUser`
- „`'`“(Hochkomma): bei `addUser` und `deleteUser`[3]
- „`€`“: bei `addUser`
- `null`: bei `addUser` und `deleteUser`

[3] In diesem Fall wurde ein Fehler durch einen Timeout oder eine falsche Nachricht bereits bei `addUser` ausgelöst und nichts in der Datenbank gespeichert. Damit konnte `deleteUser` nicht korrekt getestet werden.

- Zeichenkette mit mehr als 30 Zeichen: bei `addUser`[3]

Zusammenfassend lässt sich sagen, dass vor Allem bei Fluchtsymbolen mit einem Backslash Vorsicht geboten ist, weil sie besondere Bedeutungen haben. Das Hochkomma bereitet wahrscheinlich deswegen Probleme, weil es in SQL für Zeichenketten-Definitionen verwendet wird. Fehlerquellen sind ebenfalls (wie erwartet) Null-Werte und Strings, die länger sind als in der Datenbanktabelle definiert.

6. Fazit und Ausblick

Das Online-Evaluationssystem ist eine komplexe und ausgereifte Web-Anwendung. Die integrierten Web Services bieten einen robusten Zugriff auf die im Hintergrund liegende Datenbank. Natürlich kann man durch Tests nie alle Fehlerwirkungen finden und dadurch ein vollkommen fehlerfreies System erreichen. Dennoch haben die durchgeführten Tests gezeigt, dass die Web Services zumindest sehr fehlerarm sind. Zwei von den drei gefundenen Fehlerwirkungen wurden von nicht besonders gefährlichen Fehlerzuständen hervorgerufen, denn die Testzusammenstellungen waren nicht alltagsrelevant. Das Attribut `enabled` im Testfall 6_5 wird in diesem Zusammenhang wahrscheinlich gar nicht verwendet und die Tabelle `periods` aus dem Testfall 7_4 ist bereits mit Werten belegt, so dass dieser Fehler auch nicht auftreten würde. (Nichtsdestotrotz wurden die Fehlerzustände korrigiert.)

Das Problem mit den Sonderzeichen kann in den Web Services nicht gelöst werden. Wünschenswert wäre hier eine Filterfunktion in der Benutzeroberfläche, die es den Benutzern verbieten würde, unerlaubte Sonderzeichen und Fluchtsymbole einzugeben.

Durch den Einbau der Sicherheitsfunktion mit versteckten Passwörtern sind die Web Services sicher genug, um tatsächlich betrieben zu werden. Bevor die ganze Anwendung in Betrieb genommen wird, ist allerdings ratsam, auch die beiden oberen Schichten (*view* und *controller*) ähnlich gründlich zu testen.

Anhang

Spezifikation der Web Services

Im Folgenden werden die drei Web Services (ADBService, FDBService, FDBtoADBService), die bei ONEVA zum Zugriff auf die Datenbanken benutzt werden, spezifiziert. Diese formalen Beschreibungen werden als Testbasis für die Testfälle benötigt.

ADBService

Funktion	***Eingabeparameter***	***Rückgabewert***	***Benutzer***	***Beschreibung***
login	username: String password: String	int 1=Login als Admin 0=Login als Dozent -1=fehlgeschlagen	alle	prüft die eingegebenen Benutzerdaten
selectUsers	passwd: String	Vector (Liste aller Benutzer)	Administrator	holt alle Benutzer (Admins und Dozenten) aus der ADB
addUser	username: String password: String is_admin: Boolean passwd: String	boolean true=erfolgreich false=sonst	Administrator	speichert neue Benutzerdaten ab
saveUser	username: String password: String is_admin: Boolean passwd: String	-	Administrator	aktualisiert Benutzerdaten
deleteUser	username:String passwd: String	-	Administrator	löscht Benutzerdaten
addUserLecture	username: String lecture_id: String passwd: String	-	Administrator	erstellt eine neue Zuweisung zw. Dozent und Veranstaltung
removeUserLecture	username: String lecture_id: String passwd: String	-	Administrator	entfernt eine Zuweisung
addStudent	matr_no: Integer email: String pin: Integer passwd: String	boolean true=erfolgreich false=sonst	Administrator Student	fügt neue Studentendaten ein
checkDuplicatePIN	pin: Integer passwd: String	boolean true=PIN eindeutig false=PIN vorhanden	Administrator Student	prüft, ob eine PIN noch nicht vorhanden ist
PIN4Lecture	pin: String lecture_id: String passwd: String	boolean true=gespeichert false=Fehler	Administrato Student	speichert die Zuordnung PIN<>Lecture
selectLectures	username: String inverted: Boolean passwd: String	Vector (Liste mit Lehrveranstaltungen)	Administrator Dozent Student	holt alle Veranstaltungen
closePeriod	passwd: String	-	Administrator	schließt die letzte Periode

Tabelle 1: Web Service zum Zugriff auf die Authorisierungsdatenbank

Hinweis:Der Parameter „passwd“, der an alle Funktionen außer „login“ übergeben wird, wird zur Authorisierung der jeweiligen WS-Funktion benutzt. Anhand dieses Passworts wird ermittelt, ob der aktuelle Benutzer (Administrator, Dozent oder Student) die Funktion aufrufen darf oder nicht. Außerdem wird dadurch ein Schutz der Web Services gewährleistet, die bei Kenntnis der richtigen URLs leicht missbraucht werden könnten.

FDBService

Funktion	***Eingabeparameter***	***Rückgabewert***	***Benutzer***	***Beschreibung***
selectLectures	enabled: Boolean passwd: String	Vector (Lehrveranst.)	Administrator Dozent Student	holt aktuelle Lehrveranstaltungen aus der FDB
addLecture	lecture: String formtype_id: String domain: String passwd: String	int (ID der Veranstaltung) -1=Fehler	Administrator	fügt eine neue Veranstaltung ein
saveLecture	lecture_id: String lecture: String formtype_id: String domain: String passwd: String	-	Administrator	aktualisiert die Daten einer Veranstaltung
toggleLecture	lecture_id: String enabled: Boolean passwd: String	-	Administrator	schaltet eine Veranstaltung ein oder aus
deleteLecture	lecture: String passwd: String	-	Administrator	entfernt eine Lehrveranstaltung
selectFormtypes	passwd: String	Vector (alle Fragebogen-Typen)	Administrator Dozent Student	holt alle Fragebogentypen
addFormtype	formtype: String passwd: String	int (ID) -1=Fehler	Administrator	fügt einen Fragebogentyp ein
saveFormtype	formtype_id: String passwd: String	-	Administrator	aktualisiert die Daten eines Fragebogentyps
deleteFormtype	formtype_id: String passwd: String	-	Administrator	löscht einen Fragebogentyp
selectQuestions	formtype_id: String inverted: String passwd: String	Vector (alle Fragen)	Administrator Dozent Student	holt alle Fragen und Überschriften
addQuestion	formtype_id: String question_text: String question_type: String passwd: String	int (ID der Frage) -1=Fehler	Administrator	fügt eine neue Frage zu einem Fragebogentyp hinzu

saveTextQuestion	question_id: String question_text: String passwd: String	-	Administrator	aktualisiert den Text einer Textfrage
saveChoiceQuestion	question_id: String question_text: String choices: String[] passwd: String	-	Administrator	aktualisiert eine Auswahlfrage
saveRangeQuestion	question_id: String question_text: String start_text: String end_text: String range: String passwd: String	-	Administrator	aktualisiert eine Bereichsfrage
saveDivision	question_id: String question_text: String textarea: Boolean passwd: String	-	Administrator	aktualisiert eine Überschrift
resetQuestions	formtype_id: String questions: Vector passwd: String	-	Administrator	aktualisiert die Fragen in einem Fragebogentyp
copyQuestion	formtype_id: String question_id: String position: String passwd: String	-	Administrator	kopiert eine Frage in einen anderen Fragebogentyp
deleteQuestion	formtype_id: String question_id: Integer passwd: String	-	Administrator	entfernt eine Frage aus enem Fragebogentyp
saveAnswers	lecture_id: String answers: Vector passwd: String	-	Administrator Student	speichert einen ausgefüllten Fragebogen
selectResults	lecture_id: String period: String filters: Vector passwd: String	Vector (alle Evaluations-Ergebnisse)	Administrator	holt die Evaluations-Ergebnisse einer Lehrveranstaltung
selectPeriod	passwd: String	Period (aktuelle Periode)	Administrator Dozent Student	holt die aktuelle Periode
selectPeriods	lecture_id: String passwd: String	Vector (Perioden)	Administrator Dozent Student	holt alle Perioden einer Veranstaltung
openPeriod	name: String passwd: String	-	Administrator	erstellt eine neue Periode
closePeriod	passwd: String	-	Administrator	schließt die akuelle Periode

Tabelle 2: Web Service zum Zugriff auf die Fragebogendatenbank

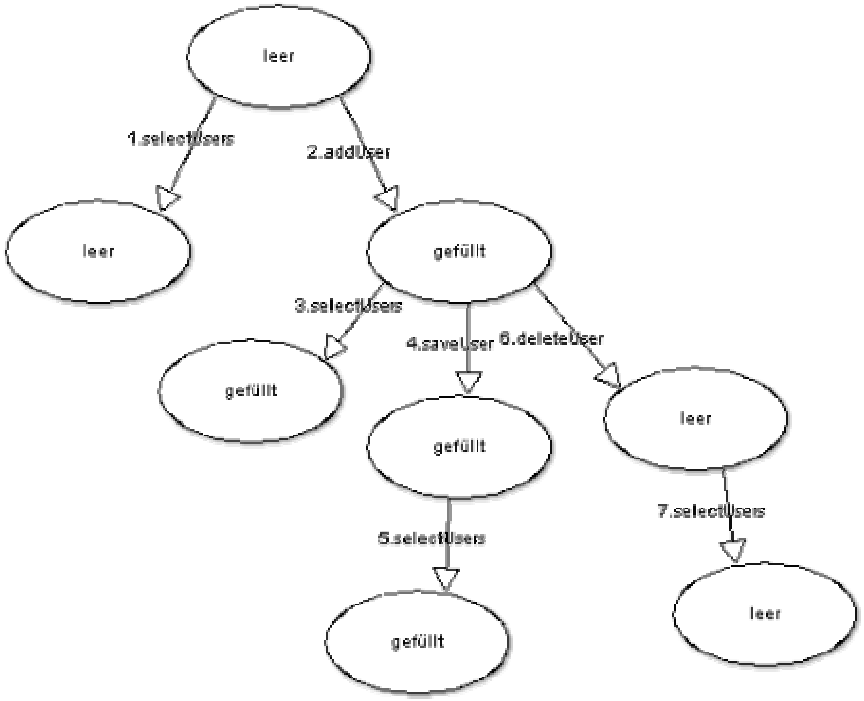
FDBtoADBService

Funktion	*Eingabeparameter*	*Rückgabewert*	*Benutzer*	*Beschreibung*
addLecture	lecture_id: String lecture: String domain: String passwd: String	-	Administrator	fügt eine neue Veranstaltung ein
updateLecture	lecture_id: String lecture: String domain: String passwd: String	-	Administrator	aktualisiert die Daten einer Veranstaltung
toggleLecture	lecture_id: String passwd: String	-	Administrator	schaltet eine Veranstaltung ein oder aus
deleteLecture	lecture_id: String passwd: String	-	Administrator	entfernt eine Lehrveranstaltung

Tabelle 3: Web Service zum Synchronisieren der beiden Datenbanken

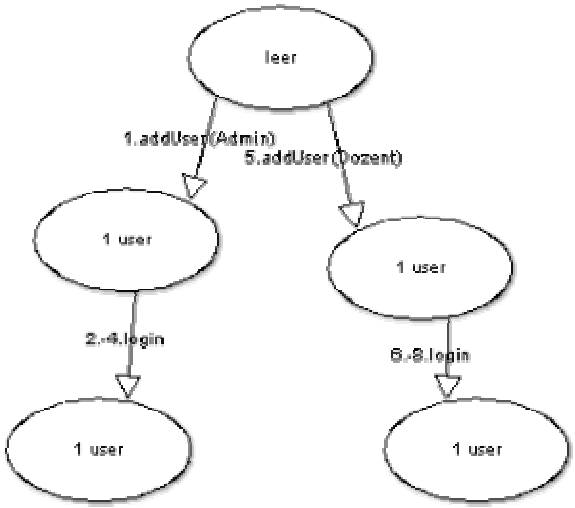
Logische Testfälle

ADBService: addUser, saveUser, selectUsers, deleteUser



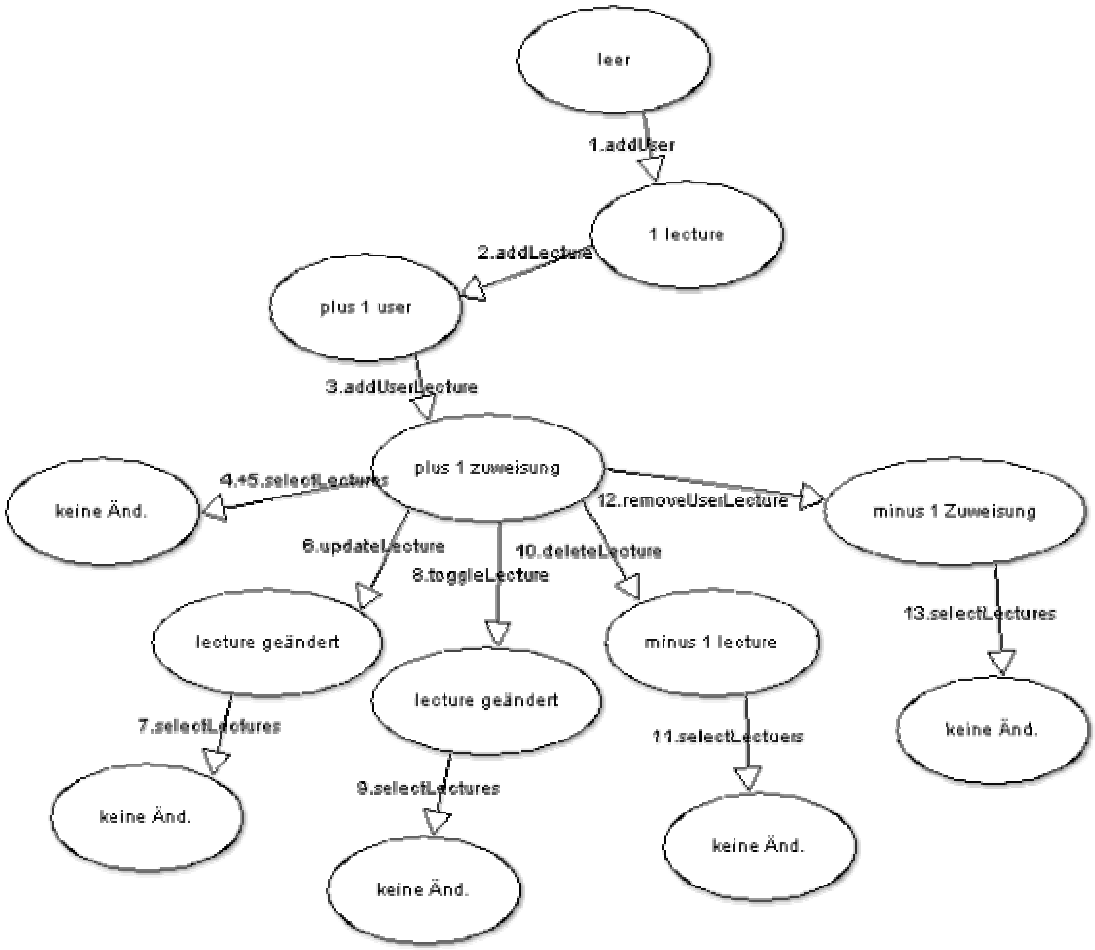
Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
1_1	1.selectUsers	keine	leerer Vector
1-2	2.addUser	reguläre Strings	true
-	3.selectUsers	keine	Vector mit dem eingefügten User-Objekt
1_3	4.saveUser	geänderte Benutzerdaten	keiner
-	5.selectUsers	keine	Vector mit dem geänderten User-Objekt
1_4	6.deleteUser	gespeicherte Benutzerdaten	keiner
-	7.selectUsers	keine	leerer Vector

ADBService: login



Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
2_1	1.addUser	reguläre Strings is_admin = true	true
-	2.login	Userdaten von 1.	1
2_2	4.login	falscher Username	-1
2_3	4.login	falsches Passwort	-1
2_4	5.addUser	reguläre Strings is_admin = false	true
-	6.login	Userdaten von 5.	0
2_5	7.login	falscher Username	-1
2_6	8.login	falsches Passwort	-1

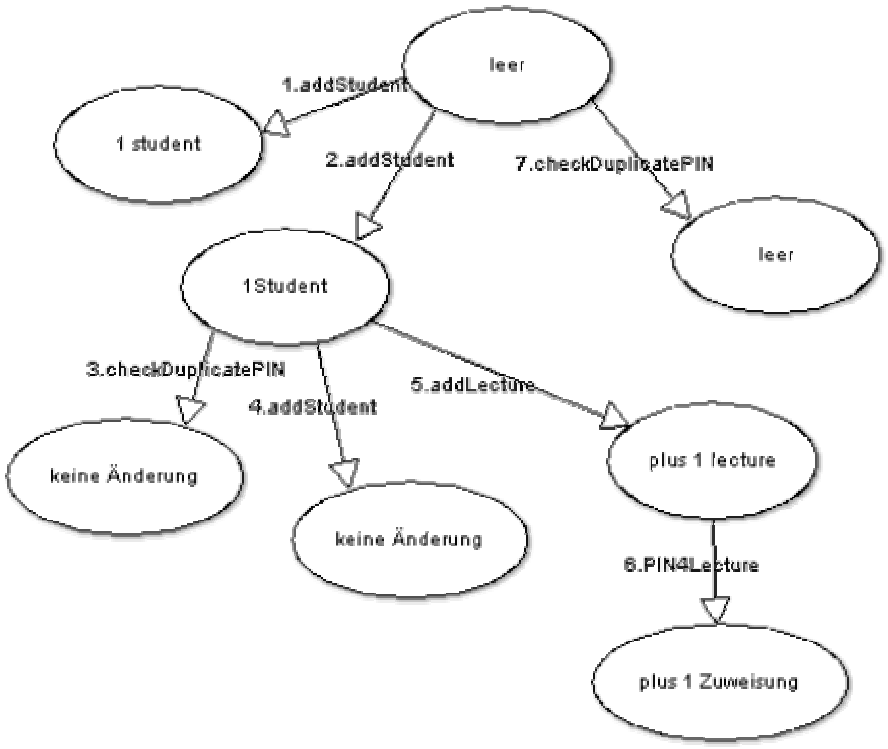
FDBtoADBService: addLecture, updateLecture, toggleLecture, deleteLecture
ADBService: selectLectures, addUserLecture, removeUserLecture



Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
3_1	1.addUser	reguläre Strings	true
-	2.addLecture	pos. Integer, reguläre Strings	-
-	3.addUserLecture	lecture_id von 2., username von 1.	-
-	4.selectLectures	username von 1., inverted=false	Vector mit einem Lecture-Objekt
3_2	5.selectLectures	username von 1.,	leerer Vector

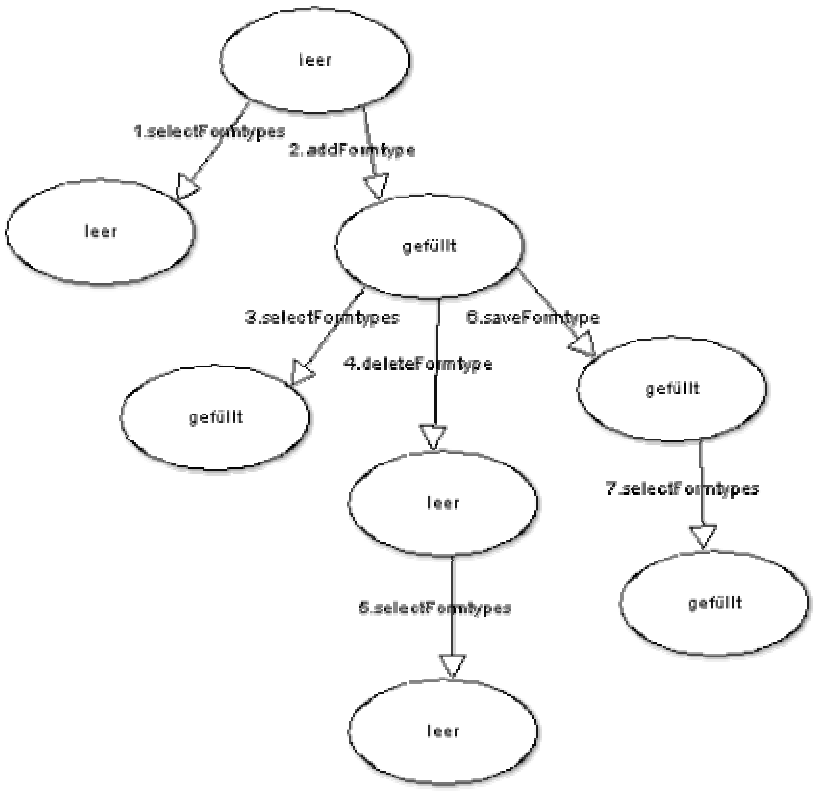
		inverted=true	
3_3	6.updateLecture	lecture_id von 2., geänderte Strings	-
-	7.selectLectures	username von 1., inverted=false	Vector mit einem geänderten Lecture-Objekt
3_4	8.toggleLecture	lecture_id von 2.	-
-	9.selectLectures	username von 1., inverted=false	Vector mit einem geänderten Lecture-Objekt
3_5	10.deleteLecture	lecture_id von 2.	-
-	11.selectLectures	username von 1., inverted=false	leerer Vector
3_6	12.removeUserLecture	lecture_id von 2., username von 1.	-
-	13.selectLectures	username von 1., inverted=false	leerer Vector

ADBService: saveStudent, checkDuplicatePIN, PIN4Lecture



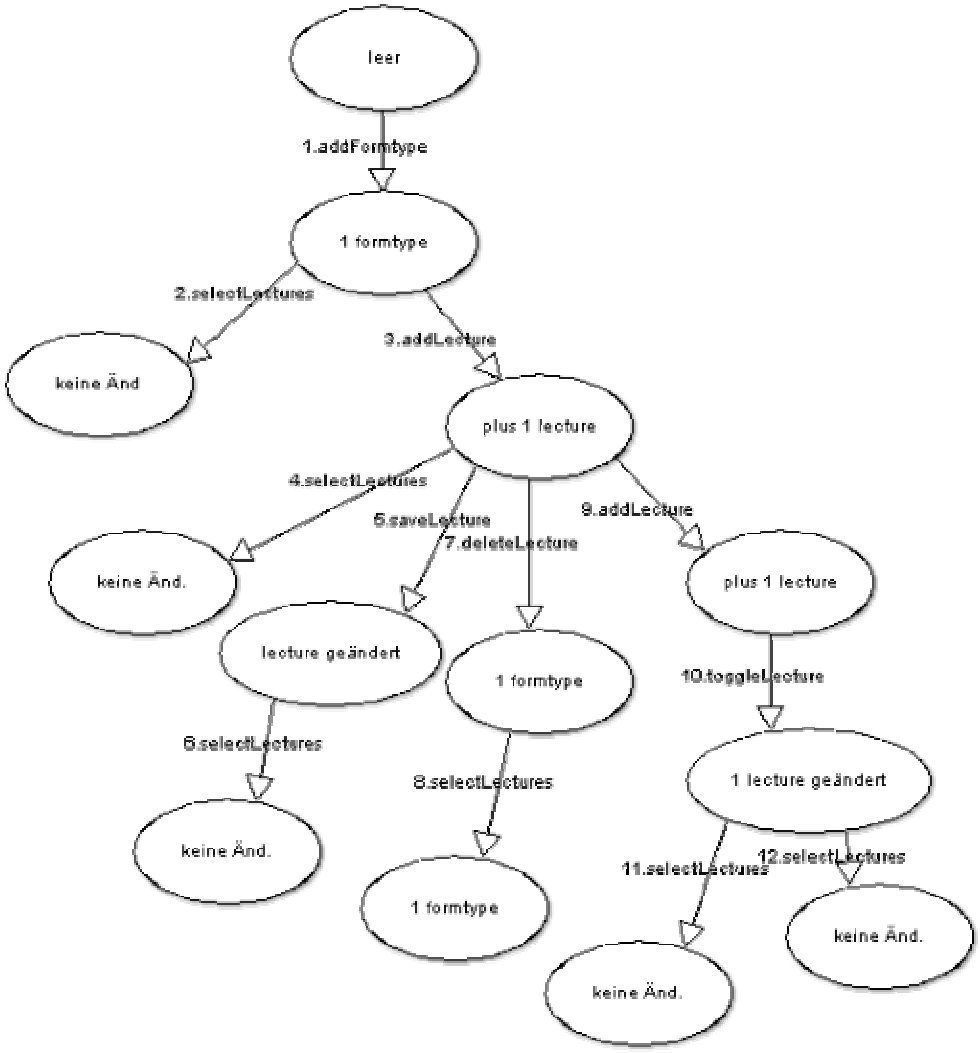
Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
4_1	1.addStudent	reguläre Strings, pos. Integer	true
4_2	2.addStudent	reguläre Strings, pos. Integer	true
-	3.checkDuplicatePIN	pin von 2.	false
4_3	4.addStudent	von 2.	false
4_4	5.addLecture	reguläre Strings	-
-	6.PIN4Lecture	pin von 2., lecture_id von 5.	true
4_5	7.checkDuplicatePIN	pos. Integer	true

FDBService: addFormtype, selectFormtypes, saveFormtype, deleteFormtype



Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
5_1	1.selectFormtypes	-	leerer Vector
5_2	2.addFormtype	Buchstaben/Zahlen	pos.Integer
-	3.selectFormtypes	-	Vector mit einem Formtype-Objekt
5_3	4.saveFormtype	geänderte Benutzerdaten	-
-	5.selectFormtypes	-	Vector mit einem geänderten Formtype-Objekt
5_4	6.deleteFormtype	gespeicherte Daten	-
-	7.selectFormtypes	-	leerer Vector

FDBService: addLecture, selectLectures, saveLecture, toggleLecture, deleteLecture

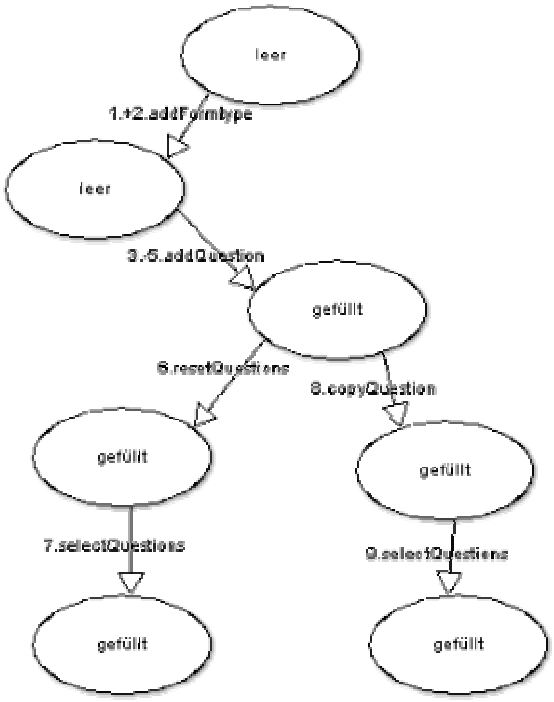


Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
6_1	1.addFormtype	reguläre Buchstaben/Zahlen	pos. Integer
-	2.selectLectures	enabled=false	leerer Vector
6_2	3.addLecture	passende Strings	positiver Integer
-	4.selectLectures	enabled=false	Vector mit einem Lecture-Objekt
6_3	5.saveLecture	geänderte Daten	-
-	6.selectLectures	enabled=false	Vector mit einem Lecture-Objekt
6_4	7.deleteLecture	gespeicherte Daten	-
-	8.selectLectures	enabled=false	leerer Vector
6_5	9.addLecture	neue Daten	positiver Integer
-	10.toggleLecture	gespeicherte Lecture enabled=false	-
-	11.selectLectures	enabled=true	Vector mit einem Lecture-Objekt
6_6	12.selectLectures	enabled=false	Vector mit zwei Lecture-Objekten

FDBService: selectQuestions, addQuestion, saveTextQuestion, saveChoiceQuestion, saveRangeQuestion, saveDivision, deleteQuestion, resetQuestions, copyQuestion

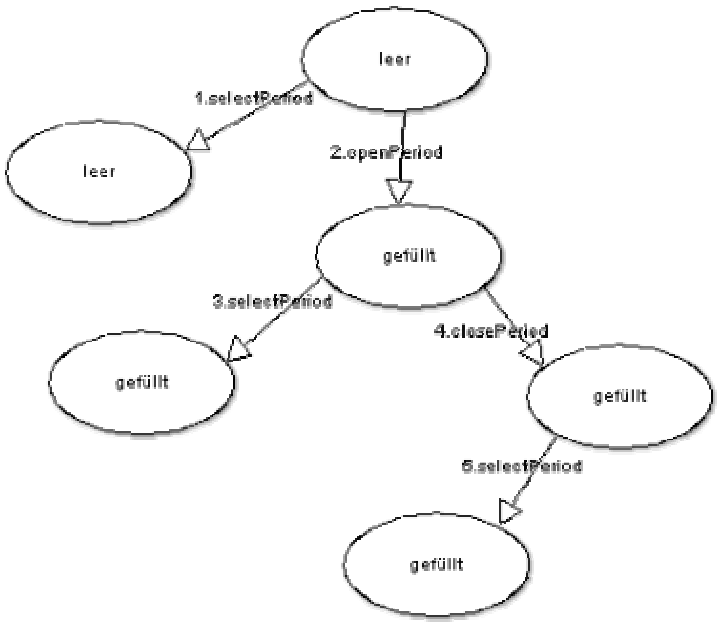
Testfall #	*Methode*	*Parameter*	*erwarteter Rückgabewert*
7_1	1.addFormtype	Buchstaben/Zahlen	pos.Integer (formtype_id)
-	2.selectQuestions	inverted=false	leerer Vector
7_2	3.addQuestion	gespeicherte formtype_id, Buchstaben/Zahlen, Auswahlfrage	pos.Integer (question_id)
-	4.selectQuestions	formtype_id von 1. inverted=false	Vector mit einem Question-Objekt von 3.
7_3	5.addFormtype	verschieden von 1.	pos.Integer (formtype_id)
-	6.addQuestion	formtype_id von 5. Buchstaben/Zahlen, Überschrift	pos.Integer (question_id)
-	7.selectQuestions	formtype_id von 1. inverted=true	Vector mit einem Question-Objekt von 6.
7_4	8.deleteQuestion	formtype_id von 1. question_id von 3.	-
-	9.selectQuestions	formtype_id von 1. inverted=false	leerer Vector
7_5	10.addQuestion	gespeicherte formtype_id,	pos.Integer

		Buchstaben/Zahlen, Textfrage	(question_id)
7_6	11.addQuestion	gespeicherte formtype_id, Buchstaben/Zahlen, Auswahlfrage	pos.Integer (question_id)
7_7	12.addQuestion	gespeicherte formtype_id, Buchstaben/Zahlen, Überschrift	pos.Integer (question_id)
7_8	13.addQuestion	gespeicherte formtype_id, Buchstaben/Zahlen, Bereichsfrage	pos.Integer (question_id)
7_5	14.saveTextQuestion	question_id von 10.	-
7_6	7.saveChoiceQuestion	question_id von 3.	-
7_7	8.saveDivision	question_id von 4.	-
7_8	9.saveRangeQuestion	question_id von 5.	-
7_5 – 7_8	10.-13.selectQuestions	formtype_id von 1. inverted=false	Vector mit einem Question-Objekt



Methode	Parameter	erwarteter Rückgabewert
1.+2.addFormtype	verschiedene Strings	pos. Integer (formtype_id)
3.+4.addQuestion	formtype_id von 1., beliebige Strings	pos. Integer (question_id)
5.addQuestion	formtype_id von 2., beliebige Strings	pos. Integer (question_id)
6.resetQuestions	formtype_id von 1., Rückgabewert von selectQuestions(Fragen getauscht)	-
7.selectQuestions	formtype_id von 1., inverted=false	Vector mit vertauschten Question-Objekten
8.copyQuestion	formtype_id von 2., question_id von 3.	-
9.selectQuestions	formtype_id von 2.	Vector mit zwei Objekten

FDBService: openPeriod, selectPeriod, closePeriod



TestFall #		*Parameter*	*erw. Rückgabewert*
8_1	1.selectPeriod	-	leeres Period-Objekt
8_2	2.openPeriod	regulärer String	-
-	3.selectPeriod	-	Period-Objekt mit open==true
8_3	4.closePeriod	-	-
-	5.selectPeriod	-	Period-Objekt mit open==false

Literaturverzeichnis

[1] Thomas Severiens, „Tutorial: Web Services“ – IuK-Herbsttagung 2003, Bad Honnef, http://www.aki-dpg.de/Dokumente/Bad_Honnef_2003/webservicestutorial.pdf, letzter Aufruf am 18.9.2006

[2] Heiko Wöhr: „Web Technologien: Konzepte – Programmiermodelle – Architekturen“, dpunkt Verlag, 1. Auflage 2004

[3] W3C: Architecture Domain, Extensible Markup Language (XML), http://www.w3.org/XML, letzter Aufruf am 19.9.2006

[4] Andreas Eberhart, Stefan Fischer: Web Services – Grundlagen und praktische Umsetzung mit J2EE und .NET, Carl Hanser Verlag, 2003

[5] Wikimedia Foundation, Wikipedia: SOAP, http://de.wikipedia.org/wiki/SOAP, letzter Aufruf am 19.9.2006

[6] Wikimedia Foundation, Wikipedia: WSDL, http://de.wikipedia.org/wiki/WSDL, letzter Aufruf am 19.9.2006

[7] W3C (World Wide Web Consortium), SOAP Specifications, http://www.w3.org/TR/soap/, letzter Aufruf am 20.9.2006

[8] W3C (World Wide Web Consortium), WSDL 1.1, http://www.w3.org/TR/wsdl, letzter Aufruf am 20.9.2006

[9] Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, dpunkt Verlag, 3. Auflage 2005

[10] Skript zur Vorlesung „Methoden und Verfahren für das Testen von Software“ von Prof. Dr. Jens Grabowski an der Uni Göttingen, Wintersemester 05/06

[11] Ingo Tributh: Erweiterung und Implementation eines Lehrevaluationssystems mit Web Services, Bachelorarbeit im Studienfach „Angewandte Informatik“, 2005

[12] Anke Sawatzki: Design eines Lehrevaluationssystems mit Web Services, Bachelorarbeit im Fach „Angewandte Informatik“, 2004

[13] OASIS, UDDI-Spezifikation, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm, letzter Aufruf am 20.9.2006

[14] TTCN-3 Homepage: Specifications, http://www.ttcn-3.org/Specifications.htm, letzter Aufruf: 21.9.2006

[15] Colin Willcock, Thomas Deiß, Stephan Tobies: An Introduction to TTCN-3, John Wiley & Sons Ltd., 2005

[16] Eclipse Community Page, http://www.eclipse.org/, letzter Aufruf am 22.9.2006

[17] Testing Technologies IST GmbH, http://www.testingtech.de/, letzter Aufruf am 22.9.2006

[18] Sun Microsystems, Inc.: JavaServer Pages Technology, http://java.sun.com/products/jsp/, letzter Aufruf am 25.9.2006

[19] Die deutsche MySQL-Homepage, http://www.mysql.de/, letzter Aufruf am 26.9.2006

[20] Apache Software Foundation: Web Services – Axis, http://ws.apache.org/axis/, letzter Aufruf am 26.9.2006

[21] Apache Software Foundation: Apache Tomcat, http://tomcat.apache.org/, letzter Aufruf am 26.9.2006

[22] Slript zur Vorlesung “Linux-Übung: Grundlagen“ von Dipl.-Math. Stefan Koospal an der Uni Göttingen, Wintersemester 05/06

[23] Apache Software Foundation: Apache Ant Manual, http://ant.apache.org/manual/index.html, letzter Aufruf am 26.9.2006