eTutor Modularchitektur

Christian Eichinger
Michael Karlinger
Georg Nitsche

Version 2
Stand Februar 2006

INSTITUT FÜR WIRTSCHAFTSINFORMATIK
dke
Data & Knowledge Engineering

Inhaltsverzeichnis

Abbildungsverzeichnis

Inhaltsverzeichnis

1. Einleitung

Das eTutor-System ist als web-basierte Anwendung konzipiert, die sich aus einem Kernsystem (eTutor Core) und einer beliebig erweiterbaren Menge von Expertenmodulen zusammensetzt. Während der eTutor Core die grundlegenden Funktionalitäten des eLearning-Systems unterstützt, decken die Modul jeweils ein bestimmtes Themengebiet ab, zu dem Übungsbeispiele spezifiziert und Lösungen zu Übungsbeispielen ausgewertet werden können. Die Schnittstellen für diese Funktionen werden vom eTutor Core vorgegeben und müssen von den Modulen implementiert werden, um in das eTutor-System integriert werden zu können. Zweck dieses Dokumentes ist es, diese Schnittstellen zu beschreiben.

In Kapitel 2 wird ein allgemeiner Überblick über die Architektur des eTutor-Systems gegeben. Der Schwerpunkt liegt dabei in erster Linie auf den Zusammenhängen zwischen dem eTutor Core und den integrierten Modulen.

Kapitel 3 und Kapitel 4 befassen sich mit den für die Integration von Modulen vom eTutor-System vorgegebenen Schnittstellen, mit denen zwei Gruppen von Funktionalitäten abgedeckt werden. In Kapitel 3 wird die Schnittstelle beschrieben, die zur Auswertung von Lösungen dient, die von Studenten zu einem Übungsbeispiel abgegeben werden. Wie die Schnittstelle zur Administration dieser Übungsbeispiele beschaffen ist, wird hingegen in Kapitel 4 beschrieben.

Während in Kapitel 2 die Architektur des eTutor-Systems im allgemeinen beschrieben wird, dient Kapitel 5 zur technischen Erläuterung der Kommunikation zwischen dem eTutor Core und einem in das eTutor-System integrierten Modul. Das eTutor-System sieht für die Integration von Modulen eine Kommunikation über RMI (Remote Method Invocation) vor.

Kapitel 6 fasst die in diesem Dokument erläuterten Schnittstellen zusammen, indem ein Beispiel für die Implementierung eines Moduls beschrieben wird. In diesem Kapitel wird nicht nur die Umsetzung der Schnittstellen verdeutlicht, sondern auch Richtlinien angesprochen, die bei der Implementierung eines

Moduls beachtet werden sollten. Dies betrifft u.a. die interne Strukturierung des Moduls.

2. Architektur des eTutor-Systems

Das eTutor-System ist in einer 3-Schichten Web-Architektur aufgebaut (siehe auch Abbildung 2.1).

- ***Client:*** Das eTutor-System verfolgt den Thin-Client Ansatz. Der Client besteht lediglich aus einem Web-Browser erweitert um ggf. Java-Script Funktionalitäten zur Prüfung von Eingaben.
- ***Web-Server:*** Der Web-Server nimmt Anfragen an das eTutor-System entgegen und leitet diese an die Module weiter. Auf dem Web-Server läuft der Kern des eTutor-Systems (*Core*), welcher die Beispiele der einzelnen Studenten verwaltet. Der Core stellt auch die Administrationsfunktionalität für das System (Bearbeiten von Beispielen, Bearbeiten von Lehrveranstaltungen, Verwaltung von Studenten, etc.) zur Verfügung.
- ***Modul-Host:*** Die Auswertungsmodule (ggf. auf einem eigenen Computer) stellen aufgabenspezifische Auswertungsfunktionalitäten zur Verfügung und verwalten Informationen, welche zur Bearbeitung der einzelnen Beispiele notwendig sind. Außerdem bewerten die Auswertungsmodule die Abgaben und stellen das Feedback zu den Abgaben bereit.

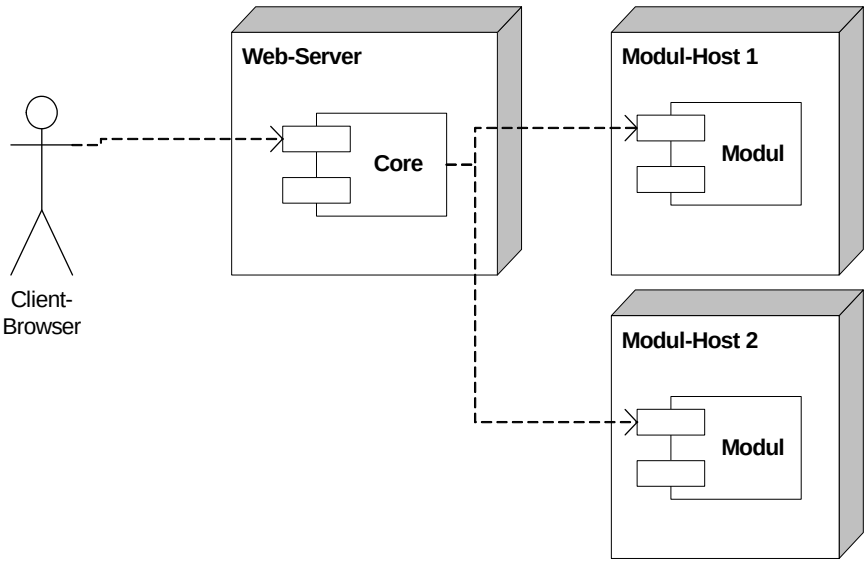

Abbildung 2.1: Überblick über die eTutor-Systemarchitektur

Um in den eTutor integriert werden zu können muss jedes Modul bestimmte Komponenten und Views zur Verfügung stellen, die die hier beschriebenen Funktionalitäten unterstützen. Als Komponenten werden hier Java-Klassen bezeichnet, die die vorgegebenen Schnittstellen für die Integration in das eTutor-System implementieren. Views können durch Java-Servlets, JSPs (Java Server Pages) oder einfache HTML-Seiten realisiert werden.

- ***Auswertungskomponente:*** Diese Komponente implementiert das vom eTutor-System vorgegebene Interface `etutor.core.evaluation.Evaluator`. Sie stellt Funktionalitäten des Moduls für Analyse, Diagnose und Feedback von Übungsbeispielen, die von Studenten ausgearbeitet werden, zur Verfügung.
- ***Studenten-Views:*** Dies umfasst (1) eine Eingabemaske, welche es Studenten ermöglicht, eine Abgabe zu einer Übungsaufgabe zu erfassen, (2) eine View für die Anzeige des Auswertungsergebnisses zu einer Abgabe, und (3) eine View für die Anzeige einer bereits erfolgten Abgabe.
- ***Administrationskomponente:*** Diese implementiert das vom eTutor-System vorgegebene Interface `etutor.core.manager.ModuleExerciseManager`. Über diese Komponente kommuniziert der eTutor Core mit dem Modul, um modulspezifische Informationen zu Beispielen erstellen, ändern, löschen und abrufen zu können. Die Administration von Beispielen kann nur durch Benutzer des eTutor-Systems durchgeführt werden, die eine Assistentenrolle besitzen.

- ***Assistenten-Views:*** Eine oder mehrere Eingabemasken, welche die Eingabe bzw. Änderung von modulspezifischen Informationen zu einem Übungsbeispiel durch Benutzer mit Assistentenrolle ermöglichen. Dies betrifft Informationen wie etwa die Musterlösung.

Betrachtet man die Architektur eines Moduls genauer, ergibt sich dazu das in Abbildung 2.2 gezeigte Bild.

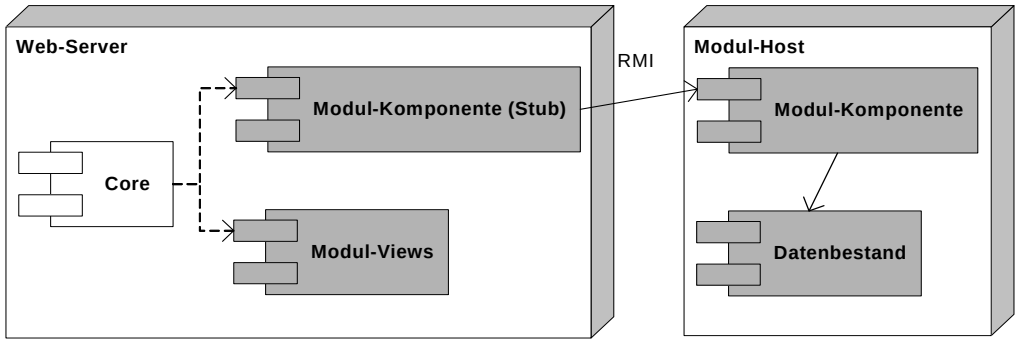

Abbildung 2.2: Modul-Architektur

Die grau markierten Komponenten werden vom jeweiligen Modul für die Integration in das eTutor-System bereitgestellt. Die Komponenten am Web-Server dienen als Schnittstelle zu den am Modul-Host liegenden Komponenten. Dies gilt gleichermaßen für die Auswertungskomponente („Evaluator“) und die Administrationskomponente („ModuleExerciseManager“). Für den eTutor Core ist lediglich dieses „Schnittstellenobjekt“ sichtbar, die Verteilung der Komponente ist für den Core daher transparent. Die Kommunikation zwischen den Komponenten am Web-Server und den Modulkomponenten ist hier über RMI (Remote Method Invocation – der Java RPC-Mechanismus) dargestellt. Die Entscheidung, die Modulkomponenten innerhalb des Web-Containers zu installieren, ist zwar zulässig, das Modul sollte jedoch im Idealfall den RMI-Mechanismus unterstützen (siehe Kapitel 5). Im Falle einer Integration des Moduls über RMI gibt es für jede Komponente ein lokal verfügbares Schnittstellenobjekt, das als Stub bezeichnet wird. Der Stub ist mit der entsprechenden Komponente des Moduls verbunden, das sich bei einer Kommunikation über RMI üblicherweise auf einem anderen Rechner befindet. Der modulspezifische Datenbestand wird vom Modul verwaltet. Die Schnittstellen zur Auswertungs- und zur Administrationskomponente werden in Kapitel 3 und in Kapitel 4 genauer beschrieben.

Abgesehen von gewissen Richtlinien für die interne Strukturierung des Moduls (siehe Kapitel 6) kann der Aufbau des Moduls frei gewählt werden. Es ist lediglich sicherzustellen, dass aufgabenspezifisch erfasste Informationen eindeutig

abrufbar sind, d.h. wenn über den eTutor Core die Lösung eines Beispiels angefordert wird, z.B. Beispiel 1, dann sollte immer derselbe modulspezifische Datenbestand aktiviert werden. Beispiel: Das SQL-Modul verwaltet zu jedem Beispiel die korrekte Ergebnisquery. Lautet die Aufgabenstellung zu Beispiel 1 „finden Sie alle Produkte mit einem Preis größer 500“ dann darf nur die für dieses Beispiel angegebene Ergebnisquery verwendet werden. Im SQL-Modul wird dies über eine Mapping-Tabelle in einer Datenbank realisiert.

Informationen welche allgemein für die Funktionstüchtigkeit eines Moduls notwendig sind, können z.B. über Konfigurationsdateien festgelegt werden. Das SQL-Modul speichert z.B. die Verbindungsinformationen zu den Beispieldatenbanken in einer eigenen Konfigurationsdatei. In der Mapping-Tabelle wird daher zusätzlich vermerkt, auf welcher Datenbank ein Beispiel ausgeführt werden soll. Diese Informationen können auch für die Eingabe von Beispielen relevant sein: In der Eingabemaske zu SQL kann der LV-Leiter nur jene DB-Verbindungen für ein Beispiel auswählen, welche auch für das Modul konfiguriert wurden.

3. Beispielauswertung

Die Beispielauswertung umfasst die Analyse einer Studentenabgabe, die Bewertung der Studentenabgabe und die Erstellung eines Feedbacks. In Abschnitt 3.1 folgt eine umfassende Darstellung des Zusammenspiels zwischen eTutor Core und der Auswertungskomponente eines Moduls. Das Interface, das für die Beispielauswertung vom eTutor Core vorgegeben wird und vom Modul implementiert werden muss, wird im Detail in Abschnitt 3.2 erläutert.

3.1. Ablauf der Beispielauswertung

Der Ablauf bei der Beispielauswertung, der in Abbildung 3.1 und Abbildung 3.2 in Form von Sequenzdiagrammen dargestellt ist, zeigt, dass für die modulspezifische Auswertung von Beispielen einerseits Views und andererseits Komponenten benötigt werden. In den Abbildungen wird davon ausgegangen, dass die Views als Servlet implementiert sind. Wie bereits aus Abbildung 2.2 ersichtlich ist, sind die Views für den eTutor Core lokal verfügbar, während das Modul mit der Auswertungskomponente (evaluator) auf einem separaten Rechner liegen kann. Die Kommunikation zwischen den modulspezifischen Views und den jeweiligen Modulkomponenten erfolgt nicht direkt sondern wird über den eTutor Core abgewickelt. Bei der Beispielauswertung nimmt diese Vermittlungsrolle das `CoreManagerServlet` (coreMgrServlet) ein, wie in den nachfolgenden Absätzen noch näher erläutert wird.

Ein Modul wird aktiviert, wenn der Student den Link einer Aufgabe (task) auswählt. Im ersten Schritt werden die für die Initialisierung des Auswertungsprozesses erforderlichen Parameter an ein Servlet des eTutor Core übergeben (`taskViewerServlet`). Die Parameter dienen zur Identifikation der Aufgabe (taskID), des Beispieltyps (typeID), des Beispiels (exerciseID) und des Ausführungsmodus' (modeID).

Der wichtigste Parameter betrifft dabei die Identifikation der Aufgabe (taskID), da sich alle übrigen Parameter daraus ableiten lassen. Als Aufgabe (task) wird die Zuteilung eines Beispiels (exercise) zu einem Studenten innerhalb einer LVA bezeichnet. Die Unterscheidung zwischen Beispiel (exercise) und Aufgabe (task) ist notwendig, weil ein Beispiel in der eTutor Datenbank unabhängig von Studenten in einem Beispiel-Pool verwaltet wird. Die Aufgabe wird vor allem innerhalb des eTutor Core, etwa zur Verwaltung des Lernfortschrittes, verwendet.

Der Beispieltyp (typeID), der zu einem Beispiel vermerkt ist, wird vom eTutor Core verwendet, um das Modul und die modulspezifischen Ressourcen für die Beispielauswertung zu identifizieren.

Ein weiterer Parameter, der sich aus der Aufgabe ableiten lässt, ist der Ausführungsmodus (modeID). Im eTutor-System können verschiedene Ausführungsmodi definiert werden, im wesentlichen werden Aufgaben entweder im Übungs- oder im Abgabemodus bearbeitet. Der Modus gibt unter anderem Aufschluss darüber, welche Aktionen dem Benutzer bei der Beispielausarbeitung zur Verfügung stehen sollen. Die Menge an möglichen Aktionen umfasst das bloße Ausführen einer Lösung ohne jegliche Fehleranalyse (run), die Ausführung mit einem kurzen Feedback, ob die Lösung korrekt oder nicht korrekt ist (check), die Diagnose von Fehlern (diagnose) und die Abgabe und Bewertung einer Lösung (submit). So soll etwa die Abgabe und Bewertung (submit) nur im Abgabemodus möglich sein, im Übungsmodus hingegen nicht. Je nach Ausführungsmodus ermittelt der eTutor-Core die Menge der zulässigen Aktionen (actions) und übermittelt sie an das Modul, das in der Lage sein sollte, die Eingabemaske entsprechend anzupassen.

Alle hier beschriebenen Parameter werden als Attribute in der Benutzer-Session gespeichert, so dass sie im folgenden Schritt für die modulspezifische Eingabemaske, die anhand des Beispieltyps (typeID) identifiziert wird, verfügbar sind. Die Ablaufkontrolle liegt ab diesem Zeitpunkt in der Verantwortlichkeit des Moduls. Über die Maske bewerkstelligt der Student die Eingabe der Lösung zu dem angegebenen Beispiel, wobei sich die Eingabe über mehrere Benutzerinteraktionen erstrecken kann.

Wählt der Student eine der zur Verfügung stehenden Aktionen, um die Lösung auswerten zu lassen, so werden modulspezifische Informationen sowie vom eTutor Core benötigte Informationen, die noch nicht bereits als Parameter im Request enthalten sind, als Attribute in der Session oder im Request gespeichert,

so dass sie in den nachfolgenden Schritten sowohl für den eTutor Core als auch für das Modul selbst verfügbar sind (submissionInfo). Zu den modulspezifischen Informationen zählen hier jegliche Informationen, die das Modul selbst in den folgenden Schritten für die Analyse, die Bewertung und das Feedback benötigt, wie etwa die Lösung des Studenten. Das `CoreManagerServlet` benötigt Informationen über die ausgeführte Aktion (action). Dieser Parameter kann einen der oben angeführten Werte haben und wird vom eTutor Core u.a. verwendet, um zu entscheiden, ob eine Bewertung der Abgabe durchgeführt werden soll.

Das `CoreManagerServlet` ist ab diesem Zeitpunkt für die Transaktion verantwortlich, die die Analyse, die Bewertung und die Anzeige des Feedbacks umfasst. Der Beispieltyp (typeID) wird hier wiederum verwendet, um die modulspezifische Klasse für die Beispielauswertung zu laden (evaluator).

Nach Aufruf der Methoden für Analyse, Bewertung und Feedback wird das im letzten Schritt generierte Report-Objekt als Session-Attribut gespeichert, so dass es für die modulspezifische Ressource zur Visualisierung der Ergebnisse (reporterServlet) verfügbar ist.

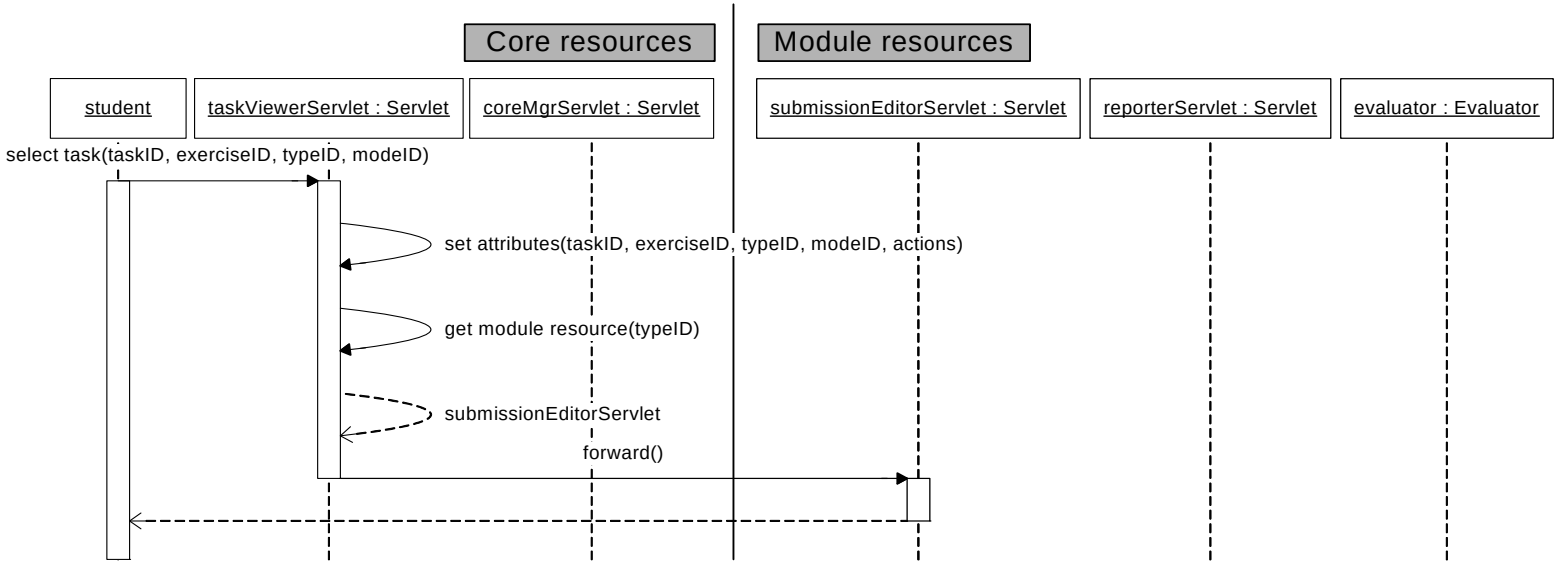

Abbildung 3.1: Ablauf der Beispielauswertung (Teil 1)

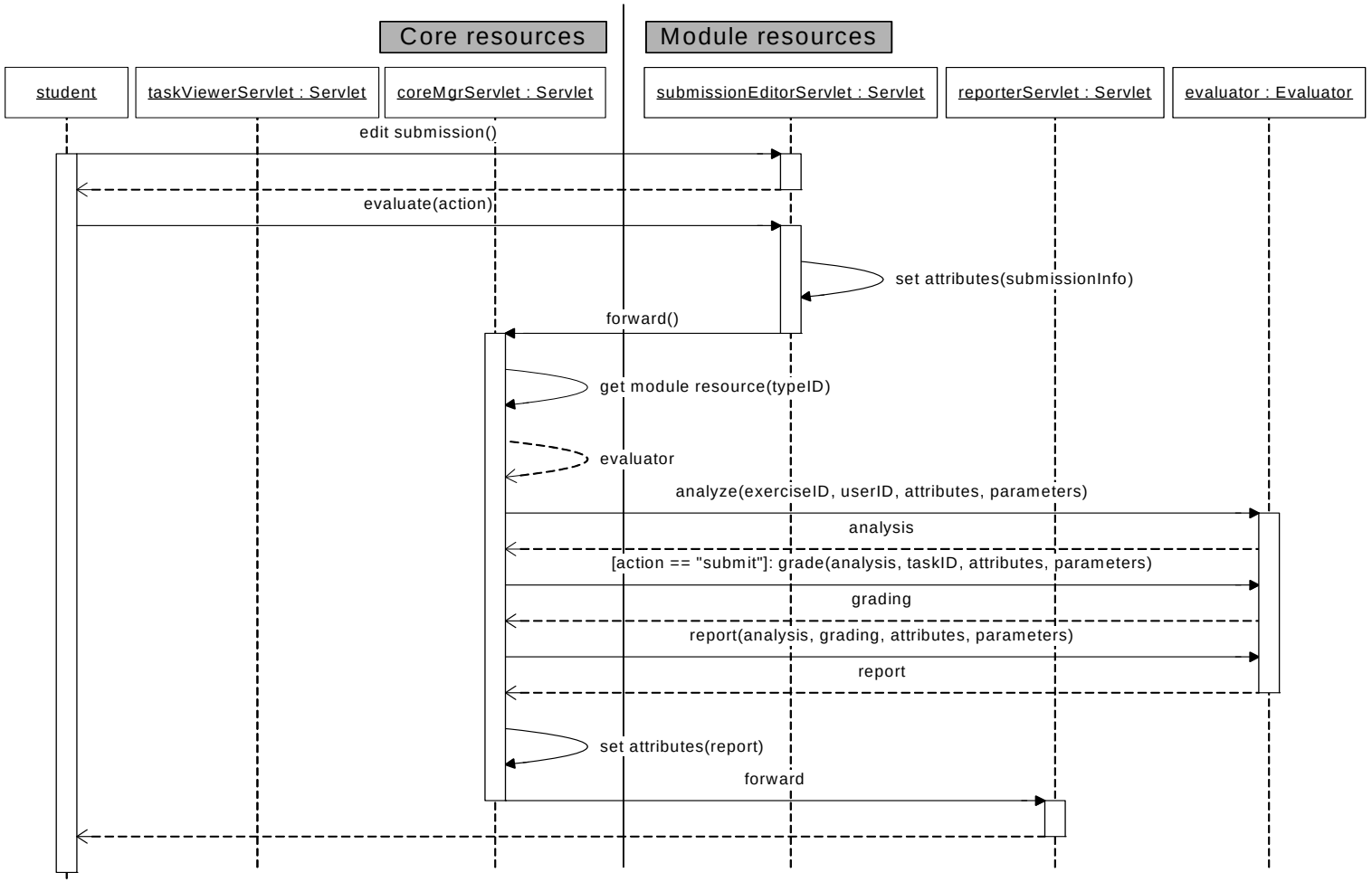

Abbildung 3.2: Ablauf der Beispielauswertung (Teil 2)

3.2. Modul-Schnittstelle Evaluator

```
public interface etutor.core.evaluation.Evaluator {

        public Analysis analyze(
                int exerciseID,
                int userID,
                Map passedAttributes,
                Map passedParameters) throws Exception;

        public Grading grade(
                Analysis analysis,
                int taskID,
                Map passedAttributes,
                Map passedParameters) throws Exception;

        public Report report(
                Analysis analysis,
                Grading grading,
                Map passedAttributes,
                Map passedParameters) throws Exception;
}
```

3.2.1. Methode analyze

Aufrufer

Methode `evaluate` der Klasse `CoreManagerServlet`.

Aufrufzeitpunkt

Die Methode `analyze` wird aufgerufen, wenn ein HTTP-Request von der modulspezifischen Maske für die Beispielausarbeitung an das `CoreManagerServlet` geschickt oder weitergeleitet wird. Die Zuordnung des HTTP-Request zum Modul wird vom eTutor Core über Session-Attribute gewährleistet, weshalb die Übermittlung von Parametern an das `CoreManagerServlet` für diesen Zweck nicht erforderlich ist.

Input-Parameter

Name	**Typ**	**Beschreibung**
exerciseID	int	ID des zu evaluierenden Beispiels.
userID	int	ID des Studenten dessen Abgabe evaluiert werden soll.
passedAttributes	Map	Session-Attribute und Request-Attribute
passedParameters	Map	Request-Parameter

Tabelle 3.1: Input-Parameter für die Analyse

Hinweis: In der Map `passedAttributes` werden die Namen der Attribute der Session und des Requests als Schlüssel verwendet. Der zu einem Schlüssel (Attribut Namen) zugehörige Wert ist der Wert des jeweiligen Attributs. Sämtliche vom eTutor Core oder vom Modul als Request- und als Session-Attribute gespeicherte Informationen sind daher für die Auswertungskomponente über diese Map zugänglich.

Beispiel: Object demoAttribute = passedAttributes.get("demoAttribute");

Hinweis: In der Map `passedParameters` werden die Namen der Parameter des Requests als Schlüssel verwendet. Die zu einem Schlüssel zugehörigen Werte sind die Werte des jeweiligen Request-Parameters, wobei ein Parameter beliebig viele Werte enthalten kann. Jeder Wert wird durch Array von String-Werten repräsentiert.

Beispiel: String demoParam = ((String[])passedParameters.get("demo"))[0];

Output-Parameter

Ein Objekt der modulspezifischen Klasse die das Interface `Analysis` implementiert. Dieses Objekt wird im folgenden als *Analyse-Objekt* bezeichnet. Es muss immer ein gültiges Analyse-Objekt zurückgegeben werden, d.h. bei einem Rückgabewert, der `null` ist, wird ein Fehler ausgelöst.

Das Analyse-Objekt muss die Studenten-Abgabe beinhalten. Die Studenten-Abgabe kann von der Modulkomponente durch die Methoden `setSubmission` und `getSubmission` gesetzt bzw. abgefragt werden. Die im Analyse-Objekt gesetzte Studenten-Abgabe muss ein serialisierbares JAVA Objekt sein und darf nicht `null` sein, um zu gewährleisten, dass die Studenten-Abgabe in der eTutor Core Datenbank gespeichert werden kann.

Im Analyse-Objekt muss vermerkt werden, ob die Studenten-Abgabe der richtigen Lösung gänzlich entspricht oder nicht. Dieser Vermerk kann durch die Methoden `setSubmissionSuitsSolution` und `submissionSuitsSolution` gesetzt bzw. abgefragt werden.

Hinweis: In der Klasse `DefaultAnalysis` (Default Implementierung des Interface `Analysis`) wird die Studenten-Abgabe im Feld `submission` gespeichert und im Konstruktor mit `null` initialisiert. Weiters wird das Feld `submissionSuitsSolution`, das angibt, ob die Studenten-Abgabe gänzlich der richtigen Lösung entspricht mit `false` im Konstruktor

initialisiert. Bei Verwendung dieser Klasse als Basisklasse für modulspezifische Analyse-Klassen darf nicht vergessen werden, die vorhin beschriebenen Felder mittels entsprechender Methoden zu setzen.

Exceptions

Grundsätzlich werden zwei Arten von Exceptions unterschieden. Einerseits können beim Analysieren Exceptions auftreten die durch fehlerhafte Studenten-Abgaben entstehen. Diese Exceptions müssen im Analyse-Objekt vermerkt werden, um dem Studenten entsprechendes Feedback geben zu können. Solche Exceptions müssen noch vom Modul in der Methode `analyze` abgefangen und entsprechend behandelt werden (try-catch Block).

Andererseits können beim Analysieren Exceptions aufgrund einer Fehlfunktion des Moduls auftreten. Solche Exceptions können von der Methode `analyze` einfach an die aufrufende Methode `evaluate` der Klasse `CoreManagerServlet` weitergeleitet werden. Zu diesem Zweck enthält die Deklaration der Methode `analyze` die "throws Exception" Klausel. Solche Exceptions werden vom `CoreManagerServlet` auf einer eigenen Seite dargestellt um den Studenten über den Zustand des Evaluierungs-Prozesses zu informieren. Weiters wird der Evaluierungs-Prozess gestoppt, sofern eine solche Exception auftritt.

Beispiel: Exceptions können bei der Ausführung eines JDBC-Programmes auftreten, wenn die Studenten-Abgabe fehlerhaft ist und die Kompilierung des JDBC-Programmes daher nicht möglich ist. Exceptions, die der JAVA Compiler dabei wirft, sind aufgrund einer fehlerhaften Studenten-Abgabe entstanden und müssen dem Studenten im Feedback mitgeteilt werden.

Beispiel: Ein Beispiel für Exceptions aufgrund einer Fehlfunktion des Moduls ist das Abbrechen der RMI Verbindung zwischen Modul und dem Stub am eTutor Core während der Analyse. Die resultierende RMI Exception ist aufgrund eines "Modul Fehlers" zustande gekommen und muss somit nicht im Modul abgefangen und im Analyse-Objekt vermerkt werden, sondern kann an die aufrufende Methode `evaluate` weitergeleitet werden.

3.2.2. Methode grade

Aufrufer

Methode `evaluate` der Klasse `CoreManagerServlet`.

Voraussetzungen für Aufruf

Die Methode `grade` wird vom `CoreManagerServlet` nur aufgerufen wenn das Modul signalisiert, dass die Lösung vom Studenten als zu bewertende Abgabe übermittelt wurde. In diesem Fall muss im HTTP-Request, den das Modul an das `CoreManagerServlet` schickt oder weiterleitet, ein Request-Parameter oder ein Request-Attribut `action` mit dem Wert `submit` enthalten sein. Wie bereits erwähnt sollte eine solche Abgabe nur möglich sein, wenn dieser Wert in der Menge von möglichen Aktionen enthalten ist, die vom eTutor Core an die modulspezifische Eingabemaske übermittelt wurde. Andernfalls besteht keine Veranlassung, die Lösung zu bewerten.

Aufrufzeitpunkt

Die Methode `grade` wird nur aufgerufen, nachdem ein Analyse-Objekt von der zuvor aufgerufenen Methode `analyze` zurückgegeben wurde und die oben erwähnten Voraussetzungen erfüllt sind.

Input-Parameter

Name	**Typ**	**Beschreibung**
analysis	Analysis	Modulspezifische Analyse
taskID	int	ID der zu bewertenden Aufgabe.
passedAttributes	Map	Siehe Abschnitt 3.2.1
passedParameters	Map	Siehe Abschnitt 3.2.1

Tabelle 3.2: Input-Parameter für die Bewertung

Output-Parameter

Ein Objekt der modulspezifischen Klasse die das Interface `Grading` implementiert. Dieses Objekt wird im folgenden als *Grading-Objekt* bezeichnet. Es muss immer ein gültiges Grading-Objekt zurückgegeben werden, d.h. bei einem Rückgabewert, der `null` ist, wird ein Fehler ausgelöst.

Das Grading-Objekt muss angeben, wie viele Punkte bei der jeweiligen Analyse maximal erreicht werden hätten können (maxPoints) und wie viele Punkte die Studenten-Abgabe wert ist (points). Entsprechende Methoden zum Setzen und Abfragen dieser Werte sind im Interface `Grading` spezifiziert. Aufgrund dieser Angaben wird vom `CoreManagerServlet` eine Gewichtung durchgeführt. Werden beispielsweise für die jeweilige Aufgabe 12 Punkte vergeben (taskPoints) und ist

`maxPoints` auf 10 bzw. `points` auf 5 gesetzt, so werden vom `CoreManagerServlet` 6 Punkte für dieses Beispiel vergeben (`taskPoints / maxPoints* points`). Deshalb ist darauf zu achten, dass `maxPoints` nie den Wert 0 enthält, da ansonsten ein Laufzeit Fehler resultiert (Division durch 0).

Hinweis: In der Klasse `DefaultGrading` (Default Implementierung des Interface `Grading`) werden die Felder `maxPoints` und `points` mit den Werten 1 bzw. 0 im Konstruktor initialisiert.

Exceptions

Beim Grading werden nur Exceptions, die aufgrund eines Fehlers im Modul entstanden sind, erwartet. Somit können im allgemeinen alle Exceptions an die aufrufende Methode `evaluate` weitergeleitet werden (siehe Beschreibung der Methode `analyze`).

3.2.3. Methode report

Aufrufer

Methode `evaluate` der Klasse `CoreManagerServlet`.

Aufrufzeitpunkt

Im Falle, dass die vom Studenten übermittelte Lösung zu bewerten ist, wird die Methode `report` aufgerufen, nachdem ein Grading-Objekt von der zuvor aufgerufenen Methode `grading` zurückgegeben wurde.

Im Falle, dass die vom Studenten übermittelte Lösung nicht zu bewerten ist, wird die Methode `report` bereits aufgerufen, nachdem ein Analyse-Objekt von der zuvor aufgerufenen Methode `analyze` zurückgegeben wurde.

Input-Parameter

Name	**Typ**	**Beschreibung**
analysis	Analysis	Modulspezifische Analyse
grading	Grading	Modulspezifische Bewertung (`null` falls Evaluierung im Practise Mode)
passedAttributes	Map	Siehe Abschnitt 3.2.1
passedParameters	Map	Siehe Abschnitt 3.2.1

Tabelle 3.3: Input-Parameter für das Feedback

Output-Parameter

Ein Objekt der modulspezifischen Klasse die das Interface `Report` implementiert. Dieses Objekt wird im folgenden als *Report-Objekt* bezeichnet. Es muss immer ein gültiges Report-Objekt zurückgegeben werden, d.h. bei einem Rückgabewert, der `null` ist, wird ein Fehler ausgelöst. Das Report-Objekt muss keine spezifischen Methoden implementieren, da das Interface `Report` derzeit lediglich ein Marker-Interface ist.

Exceptions

Beim Erstellen des Feedbacks werden nur Exceptions, die aufgrund eines Fehlers im Modul entstanden sind, erwartet. Somit können alle Exceptions an die aufrufende Methode `evaluate` weitergeleitet werden (siehe Beschreibung der Methode `analyze`).

3.3. Request- und Session-Attribute

Die in der nachfolgenden Tabelle beschriebenen Attribute werden im Rahmen einer Beispielauswertung vom eTutor-System in der Benutzer-Session der Web-Applikation verwaltet. Diese Informationen sind u.U. für die Umsetzung der Eingabemasken von Bedeutung. Bei der Anzeige einer modulspezifischen Eingabemaske kann davon ausgegangen werden, dass diese Informationen den aktuellen Zustand der Benutzer-Session widerspiegeln. So sorgt der eTutor Core dafür, dass beispielsweise das Attribut *exerciseID* dem Beispiel entspricht, das der Student unmittelbar vor der Aktivierung des Moduls ausgewählt hat.

Name	**Typ**	**Beschreibung**
userID	String	ID des angemeldeten Benutzers
exerciseID	String	ID des zu evaluierenden Beispiels
taskID	String	ID der zu evaluierenden Aufgabe, die eine Zuordnung eines Beispiels zu einem Studenten repräsentiert
modeID	String	ID des aktuellen Ausführungsmodus'
typeID	String	ID des momentanen Beispieltyps
actions	Set	Ein Set von String-Werten, die die Menge der Aktionen vorgibt, die einem Studenten für die Beispielausarbeitung zur Verfügung stehen. Mögliche Werte sind *run*, *check*, *diagnose*, *submit*
action	String	Muss vom Modul definiert werden, um dem eTutor

		Core bei der Beispielauswertung die ausgewählte Aktion anzuzeigen

Tabelle 3.4: Attribute für die Beispielauswertung

Name	**Verwendungszweck**
userID	Z.B. Logging-Information, Anlegen von unterschiedlichen temporären Verzeichnissen und Dateien je nach Benutzer, Überprüfung der Gültigkeit der Session, etc.
exerciseID	Identifikation des vom Studenten ausgewählten Beispiels im Datenbestand
taskID	Z.B. Erzeugung eines eindeutigen Namens, unter dem eine Studenten-Lösung in der Session gespeichert werden kann, um sie bei Auswahl derselben Aufgabe zu einem späteren Zeitpunkt wieder anzeigen zu können
modeID	Für das Modul nicht von Bedeutung
typeID	Für das Modul nicht von Bedeutung
actions	Wird vom CoreManagerServlet als Session-Attribut gespeichert, bevor die modulspezifische View für die Beispielausarbeitung durch Studenten aufgerufen wird. Diese sollte die Informationen verwenden, um die entsprechenden Interaktionsmöglichkeiten zur Verfügung zu stellen und die Analyse, Bewertung und das Feedback anzupassen.
action	Bevor der Request von der modulspezifischen View an das `CoreManagerServlet` weitergeleitet wird, muss dieses Attribut im Request gespeichert werden. Der Wert sollte dabei einem der vom eTutor Core vorgegebenen Aktionen entsprechen und Auskunft über die vom Benutzer ausgewählte Aktion geben.

Tabelle 3.5: Attribute für die Beispielauswertung (Verwendungszweck)

4. Beispieladministration

Die Beispieladministration umfasst unter anderem Aufgaben wie die Erstellung, die Änderung oder das Löschen von Übungsbeispielen. In Abschnitt 4.1 wird die Erstellung eines neuen Übungsbeispiels näher erläutert. Dies ist zwar nur ein bestimmter Anwendungsfall, der durch die Administrationskomponente des Moduls unterstützt werden soll, anhand des gezeigten Beispiels soll jedoch das Zusammenspiel zwischen eTutor Core und der Administrationskomponente eines Moduls demonstriert werden. Das Interface, das für die Beispieladministration vom eTutor Core vorgegeben wird und vom Modul implementiert werden muss, wird im Detail in Abschnitt 4.2 erläutert.

4.1. Ablauf der Beispieladministration

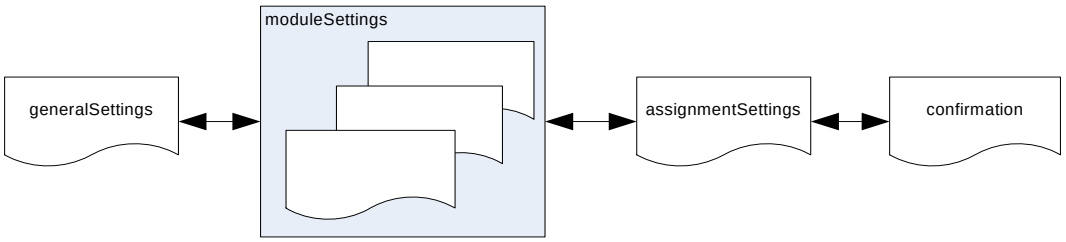

Abbildung 4.1: Schritte bei der Beispieladministration

Die Erstellung und die Änderung eines Beispiels setzen sich jeweils aus einer Sequenz einzelner Schritte zusammen (siehe Abbildung 4.1). Jeder Schritt entspricht dabei einer View, die dem Benutzer angezeigt wird. Alle Informationen, die für die Spezifikation von Übungsbeispielen benötigt werden, werden über diese Folge von Schritten hinweg eingegeben und im letzten Schritt angewendet.

Im ersten Schritt werden vom Benutzer allgemeine beispielbezogene Informationen eingegeben, wie etwa die Auswahl des Beispieltyps. Der Beispieltyp wird im nächsten Schritt benötigt, um das entsprechende Modul zu identifizieren. Für die Eingabe von modulspezifischen Informationen muss das

Modul zumindest eine View bereitstellen, die Eingabe kann sich jedoch auch über mehrere Views hinweg erstrecken. Dieser Schritt stellt daher aus Sicht des eTutor Core eine Black Box dar. Im darauf folgenden Schritt kann der Benutzer den Angabetext zur Übungsaufgabe spezifizieren. Bevor die Änderungen wirksam werden, wird der Benutzer im letzten Schritt noch zur Bestätigung der gewünschten Aktion aufgefordert.

Ein zentrales Servlet, das `ExerciseManagerServlet`, sorgt für die Navigation zwischen den einzelnen Views. Dazu ist eine Menge bestimmter Navigationselemente vorgesehen. In erster Linie kann zwischen den einzelnen Views vorwärts (next) und rückwärts (back) navigiert werden. Darüber hinaus sollte es in jeder View die Möglichkeit geben, den gesamten Vorgang abzubrechen (cancel). Die tatsächliche Anwendung der Änderungen ist dem letzten Schritt vorbehalten (finish). Tabelle 4.1 enthält eine Zusammenfassung der vorgesehenen Navigationselemente für jeden Schritt bei der Beispieladministration. Das `ExerciseManagerServlet` berechnet aus jeder Kombination aus dem Status, d.h. aus der zuletzt angezeigten View, und dem betätigten Navigationselement die nächste View. Umgekehrt kann das Modul, falls es mehrere Views anbietet, anhand dieser Informationen entscheiden, welche View angezeigt wird. Wird beispielsweise im Schritt *assignmentSettings* vom Benutzer das Navigationselement *back* ausgewählt, so leitet das `ExerciseManagerServlet` daraus ab, dass die modulspezifische View angezeigt werden soll. Das Modul wiederum weiß aufgrund des gewählten Navigationselementes *back*, dass die letzte Seite in der Reihe der modulspezifischen Views angezeigt werden soll. Die Navigation zwischen den einzelnen modulspezifischen Views wird durch das Modul selbst bewerkstelligt.

	generalSettings	**moduleSettings**	**assignmentSettings**	**confirmation**
back		√	√	√
next	√	√	√	
cancel	√	√	√	√
finish				√

Tabelle 4.1: Navigationselemente bei der Beispieladministration

Die Beispieladministration wird anhand von Sequenzdiagrammen in Abbildung 4.2 und Abbildung 4.3 detaillierter dargestellt. Die beiden Diagramme zeigen nur einen vereinfachten Ausschnitt aus einer möglichen Sequenz zur Erstellung eines neuen Beispiels. Die Darstellung der Schritte, die bei der Änderung eines existierenden Beispiels durchgeführt werden, würde sich zwar im Detail von den gezeigten Diagrammen unterscheiden, die Abläufe bei Erstellung und Änderung von Beispielen sind jedoch grundsätzlich vergleichbar. Das Löschen eines Beispiels erweist sich als vergleichsweise weniger komplexer Vorgang, für den das Modul auch keine zusätzlichen Views bereitstellen muss. Aus diesem Grund wird auch diese Funktion der Administrationskomponente hier nicht näher erläutert.

Der Ablauf der Beispieladministration folgt einem Muster, das dem in Abschnitt 3.1 beschriebenen Ablauf bei der Beispielauswertung entspricht. Einerseits wird eine Unterscheidung zwischen Views und Komponenten getroffen, und andererseits zwischen modulspezifischen Ressourcen und Ressourcen des eTutor Core. In den Abbildungen wird davon ausgegangen, dass Views als Servlets implementiert sind. Ebenso wie bei der Beispielauswertung gibt es bei der Beispieladministration mit dem `ExerciseManagerServlet` ein zentrales Servlet, das die wichtigsten Aufgaben bei der Steuerung des Kontrollflusses und bei der Kommunikation zwischen den einzelnen Views übernimmt (exerciseMgrServlet). Außerdem übernimmt es Kommunikationsaufgaben zwischen den modulspezifischen Views (moduleExerciseServlet) und der Administrationskomponente des Moduls (moduleExerciseMgr). Dies betrifft vor allem die Abfrage eines modulspezifischen Objektes über die Administrationskomponente, das als Attribut in der Session gespeichert wird. Die modulspezifischen Views greifen auf dieses Attribut zu, um das Objekt anzuzeigen und den Benutzerangaben entsprechend zu ändern. Umgekehrt wird dieses Objekt wiederum vom `ExerciseManagerServlet` verwendet, um es über die Administrationskomponente des Moduls zum Erzeugen oder Ändern des Beispiels zu verwenden.

Der Vorgang wird initialisiert, wenn in der Benutzerschnittstelle das Kontrollelement zum Erstellen eines neuen Beispiels ausgewählt wird. Das `ExerciseManagerServlet` (exerciseMgrServlet) initialisiert ein Objekt, das alle für den eTutor Core relevanten Informationen eines neuen Beispieles aufnehmen kann (exercise). Dieses Objekt wird für die Views des eTutor Core als Attribut in der Session abgelegt, ebenso wie die ausgewählte Aktion (*create*). Von dieser Aktion hängen im weiteren Verlauf verschiedene Entscheidungen des eTutor Core

ab, für das Modul ist diese Information jedoch irrelevant, da für das Erstellen und für das Ändern von Beispielen die gleichen Views verwendet werden.

Der Benutzer wird nun auf eine Seite zur Angabe von allgemeinen Informationen geleitet (exerciseServlet). Die vom Benutzer eingegebenen Informationen werden im Objekt (exercise) zwischengespeichert und der Kontrollfluss wieder zurück an das `ExerciseManagerServlet` gegeben. Um den nächsten Schritt ermitteln zu können, müssen zwei Attribute definiert werden. Dies ist einerseits der aktuelle Status (*coreExerciseMgrStatus*), der in diesem Fall den Wert *‚generalSettings'* erhält, und andererseits das vom Benutzer ausgewählte Navigationselement (*coreExerciseMgrCmd*) mit dem Wert *‚next'*. Daraus wird abgeleitet, dass dem Benutzer im nächsten Schritt die modulspezifische View angezeigt wird. Sowohl die Administrationskomponente (moduleExerciseMgr) als auch die View (moduleExerciseServlet) werden anhand des Beispieltyps (typeID) ermittelt, den der Benutzer bei der Angabe allgemeiner Informationen zum Beispiel ausgewählt hat. Mithilfe der Methode `fetchExerciseInfo`, das im Interface `ModuleExerciseManager` definiert ist, erhält das `ExerciseManagerServlet` ein modulspezifisches Objekt, das alle für das Modul relevanten Informationen in Bezug auf das neu zu erstellende Beispiel enthält. Im Diagramm ist bei diesem Aufruf die Bedingung, dass die ausgewählte Aktion *‚create'* sein muss, zu sehen. Dadurch soll zum Ausdruck gebracht, dass diese Methode nur dann aufgerufen wird, wenn ein Beispiel neu erstellt werden soll. Im Gegensatz dazu würde beim Ändern eines existierenden Beispiels hier die Methode `fetchExercise` aufgerufen.

Die Administrationskomponente initialisiert ein modulspezifisches Objekt (moduleExercise) und liefert es an das `ExerciseManagerServlet` zurück. Dort wird das Objekt als Attribut in der Session gespeichert, wo es im nächsten Schritt für die modulspezifische View verfügbar ist.

Nachdem der Benutzer die modulspezifischen Informationen eingegeben hat (was sich wie weiter oben beschrieben über mehrere Views erstrecken kann), und das Navigationselement *next* ausgewählt hat, führt die modulspezifische View (moduleExerciseServlet) dieselben Schritte wie davor das Servlet des eTutor Core (exerciseServlet) durch. Einerseits wird das Objekt (moduleExercise) vorbereitet, andererseits werden für das `ExerciseManagerServlet` im Request Attribute vermerkt, die Aufschluss über den Status (*moduleSettings*) und das ausgewählte Navigationselement (*next*) geben. Das `ExerciseManagerServlet` leitet daraus nun wiederum die nächste View ab. In der Abbildung wird auf die Darstellung der

weiteren Schritte (wie z.B. die Spezifikation des Angabetextes) verzichtet und angenommen, dass ab diesem Zeitpunkt bereits versucht wird, ein neues Beispiel im Datenbestand anzulegen.

Anhand des Beispieltyps (typeID) wird wiederum die Administrationskomponente (moduleExerciseMgr) des Moduls ermittelt. Nachdem das Beispiel neu erstellt werden soll, wird über diese Modul-Schnittstelle die Methode `createExercise` aufgerufen. Bei einem zu ändernden Beispiel würde die Methode `modifyExercise` aufgerufen werden. Nur wenn diese Aktion erfolgreich ist, wird im nächsten Schritt versucht, das Beispiel im Datenbestand des eTutor Core anzulegen. Abschließend wird dem Benutzer das Resultat dieser Transaktion angezeigt.

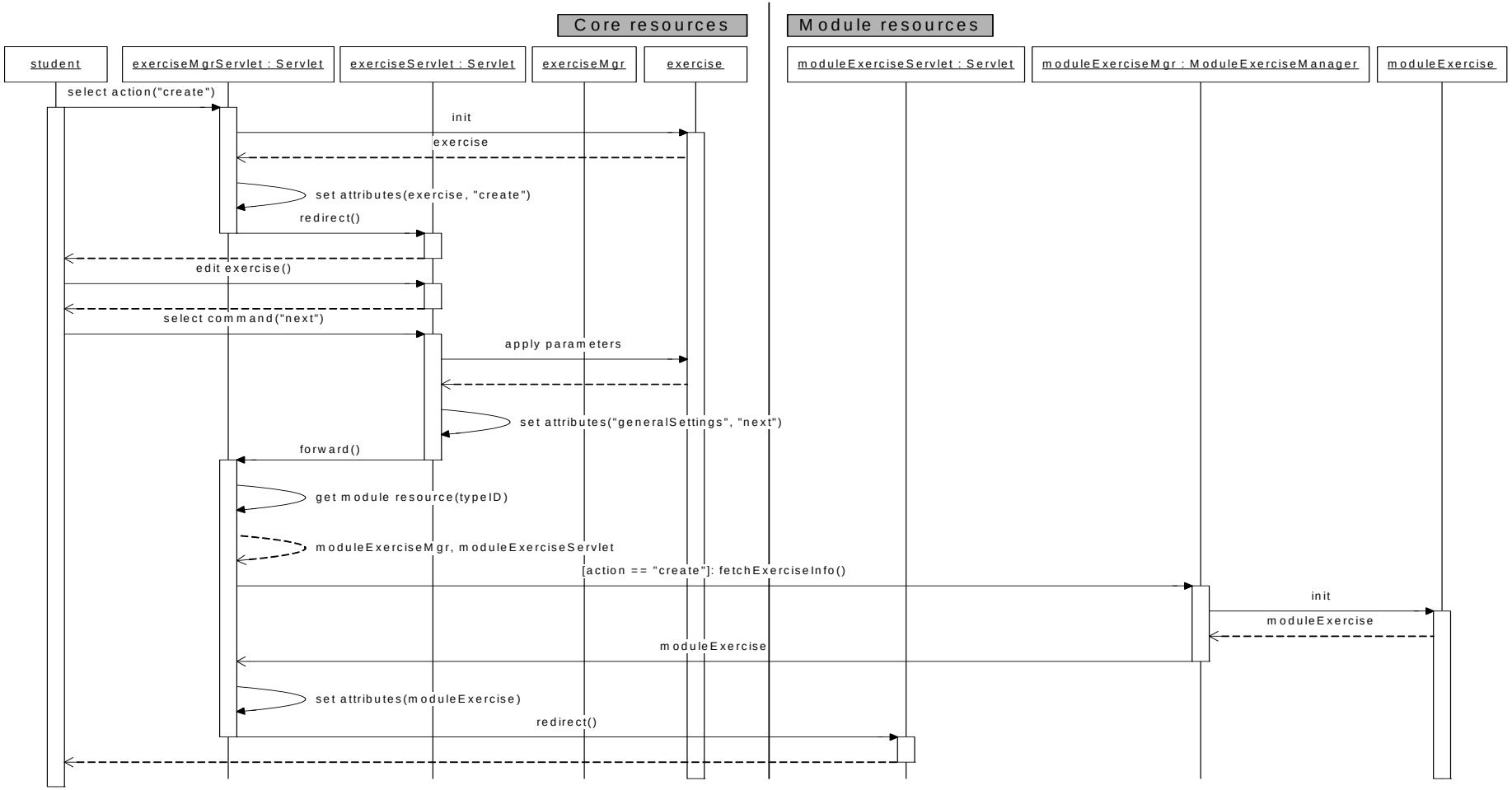

Abbildung 4.2: Ablauf der Beispieladministration (Teil 1)

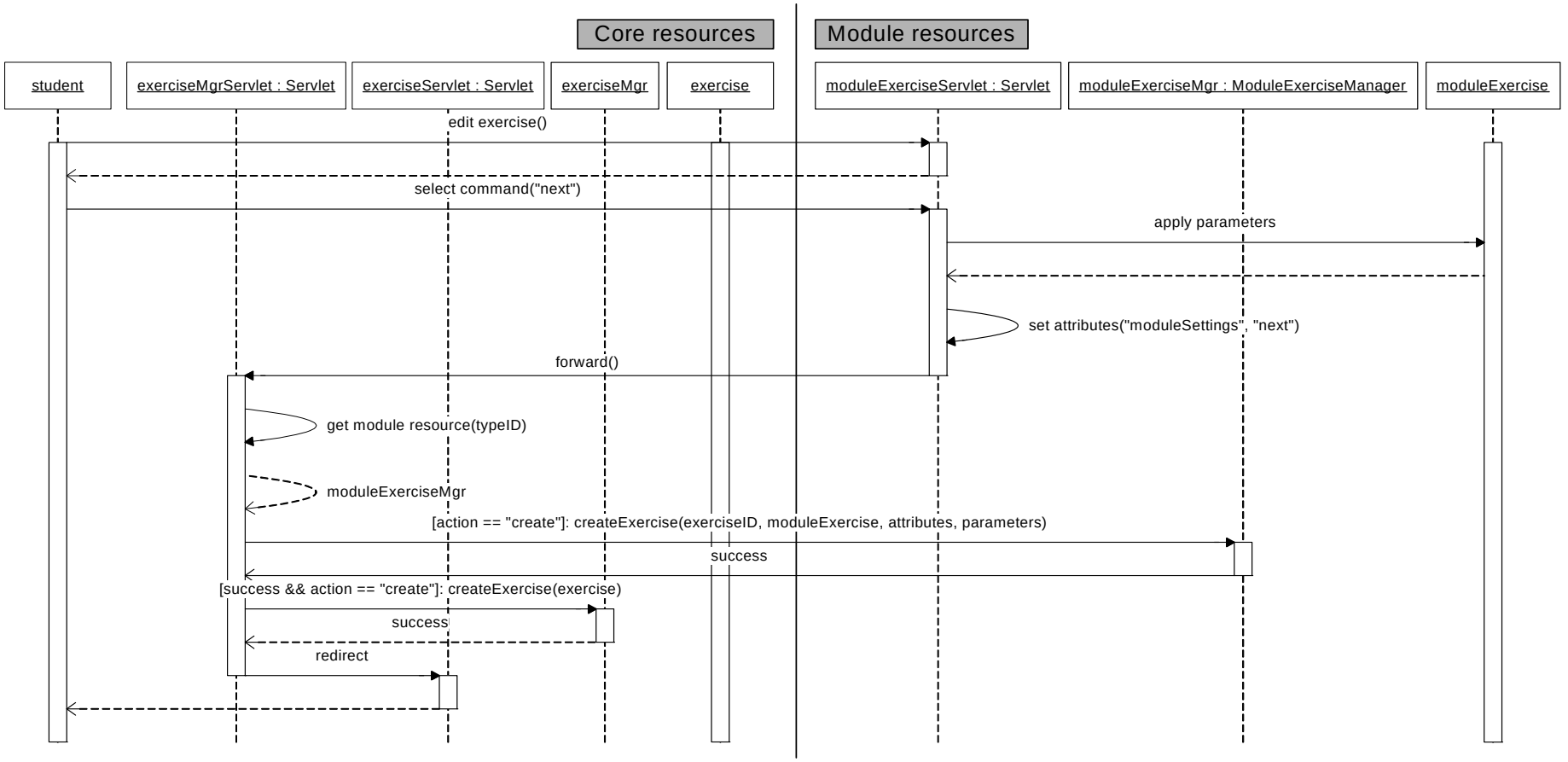

Abbildung 4.3: Ablauf der Beispieladministration (Teil 2)

Diesen Abschnitt abschließend sei noch eine Erläuterung zur Kommunikation zwischen den Views hinzugefügt. Die Java Servlet API sieht unterschiedliche Möglichkeiten vor, aus einer Web-Ressource eine weitere Web-Ressource aufzurufen. Einerseits kann der Request als Objekt an eine weitere Ressource im Web-Container geleitet werden (Methode `forward`) und andererseits kann der Client-Browser dazu veranlasst werden, eine Ressource neu zu laden (Methode `redirect`). Während in der ersteren Methode alle Parameter und die im Request gespeicherten Attribute für die aufgerufene Ressource erhalten bleiben, wird mit der zweiten Methode garantiert, dass an die aufgerufene Ressource keine Parameter oder Attribute übermittelt werden, die nicht erwartet oder im schlimmsten Fall als Fehler interpretiert werden. Nachdem das `ExerciseManagerServlet` zwischen unterschiedlichen Views vermittelt, wird zur Vermeidung unerwünschter Seiteneffekte die Methode `redirect` verwendet, um Views aufzurufen, die dem Benutzer für die Spezifikation von Beispielinformationen angezeigt werden sollen. Das `ExerciseManagerServlet` selbst hingegen kann sowohl weitergeleitete Requests als auch direkte HTTP-Requests empfangen (siehe Abbildung 4.4).

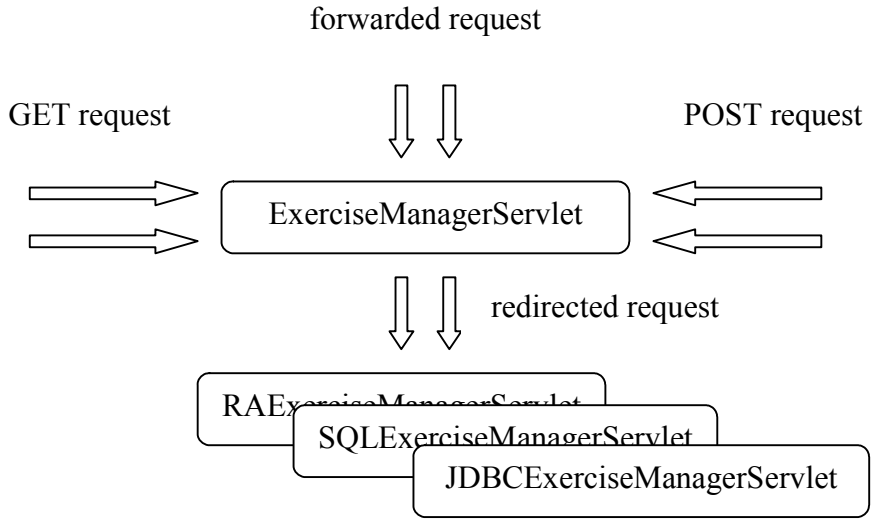

Abbildung 4.4: Vermittlungsrolle des ExerciseManagerServlet

4.2. Modul-Schnittstelle ModuleExerciseManager

```
public interface etutor.core.manager.ModuleExerciseManager {

    public boolean createExercise(
            int exerciseId,
            Serializable exercise,
            Map attributes,
            Map parameters) throws Exception;
```

```
        public boolean modifyExercise(
                int exerciseId,
                Serializable exercise,
                Map attributes,
                Map parameters) throws Exception;

        public boolean deleteExercise(int exerciseID) throws Exception;

        public Serializable fetchExercise(int exerciseID) throws Exception;

        public Serializable fetchExerciseInfo() throws Exception;

        public String generateHtml(Serializable exercise, Locale locale);

}
```

4.2.1. Methode createExercise

Aufrufer

Methode `doCreateExercise` der Klasse `ExerciseManagerServlet`.

Aufrufzeitpunkt

Die Methode `createExercise` wird aufgerufen, wenn der Benutzer in einer vom eTutor Core angezeigten Seite seine Angaben bestätigt und damit den Versuch, das Beispiel anzulegen, initialisiert. Wenn das Anlegen der modulspezifischen Informationen zum Übungsbeispiel erfolgreich war, so wird im nächsten Schritt versucht, die vom eTutor Core verwalteten Informationen zum Übungsbeispiel zu speichern. Erst wenn dieser Schritt ebenso erfolgreich war, so gilt das Übungsbeispiel als erzeugt. Aufgrund einer fehlenden Transaktionskontrolle zwischen eTutor Core und Modul ist es im schlimmsten Fall möglich, dass die Änderungen am modulspezifischen Datenbestand durchgeführt werden, während Änderungen am Datenbestand des eTutor Core fehlschlagen, bzw. rückgängig gemacht werden müssen.

Input-Parameter

Name	**Typ**	**Beschreibung**
exerciseID	int	ID des neuen Beispiels. Wenn alle vom Modul verwalteten modulspezifischen Informationen zu Beispielen einer Teilmenge der vom eTutor Core verwalteten Beispiele entspricht, so kann das Modul davon ausgehen, dass ein Beispiel mit der übergebenen ID noch nicht existiert.

exercise	Serializable	Modulspezifisches Objekt, das alle vom Modul zum Erstellen des Beispiels benötigten Informationen enthält. Das Objekt entspricht dem Rückgabewert der in Abschnitt 4.2.4 erläuterten Methode.
passedAttributes	Map	Siehe Abschnitt 3.2.1
passedParameters	Map	Siehe Abschnitt 3.2.1

Tabelle 4.2: Input-Parameter für das Erstellen eines Beispiels

Output-Parameter

Das Modul muss über den Rückgabewert anzeigen, ob das Beispiel in der modulspezifischen Datenbasis erfolgreich eingefügt wurde (*true*) oder nicht (*false*). Ein Fehler sollte signalisiert werden, indem eine Exception ausgelöst, bzw. an die aufrufende Klasse weitergeleitet wird. Im Gegensatz dazu kann der Rückgabewert *false* sein, wenn die Funktionalität vom Modul nicht, bzw. noch nicht unterstützt wird. Ist der Rückgabewert *true*, so sollte das erzeugte Beispiel in einem nachfolgenden Schritt über die Methode `fetchExercise` wieder abrufbar sein.

Exceptions

Jegliche beim Anlegen der modulspezifischen Informationen in der Datenbasis des Moduls auftretenden Probleme sollten dem eTutor Core durch eine Exception angezeigt werden. Der Benutzer erhält in diesem Fall einen vom eTutor Core generierten Hinweis auf den fehlgeschlagenen Versuch, das Beispiel anzulegen. Wenn das Modul selbst eine bestimmte Fehlermeldung generiert, die an den Benutzer adressiert werden soll, so kann eine Exception vom Typ `etutor.core.ModuleException` ausgelöst werden. Das Modul kann über die Methode `setUserMsg` in der Exception eine Nachricht speichern, die vom eTutor Core für den Benutzer angezeigt wird. Auf diese Weise kann der Benutzer auch über fehlerhafte Angaben informiert werden, die erst bei dem endgültigen Versuch, das Beispiel anzulegen, überprüft werden können, und nicht etwa bereits bei der Eingabe.

4.2.2. Methode modifyExercise

Aufrufer

Methode `doUpdateExercise` der Klasse `ExerciseManagerServlet`.

Aufrufzeitpunkt

Die Methode `createExercise` wird aufgerufen, wenn der Benutzer in einer vom eTutor Core angezeigten Seite seine Änderungen bestätigt und damit den Versuch, das Beispiel zu modifizieren, initialisiert. Wenn das Ändern der modulspezifischen Informationen zum Übungsbeispiel erfolgreich war, so wird im nächsten Schritt versucht, die vom eTutor Core verwalteten Informationen zum Übungsbeispiel zu ändern. Erst wenn dieser Schritt ebenso erfolgreich war, so gilt das Übungsbeispiel als geändert. Aufgrund einer fehlenden Transaktionskontrolle zwischen eTutor Core und Modul ist es im schlimmsten Fall möglich, dass die Änderungen am modulspezifischen Datenbestand durchgeführt werden, während Änderungen am Datenbestand des eTutor Core fehlschlagen, bzw. rückgängig gemacht werden müssen.

Input-Parameter

Name	**Typ**	**Beschreibung**
exerciseID	int	ID des zu ändernden Beispiels.
exercise	Serializable	Modulspezifisches Objekt, das alle vom Modul zum Ändern des Beispiels benötigten Informationen enthält. Das Objekt entspricht dem Rückgabewert der in Abschnitt 4.2.5 erläuterten Methode.
passedAttributes	Map	Siehe Abschnitt 3.2.1
passedParameters	Map	Siehe Abschnitt 3.2.1

Tabelle 4.3: Input-Parameter für das Erstellen eines Beispiels

Output-Parameter

Das Modul muss über den Rückgabewert anzeigen, ob das Beispiel in der modulspezifischen Datenbasis erfolgreich geändert wurde (*true*) oder nicht (*false*). Ein Fehler sollte signalisiert werden, indem eine Exception ausgelöst, bzw. an die aufrufende Klasse weitergeleitet wird. Im Gegensatz dazu kann der Rückgabewert *false* sein, wenn die Funktionalität vom Modul nicht, bzw. noch nicht unterstützt wird. Ist der Rückgabewert *true*, so sollte das geänderte Beispiel in einem nachfolgenden Schritt über die Methode `fetchExercise` wieder abrufbar sein.

Exceptions

Jegliche beim Ändern der modulspezifischen Informationen in der Datenbasis des Moduls auftretenden Probleme sollten über eine Exception angezeigt werden. Der Benutzer erhält in diesem Fall einen vom eTutor Core generierten Hinweis auf den fehlgeschlagenen Versuch, das Beispiel zu ändern. Wenn das Modul selbst eine bestimmte Fehlermeldung generiert, die an den Benutzer adressiert werden soll, so kann eine Exception vom Typ `etutor.core.ModuleException` ausgelöst werden. Das Modul kann über die Methode `setUserMsg` in der Exception eine Nachricht speichern, die vom eTutor Core für den Benutzer angezeigt wird. Auf diese Weise kann der Benutzer auch über fehlerhafte Angaben informiert werden, die erst bei dem endgültigen Versuch, das Beispiel zu ändern, überprüft werden können, und nicht etwa bereits bei der Eingabe.

4.2.3. Methode deleteExercise

Aufrufer

Methode `doDeleteExercise` der Klasse `ExerciseManagerServlet`.

Aufrufzeitpunkt

Die Methode `deleteExercise` wird aufgerufen, wenn der Benutzer in einer vom eTutor Core angezeigten Seite den Wunsch, ein bestimmtes Beispiel zu löschen, bestätigt. Im Gegensatz zum Anlegen und Ändern wird beim Löschen eines Beispiels im ersten Schritt versucht, die vom eTutor Core verwalteten Informationen zum Übungsbeispiel zu löschen, und erst danach durch Aufruf der Methode `deleteExercise` die modulspezifischen Informationen. Dadurch wird versucht, eine fehlende Transaktionskontrolle zwischen eTutor Core und Modul zu kompensieren, indem die Änderungen im Datenbestand des eTutor Core rückgängig gemacht werden, wenn das Löschen der modulspezifischen Informationen fehlschlägt.

Input-Parameter

Name	**Typ**	**Beschreibung**
exerciseID	int	ID des zu löschenden Beispiels.

Tabelle 4.4: Input-Parameter für das Löschen eines Beispiels

Output-Parameter

Das Modul muss über den Rückgabewert anzeigen, ob das Beispiel in der modulspezifischen Datenbasis erfolgreich gelöscht wurde (*true*) oder nicht (*false*).

Ein Fehler sollte signalisiert werden, indem eine Exception ausgelöst, bzw. an die aufrufende Klasse weitergeleitet wird. Im Gegensatz dazu kann der Rückgabewert *false* sein, wenn die Funktionalität vom Modul nicht, bzw. noch nicht unterstützt wird. Ist der Rückgabewert *true*, so sollte das gelöschte Beispiel in einem nachfolgenden Schritt über die Methode `fetchExercise` nicht mehr abrufbar sein.

Exceptions

Jegliche beim Löschen der modulspezifischen Informationen in der Datenbasis des Moduls auftretenden Probleme sollten über eine Exception angezeigt werden. Der Benutzer erhält in diesem Fall einen vom eTutor Core generierten Hinweis auf den fehlgeschlagenen Versuch, das Beispiel zu löschen.

4.2.4. Methode fetchExerciseInfo

Aufrufer

Methode `showModuleExerciseWizard` der Klasse `ExerciseManagerServlet`.

Aufrufzeitpunkt

Die Methode `fetchExerciseInfo` wird aufgerufen, wenn der Benutzer in einer Sequenz von Schritten, die das Erzeugen eines Beispiels umfasst, vom eTutor Core auf die modulspezifische View zur Spezifikation der modulspezifischen Informationen geleitet wird. Nachdem zwischen modulspezifischen Views und der Administrationskomponente keine direkte Kommunikation stattfindet, wird das `ExerciseManagerServlet` als Vermittler zwischengeschaltet, um über die Methode `fetchExerciseInfo` ein Objekt zu ermitteln, das für die modulspezifische View zur Anzeige und zur Speicherung der vom Benutzer eingegebenen Informationen verfügbar gemacht wird.

Input-Parameter

Die Methode `fetchExerciseInfo` benötigt keine Input-Parameter.

Output-Parameter

Das Modul liefert ein Objekt eines beliebigen modulspezifischen Typs, der aber zumindest das Interface `java.io.Serializable` implementieren und das Objekt dementsprechend serialisierbar sein muss. Dieses Objekt ist für die modulspezifischen Views zur Anzeige und zur Speicherung der vom Benutzer eingegebenen Informationen als Session-Attribut verfügbar (siehe Abschnitt 4.3).

Dieses Objekt wird, nachdem eventuelle Änderungen von modulspezifischen Views in diesem Objekt zwischengespeichert wurden, wieder an die entsprechenden Methoden zum Erzeugen (Abschnitt 4.2.1) von modulspezifischen Informationen übergeben.

In diesem Objekt können vom Modul jegliche Informationen vermerkt werden, die für die Eingabe von Informationen für das Erzeugen eines neuen Beispiels relevant sind. Beispielsweise kann der Benutzer beim Erzeugen eines Beispiels für das SQL-Modul aus einer Liste von Datenbankschemata das Referenzschema zum Ausführen von SQL-Queries auswählen.

Exceptions

Treten beim Abrufen des Beispiels Probleme auf, so sollte dies dem eTutor Core als Exception signalisiert werden.

4.2.5. Methode fetchExercise

Aufrufer

Methode `showModuleExerciseWizard` der Klasse `ExerciseManagerServlet`.

Aufrufzeitpunkt

Die Methode `fetchExercise` wird aufgerufen, wenn der Benutzer in einer Sequenz von Schritten, die das Ändern eines Beispiels umfasst, vom eTutor Core auf die modulspezifische View zur Spezifikation der modulspezifischen Informationen geleitet wird. Nachdem zwischen modulspezifischen Views und der Administrationskomponente keine direkte Kommunikation stattfindet, wird das `ExerciseManagerServlet` als Vermittler zwischengeschaltet, um über die Methode `fetchExercise` ein Objekt zu ermitteln, das für die modulspezifische View zur Anzeige und zur Speicherung der vom Benutzer eingegebenen Informationen verfügbar gemacht wird.

Input-Parameter

Name	**Typ**	**Beschreibung**
exerciseID	int	ID des gefragten Beispiels.

Tabelle 4.5: Input-Parameter für das Ermitteln eines existierenden Beispiels

Output-Parameter

Das Modul liefert ein Objekt eines beliebigen modulspezifischen Typs, der aber zumindest das Interface `java.io.Serializable` implementieren und das Objekt dementsprechend serialisierbar sein muss. Dieses Objekt ist für die modulspezifischen Views zur Anzeige und zur Speicherung der vom Benutzer eingegebenen Informationen als Session-Attribut verfügbar (siehe Abschnitt 4.3). Dieses Objekt wird, nachdem eventuelle Änderungen von modulspezifischen Views in diesem Objekt zwischengespeichert wurden, wieder an die entsprechenden Methoden zum Ändern (Abschnitt 4.2.2) von modulspezifischen Informationen übergeben.

Wenn anhand der übergebenen ID zur Identifizierung des Beispiels kein entsprechender Eintrag im modulspezifischen Datenbestand gefunden werden kann, so sollte `null` zurückgeliefert werden.

Exceptions

Treten beim Abrufen des Beispiels Probleme auf, so sollte dies dem eTutor Core als Exception signalisiert werden.

4.2.6. Methode generateHtml

Aufrufer

Methode `generateAssignment` der Klasse `ExerciseAssignmentServlet`.

Aufrufzeitpunkt

Der Benutzer kann in einer vom eTutor Core bereitgestellten View den Angabetext aus den modulspezifischen Informationen zum Beispiel generieren lassen, sofern dies durch das Modul unterstützt wird. Dadurch ist es möglich, den Benutzer von Routine-Aufgaben zu entlasten.

Input-Parameter

Name	**Typ**	**Beschreibung**
exercise	Serializable	Modulspezifisches Objekt, das alle relevanten Informationen zu einem Beispiel enthält. Das Objekt entspricht dem Rückgabewert der in Abschnitt 4.2.4 und Abschnitt 4.2.5 erläuterten Methoden.
locale	Locale	Parameter, mit dem die gewünschte Sprache definiert wird.

Tabelle 4.6: Input-Parameter für das Generieren des Angabetextes

Output-Parameter

Wenn das Modul die Transformation der Informationen zu einem Beispiel in einen Angabetext in der gewünschten Sprache unterstützt, so sollte hier ein HTML-formatierter `String` zurückgeliefert werden. Dieser wird vom eTutor Core bei der Anzeige einer Aufgabe eines Studenten in das `body` Element eines vom eTutor Core zusammengestellten HTML-Dokumentes eingefügt.

Wenn das Modul diese Funktion, bzw. die gewünschte Sprache nicht unterstützt, so sollte `null` zurückgeliefert werden.

Exceptions

Treten beim Generieren des Angabetextes Probleme auf, so sollte dies dem eTutor Core als Exception signalisiert werden.

4.3. Request- und Session-Attribute

Die in der nachfolgenden Tabelle beschriebenen Attribute werden im Rahmen einer Sequenz von Schritten, die der Spezifikation eines neuen Beispiels oder eines zu ändernden Beispiels dienen, zur Kommunikation zwischen Ressourcen in der Web-Applikation verwendet. Attribute werden einerseits vom eTutor-System in der Benutzer-Session verwaltet, manche Attribute werden hingegen vor der Weiterleitung eines Requests (Methode `forward`) im Request-Objekt gespeichert. Diese Informationen sind einerseits für die Umsetzung der Eingabemasken von Bedeutung und andererseits für die Kommunikation zwischen Views des eTutor Core, bzw. zwischen Views des eTutor Core und modulspezifischen Views. Bei der Anzeige einer modulspezifischen Eingabemaske kann davon ausgegangen werden, dass diese Informationen den aktuellen Zustand der Benutzer-Session widerspiegeln. So sorgt der eTutor Core dafür, dass beispielsweise das Session-Attribut *moduleExercise* ein Objekt vom Typ ist, den die modulspezifische View zur Anzeige und zur Zwischenspeicherung von Benutzerangaben zu einem Beispiel verwendet.

Name	**Typ**	**Beschreibung**
userID	String	Siehe Abschnitt 3.3
moduleExercise	Serializable	Modulspezifisches Objekt zur Anzeige und Zwischenspeicherung von

		beispielbezogenen Informationen.
coreExerciseMgrCmd	String	Wird vom Modul verwendet, um beim Weiterleiten des Requests an das `ExerciseManagerServlet` die Aktion anzuzeigen, die der Benutzer ausgelöst hat. Mögliche Werte sind *back*, *next*, *cancel* und *finish*, wobei die Aktion *finish* für Module nicht relevant ist.
coreExerciseMgrStatus	String	Wird vom Modul verwendet, um beim Weiterleiten des Requests an das `ExerciseManagerServlet` den Status bei der Eingabe von Informationen für das Erzeugen oder Ändern von Informationen anzuzeigen. Mögliche Werte sind *moduleSettings*, *generalSettings*, *assignmentSettings* und *confirmation*. Von einem Modul sollte der Wert des Attributs auf *moduleSettings* gesetzt werden.

Tabelle 4.7: Attribute für die Beispieladministration

Name	**Verwendungszweck**
userID	Siehe Abschnitt 3.3
moduleExercise	In einer modulspezifischen View wird in diesem Session-Attribut ein Objekt erwartet, das die Informationen zu einem Beispiel repräsentiert. Alle Informationen, die das Objekt enthält, werden entweder in einer einzelnen Seite oder über mehrere Seiten verteilt für den Benutzer angezeigt. Umfasst die Spezifikation von modulspezifischen Informationen mehrere Seiten, so wird die Navigation zwischen diesen Seiten durch das Modul selbst bestimmt. Der eTutor Core fasst die modulspezifischen Views als Black Box auf und nimmt auf diese Navigation keinen Einfluss.
coreExerciseMgrCmd und coreExerciseMgrStatus	Wenn der Benutzer in der modulspezifischen View den Button *next* auswählt, und die View stellt die letzte von mehreren möglichen modulspezifischen Seiten für

	die Spezifikation eines Übungsbeispiels dar, so sollte das Modul diese beiden Attribute im Request speichern und den Request an das `ExerciseManagerServlet` weiterleiten. Für das Attribut *coreExerciseMgrCmd* sollte das Modul in diesem Fall den Wert ‚*next*' festlegen. Für das Attribut *coreExerciseMgrStatus* sollte vom Modul in jedem Fall der Wert ‚*moduleSettings*' festgelegt werden.

Tabelle 4.8: Attribute für die Beispieladministration (Verwendungszweck)

5. Remote Method Invocation (RMI)

In Kapitel 2 wurde bereits erwähnt, dass Module so implementiert werden sollten, dass sie nicht lokal für den eTutor Core verfügbar sein müssen, sondern als eigenständige Komponenten, die gegebenenfalls auf einem separaten Rechner installiert sind, in das eTutor-System integriert werden können. Der Aufruf der Auswertungskomponente (siehe Kapitel 3) und der Administrationskomponente (siehe Kapitel 4) eines entfernt installierten Moduls erfolgt in diesem Fall über *Remote Method Invocation* (RMI).

In Abschnitt 5.1 werden die Voraussetzungen beschrieben, die ein Modul im Sinne der RMI-Spezifikation der Java Plattform erfüllen muss, um über RMI in das eTutor-System integriert zu werden. In Abschnitt 5.2 wird hingegen der Vorgang der Integration in das eTutor-System beschrieben, sowie die Hilfsmittel, die das eTutor-System dazu zur Verfügung stellt. Während also Abschnitt 5.1 wichtige Anhaltspunkte für Entwurfsentscheidungen bei einem zu implementierenden Modul gibt, liefert Abschnitt 5.2 Informationen über die unmittelbare Konfiguration eines implementierten Moduls für die Kommunikation über RMI.

5.1. Voraussetzungen für die Integration eines Moduls über RMI

Um den RMI-Mechanismus zu unterstützen, müssen bei der Implementierung der Modulkomponenten eine Reihe spezieller Aspekte beachtet werden, die in Abbildung 5.1 verdeutlicht werden. Das XQuery-Modul dient dabei als Beispiel, um zu demonstrieren, wie sowohl die vom Core vorgegebenen Schnittstellen als auch die von der Java Plattform definierten RMI-spezifischen Schnittstellen durch die Komponenten des Moduls implementiert werden können.

- Für jede vom Modul implementierte Komponente muss ein modulspezifisches Interface definiert werden (`XQEvaluator` und `XQExerciseManager`). Im Gegensatz zu den entsprechenden Implementierungen (`XQEvaluatorImpl` und `XQExerciseManagerImpl`) müssen diese Interfaces für den Client, d.h. in diesem Fall für den eTutor Core, verfügbar sein.
- Die Interfaces erweitern das RMI-spezifische Interface `java.rmi.Remote`, womit eine wichtige Voraussetzung für die Kommunikation über RMI erfüllt ist. Die vom eTutor Core vorgegebenen Voraussetzungen für die Integration in das eTutor-System werden durch die Erweiterung der Interfaces `Evaluator` und `ModuleExerciseManager` erfüllt.
- Die RMI-Spezifikation sieht vor, dass jede im Client Interface (`XQEvaluator` und `XQExerciseManager`) definierte Methode zumindest eine Exception vom Typ `java.rmi.RemoteException` wirft. Nachdem in jeder Methode der vom eTutor Core vorgegebenen Interfaces eine allgemeine Exception vom Typ `java.lang.Exception` definiert ist, und `RemoteException` eine Spezialisierung dieser Exception ist, wird diese Bedingung automatisch erfüllt.
- Eine weitere Besonderheit der RMI-Spezifikation ist der Aspekt, dass die Instanz einer RMI-Klasse (d.h. in diesem Fall ein Objekt der Klasse `XQEvaluatorImpl` oder `XQExerciseManagerImpl`) nicht nur initialisiert, sondern auch „exportiert“ werden muss. Dadurch wird das Objekt in die Lage versetzt, über einen bestimmten Port entfernte Methodenaufrufe zu empfangen. Die Erweiterung der Klasse `UnicastRemoteObject`, wie in der Abbildung gezeigt, stellt dabei nur eine von mehreren Möglichkeiten dar, diesen Aspekt umzusetzen. Eine von `UnicastRemoteObject` abgeleitete Klasse wird bei der Initialisierung automatisch exportiert, wobei dies wiederum voraussetzt, dass in der Klasse ein Konstruktor ohne Input-Parameter existiert, der zumindest eine Exception vom Typ `RemoteException` wirft.
- Weiters muss beachtet werden, dass alle Parameter, die über die Methoden eines entfernten Objekten übergeben, bzw. empfangen werden, die von der Java Plattform definierten Voraussetzungen für Objekt-Serialisierung erfüllen müssen.

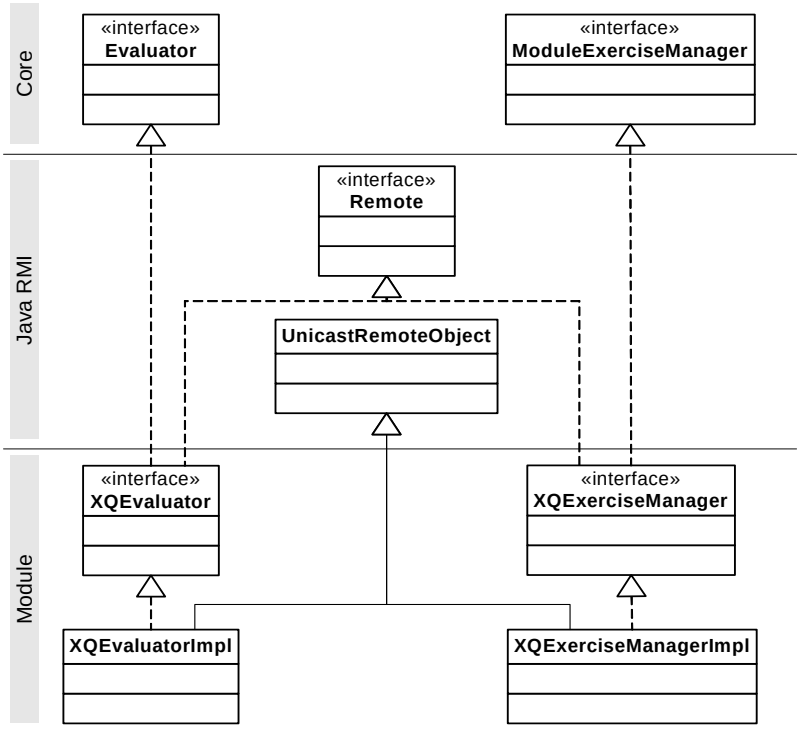

Abbildung 5.1: RMI-Unterstützung der Komponenten im XQuery-Modul

Aus den vom Modul bereitgestellten Komponenten für die Auswertung und Administration von Beispielen wird durch einen speziellen Compiler (*rmic*) eine Stub-Klasse erzeugt, die dem Client, d.h. in diesem Fall dem im Web-Container installierten eTutor Core, als Schnittstelle zur entfernt ausgeführten Komponente dient.

5.2. Integration eines Moduls über RMI

In diesem Abschnitt werden wichtige Aspekte erläutert, die die Integration eines Moduls in das eTutor-System über RMI betreffen. In Abschnitt 5.2.1 wird beschrieben, welche Abhängigkeiten zwischen dem eTutor Core und einzelnen Modulen bestehen, und wie diese bei der Verteilung des Systems berücksichtigt werden. In den darauf folgenden Abschnitten 5.2.2 und 5.2.3 wird erklärt, wie sich ein RMI-Modul mit den vom eTutor-System zur Verfügung gestellten Hilfsmitteln konfigurieren und starten lässt.

5.2.1. Deployment

Beim Deployment von Modulen, die außerhalb des Web-Containers laufen sollen, müssen Klassenabhängigkeiten zwischen dem im Web-Container ausgeführten

eTutor Core und den Modulen beachtet werden. Tabelle 5.1 enthält eine Zusammenfassung der unterschiedlichen Situationen, in denen diese Abhängigkeiten zur Geltung kommen, sowie eine Auflistung der davon betroffen Klassen und Interfaces. Die folgenden Situationen wurden dabei identifiziert:

- Ein Modul, das außerhalb der Web-Applikation ausgeführt wird, benötigt alle im Rahmen dieses Dokumentes beschriebenen Interfaces des eTutor Core, die in Tabelle 5.1 vollständig aufgezählt werden.
- Umgekehrt benötigt der eTutor Core beim Aufruf der Methode eines entfernten Objektes die Klassen, die die RMI-Schnittstelle darstellen, d.h. in erster Linie den Stub und das vom Modul definierte Interface. In Tabelle 5.1 sind die entsprechenden Klassen des XQuery-Moduls angeführt.
- Wenn in den Views, die das Modul zur Verfügung stellt (Servlets, JSPs, etc.), modulspezifische Klassen importiert werden, etwa indem eine Instanz einer modulspezifischen Klasse deklariert wird, so müssen diese Klassen für die statische Typüberprüfung ebenso im Web-Container verfügbar sein. Genauer betrachtet, kommt die statische Typüberprüfung bei der vom Servlet-Container durchgeführten Übersetzung von Servlets und JSPs in Klassendateien zum Tragen. Tabelle 5.1 zeigt beispielhaft eine nicht vollständige Aufzählung von Klassen für das XQuery-Modul.
- Ein weiterer kritischer Aspekt hinsichtlich der Verfügbarkeit von Klassendefinitionen ist die Serialisierung und Deserialisierung von modulspezifischen Objekten in der vom eTutor Core verwalteten Datenbasis. So werden etwa die modulspezifischen Report-Objekte, die von einem Modul bei der Auswertung von abgegebenen Lösungen geliefert werden (siehe Abschnitt 3.2.3), vom eTutor Core in einer zentralen Datenbank gespeichert. Wird ein solches Objekt zu einem späteren Zeitpunkt wieder deserialisiert, etwa zur Anzeige einer früheren Abgabe für einen Studenten, so müssen für den Deserialisierungs-Mechanismus alle Klassendefinitionen verfügbar sein. Tabelle 5.1 zeigt beispielhaft eine nicht vollständige Aufzählung von Klassen für das XQuery-Modul.

Situation	**Klassen**

Vom Modul benötigte Klassen des eTutor Core	`etutor.core.ModuleException`
	`etutor.core.evaluation.Analysis`
	`etutor.core.evaluation.DefaultAnalysis`
	`etutor.core.evaluation.DefaultGrading`
	`etutor.core.evaluation.DefaultReport`

	`etutor.core.evaluation.Evaluator`
	`etutor.core.evaluation.Grading`
	`etutor.core.evaluation.Report`
	`etutor.core.manager.ModuleExerciseManager`

Vom eTutor Core benötigte Klassen des Moduls für RMI	Z.B.
	`etutor.modules.xquery.XQEvaluator`
	`etutor.modules.xquery.XQEvaluatorImpl_Stub`

Von modulspezifischen Views im Web-Container benötigte Klassen des Moduls	Z.B.
	`etutor.modules.xquery.report.XQFeedback`
	`etutor.modules.xquery.report.ErrorCategory`
	etc.

Vom eTutor Core benötigte Klassen des Moduls für Deserialisierung	Z.B.
	`etutor.modules.xquery.report.XQFeedback`
	`etutor.modules.xquery.report.ErrorCategory`
	etc.

Tabelle 5.1: Klassenabhängigkeiten

Wie man Tabelle 5.1 entnehmen kann, werden vom eTutor Core in erster Linie die RMI-Schnittstellen eines Moduls benötigt. Davon abgesehen hängt es von den vom Modul bereitgestellten Views und den serialisierten Objekten ab, welche Klassendefinitionen im Web-Container, und somit auch für den eTutor Core zusätzlich verfügbar sein müssen. Diese Entscheidung muss demnach beim Deployment für jedes Modul individuell getroffen werden.

Bei der Bereitstellung der benötigten Klassendefinitionen für den eTutor Core und für die Module werden im eTutor-System zwei Ansätze verfolgt:

- Die benötigten Klassen werden im Rahmen des Deployment-Prozesses einfach in den eTutor Core, bzw. in das Modul kopiert. Dort werden sie lokal eingebunden, d.h. als *.class*- oder *.jar*-Dateien in den Klassenpfad aufgenommen.
- Eine Variante, die sauberer erscheint, ist die Definition einer sogenannten *codebase*. Eine *codebase* stellt einen Dienst dar, an den über das HTTP-Protokoll eine Anfrage gesendet werden kann, und der als Antwort die gewünschte Klasse überträgt, falls sie in der *codebase* verfügbar ist. Die *codebase* wird durch eine URL spezifiziert, und kann in einer Anwendung zur Ergänzung des Klassenpfades eingebaut werden.

Eine *codebase* wird derzeit nur für modulspezifische Klassen von entfernten Modulen eingerichtet (siehe Abschnitt 5.2.3). Die *codebase* erfüllt ihren Zweck bei der Kommunikation des eTutor Core mit einem Modul über RMI sowie bei der Deserialisierung von modulspezifischen Objekten. Im Gegensatz dazu gibt es derzeit keine vertretbare Lösung, über eine externe *codebase* auf das Laden von Klassen, die in Ressourcen (Java-Klassen, Servlets, JSPs) innerhalb der Web-Applikation importiert werden müssen, Einfluss zu nehmen. Aus diesem Grund werden die in den modulspezifischen Views benötigten Klassen im Rahmen des Deployment-Prozesses in die Web-Applikation kopiert. Aus dem selben Grund werden die von den Modulen benötigten Klassen des eTutor Core jeweils in das Modul kopiert, auch wenn eine zentrale *codebase* sowohl für den eTutor Core als auch die Module eine bessere Lösung darstellen würde.

5.2.2. RMI Server (registry)

Das Modul wird auf einem Rechner installiert, wo ein Prozess – die sogenannte RMI-Registry – gestartet wird. Dieser Prozess nimmt über einen Port Methodenaufrufe entgegen und leitet sie an die entsprechenden Objekte weiter. Diese Objekte stellen Instanzen von Klassen dar, die das `Remote`-Interface implementieren, und werden in der RMI-Registry mit einer Bezeichnung registriert. Die Bezeichnung wird zusammen mit der IP-Addresse und dem Port verwendet um die Objekte von einem anderen Rechner aus identifizieren zu können.

Das eTutor-System stellt ein Tool zum Starten einer RMI-Registry zur Verfügung. Eine durch dieses Tool gestartete Instanz einer RMI-Registry wird nachfolgend als RMI Server bezeichnet. Um einen RMI Server zu starten, wird die Java-Klasse `etutor.rmi.RMIServer` aufgerufen. Auf einem Rechner können mehrere Serverinstanzen gestartet werden, solange jeweils ein freier Port belegt wird. Beim Aufruf der Klasse kann ein Parameter übergeben werden, der den Pfad der XML-Datei für die Konfiguration des Servers angibt. Wenn kein Parameter übergeben wird, so wird standardmäßig nach einer Datei *remote.xml* im Verzeichnis, in dem das Programm ausgeführt wird, gesucht. Das folgende Beispiel demonstriert den Aufruf der Klasse:

```
java etutor.rmi.RMIServer remote.xml
```

Für einen solchen Aufruf müssen lediglich die Klassen aus dem Package `etutor.rmi` im Klassenpfad enthalten sein, sowie die XML-Java-Bibliothek

dom4j. Für das Ausführen des Befehls auf einem Windows-System steht die Datei *registry.bat* zur Verfügung.

Wie in Abschnitt 5.2.3 verdeutlicht wird, ist das Konfigurationsschema so konzipiert, dass nicht nur der RMI Server konfiguriert werden kann, sondern auf Basis derselben Konfigurationsdatei auch ein Classfile Server gestartet werden kann, der als *codebase* dient.

Abbildung 5.2 zeigt ein Beispiel einer solchen Konfigurationsdatei, in der ein RMI Server und ein Classfile Server für das Datalog- und das XQuery-Modul konfiguriert werden. In diesem Beispiel wird auch gezeigt, wie ein RMI Server, bzw. ein Classfile Server für mehrere Module gestartet wird, während auch die Möglichkeit besteht, für jedes Modul einen eigenen Server zu starten. Getrennte Server lassen sich auf demselben Rechner mit unterschiedlichen Ports starten, bzw. können überhaupt auf unterschiedlichen Rechnern eingesetzt werden.

Zum Starten eines RMI Servers muss ein auf dem Rechner noch nicht belegter Port im Element `rmi-startup` angegeben werden, während das Element `codebase-startup` für den RMI Server nicht von Bedeutung ist. Der Port wird in dem Beispiel mit `1099` angegeben, was dem Standard-Port einer RMI-Registry entspricht.

Jedes der folgenden `module`-Elemente repräsentiert ein Modul, in diesem Fall also das Datalog- und das XQuery-Modul. Die Module werden durch die im optionalen Element `module-name` angegebene Bezeichnung eindeutig voneinander unterschieden. Die eindeutige Bezeichnung garantiert, dass die Module voneinander getrennt innerhalb des Servers ausgeführt werden. Die Zusammenfassung der Konfiguration für das Datalog- und das XQuery-Modul in einem einzigen `module`-Element wäre zwar ebenso möglich, nachdem die Module aber dadurch nicht getrennt ausgeführt werden, können unerwünschte Nebeneffekte auftreten. Mehrere `module`-Elemente mit derselben Bezeichnung im `module-name`-Element werden darüber hinaus als einzelnes Modul aufgefasst.

Innerhalb des `modul`-Elements kann eine beliebige Menge von `codebase`-Elementen definiert werden, mit denen Ressourcen definiert werden, aus denen vom RMI Server der Klassenpfad eines Moduls zusammengesetzt wird. Mit den optionalen Elementen `host` und `port` können die Informationen angegeben werden, aus denen sich eine URL ableiten lässt, die einen Classfile Server lokalisiert. Im Falle des XQuery-Moduls wird aus den angegebenen Informationen die URL `http://127.0.0.1:2002/` abgeleitet. Dies entspricht der URL, die

standardmäßig abgeleitet wird, wenn das `port`-Element, nicht aber das `host`-Element angegeben wird. Damit wird im Standardfall angenommen, dass der Classfile Server auf demselben Rechner wie der RMI Server läuft. Wäre als Host beispielsweise `etutor1` eingetragen, so würde die URL `http://etutor1:2002/` abgeleitet werden.

In einer beliebigen Anzahl von `classes`-Elementen lassen sich die Verzeichnisse angeben, in denen die Klassen des auszuführenden Moduls zu finden sind. Zu beachten ist dabei, dass entweder absolute Pfade angegeben werden, oder zum Verzeichnis, in dem der RMI Server gestartet wird, relative Pfade. Mehrere Pfade lassen sich auch in einem einzigen `classes`-Element, jeweils getrennt durch ein Leerzeichen, einen Beistriche oder ein Semikolon, definieren.

Nach demselben Prinzip wie die Klassenverzeichnisse lassen sich in einer beliebigen Anzahl von `archive`-Elementen die Klassenbibliotheken angeben. Weist ein Pfad auf ein Verzeichnis, so werden alle in diesem Verzeichnis enthaltenen *.jar*-Dateien in den Klassenpfad aufgenommen, ansonsten wird der Pfad als Datei interpretiert und unverändert in den Klassenpfad aufgenommen.

Schließlich werden in einer beliebigen Menge von `mapping`-Elementen die RMI-Klassen definiert, die unter einer bestimmten Bezeichnung in der RMI-Registry registriert werden sollen. Im Falle des Datalog-Moduls wird z.B. beim Starten des RMI Servers u.a. ein Objekt der Klasse `DatalogEvaluatorImpl`, die der Auswertung von Beispielen dient, unter dem Namen *DatalogEvaluator* registriert. Spätestens zu diesem Zeitpunkt werden die Ressourcen benötigt, die wie oben beschrieben für das Modul angegeben wurden (Codebases, Klassen und Klassenbibliotheken).

```
<remote-etutor>
    <rmi-startup>
        <port>1099</port>
    </rmi-startup>
    <codebase-startup>
        <port>2002</port>
    </codebase-startup>
    <module>
        <module-name>datalog</module-name>
        <codebase>
            <host>127.0.0.1</host>
            <port>2002</port>
            <classes>./modules/datalog/classes</classes>
            <archives>./modules/datalog/lib</archives>
        </codebase>
        <mapping>
            <name>DatalogEvaluator</name>
            <object>etutor.modules.datalog.DatalogEvaluatorImpl</object>
        </mapping>
```

```
            <mapping>
                <name>DatalogExerciseManager</name>
                <object>
                 etutor.modules.datalog.exercise.DatalogExerciseManagerImpl
                </object>
            </mapping>
        </module>
        <module>
            <module-name>xquery</module-name>
            <codebase>
                <host>127.0.0.1</host>
                <port>2002</port>
                <classes>./modules/xquery/classes</classes>
                <archives>./modules/xquery/lib</archives>
            </codebase>
            <mapping>
                <name>XQEvaluator</name>
                <object>etutor.modules.xquery.XQEvaluatorImpl</object>
            </mapping>
            <mapping>
                <name>XQExerciseManager</name>
                <object>
                 etutor.modules.xquery.exercise.XQExerciseManagerImpl
                </object>
            </mapping>
        </module>
</remote-etutor>
```

Abbildung 5.2: Konfiguration des RMI Servers und des Classfile Servers

Die folgenden Beispiele sollen aus der Sicht des eTutor Core den Unterschied zwischen einem lokal verfügbaren Modul und einem Modul, das über RMI eingebunden wird, verdeutlichen. Bei einem lokal verfügbaren Modul benötigt der eTutor Core den vollen Namen der modulspezifischen Klasse, die ein vom eTutor Core vorgegebenes Interface implementiert. Unter der Annahme, dass das XQuery-Modul lokal verfügbar ist und die modulspezifische Auswertungskomponente `Evaluator` benötigt wird, würde der Standard-Mechanismus zum dynamischen Laden von Klassen angewandt. Über die statische Methode `forName` der Klasse `Class` wird aus dem Klassenpfad die Klasse geladen, die dem angegebenen Klassennamen entspricht. Im zweiten Schritt wird von dieser Klasse eine neue Instanz erzeugt.

```
Class c = Class.forName("etutor.modules.xquery.XQEvaluatorImpl");
Evaluator evaluator = (Evaluator)c.newInstance();
```

Im Gegensatz dazu wird ein entferntes Objekt über den Rechnernamen, den Port und die Bezeichnung, unter der das Objekt in der RMI-Registry registriert ist, identifiziert. Die Auswertungskomponente wird in diesem Fall über die statische Methode `lookup` der Klasse `Naming` geladen. Der übergebene Name entspricht dem Konfigurationsbeispiel aus Abbildung 5.2, wobei angenommen wird, dass der RMI Server auf dem Rechner `etutor1` läuft.

```
Object obj = Naming.lookup("//etutor1:1099/XQEvaluator");
Evaluator evaluator = (Evaluator)obj;
```

Ein weiterer Unterschied zeigt sich darin, dass ein lokales Objekt neu initialisiert wird (Methode `newInstance`), während über RMI eine Kommunikation zu einem entfernten Objekt aufgebaut wird, das bereits initialisiert und in der RMI-Registry eingetragen ist.

5.2.3. Classfile Server (codebase)

Abgesehen von einem Tool zum Starten eines RMI Servers beinhaltet das eTutor-System auch ein Tool zum Starten eines Classfile Servers. Eine Instanz eines Classfile Servers startet einen Prozess, der über einen bestimmten Port HTTP-Anfragen entgegennimmt, über die ein Anwendung versucht, Klassen zu laden. Wenn die Klasse als *.class*-Datei oder in einer *.jar*-Datei für den Classfile Server verfügbar ist, so wird diese Klasse an die aufrufende Instanz übermittelt. Mit dem in Abbildung 5.2 dargestellten Konfigurationsbeispiel lässt sich nicht nur ein RMI Server, sondern gleichermaßen ein Classfile Server starten, der über eine URL als *codebase* für die Module und den eTutor Core eingesetzt wird.

Um einen Classfile Server zu starten, wird die Java-Klasse `etutor.rmi.ClassFileServer` aufgerufen. Auf einem Rechner können mehrere Serverinstanzen gestartet werden, solange jeweils ein freier Port belegt wird. Beim Aufruf der Klasse kann ein Parameter übergeben werden, der den Pfad der XML-Datei für die Konfiguration des Servers angibt. Wenn keine Datei angegeben wird, so wird standardmäßig nach einer Datei *remote.xml* im Verzeichnis, in dem das Programm ausgeführt wird, gesucht. Das folgende Beispiel demonstriert den Aufruf der Klasse:

```
java etutor.rmi.ClassFileServer remote.xml
```

Für einen solchen Aufruf müssen lediglich die Klassen aus dem Package `etutor.rmi` im Klassenpfad enthalten sein, sowie die XML-Java-Bibliothek dom4j. Für das Ausführen dieses Befehls auf einem Windows-System steht die Datei *codebase.bat* zur Verfügung.

Zum Starten eines Classfile Servers muss in der Konfigurationsdatei (siehe Abbildung 5.2) ein auf dem Rechner noch nicht belegter Port im Element `rmi-codebase` angegeben werden, während das Element `rmi-startup` für den Classfile Server nicht von Bedeutung ist.

Hinsichtlich der `module`-Elemente ist zu beachten, dass im Gegensatz zum RMI Server keine Trennung nach Modulen erfolgt, falls in der Konfigurationsdatei wie in dem gezeigten Beispiel unterschiedliche Module definiert werden. Dies bedeutet in erster Linie, dass der Classfile Server die erste Klasse liefert, die auf eine an den Server gerichtete HTTP-Anfrage zutrifft, egal ob die Klasse in den angegebenen Ressourcen des Datalog- oder des XQuery-Moduls enthalten ist (Codebases, Klassen, Klassenbibliotheken).

Eine Einschränkung der für einen Classfile Server relevanten Ressourcen kann jedoch durch die `host`- und `port`-Elemente innerhalb der `codebase`-Elemente getroffen werden. Der Classfile Server registriert nur diejenigen Ressourcen der `codebase`-Elemente, die auf den zu startenden Classfile Server referenzieren. Dies wird festgelegt, indem im `port`-Element derselbe Port wie im `codebase-startup`-Element eingetragen wird. Zusätzlich muss gelten, dass entweder kein `host`-Element existiert, oder der Wert des `host`-Elements `localhost` oder `127.0.0.1` ist.

Alle innerhalb der relevanten `codebase`-Elemente angegebene Klassen und Klassenbibliotheken werden zum Startzeitpunkt des Classfile Servers registriert. Wie bereits erwähnt trifft der Classfile Server dabei keine Unterscheidung, zu welchen Modulen diese Ressourcen zugeordnet werden, d.h. in diesem Fall ist es egal, in welchen `module`-Elementen diese Ressourcen definiert werden. Die Klassen werden aus den `classes`-Elementen wie beim Starten eines RMI Servers ausgelesen, ebenso wie die Klassenbibliotheken aus den `archive`-Elementen (siehe Abschnitt 5.2.2). Die `mapping`-Elemente sind für den Classfile Server ohne Bedeutung.

Im Endeffekt wird in dem gezeigten Beispiel eine *codebase* initialisiert, die unter der Annahme, dass der Rechner im Netzwerk als `etutor1` identifiziert wird, über die URL `http://etutor1:2002/` referenziert werden kann. Die *codebase* umfasst alle Ressourcen, die für das Datalog- und das XQuery-Modul definiert sind, da in jedem der `codebase`-Elemente dieser Module der Port des Classfile Servers und der Host mit `127.0.0.1` angegeben wird.

6. Beispiel: RDBD-Modul

Anhand des Moduls für *Relational Data Base Design* (RDBD) wird in diesem Kapitel die Implementierung eines Moduls demonstriert. Dazu wird beschrieben, wie die vom eTutor Core für die Integration in das eTutor-System vorgegebenen Schnittstellen für die Auswertung und Administration von Übungsbeispielen umgesetzt werden. Während für umfassende Informationen zu diesem Modul auf die Dokumentation des RDBD-Moduls verwiesen wird, dient dieses Kapitel u.a. dazu, um Richtlinien anzusprechen, die bei der Implementierung eines Moduls beachtet werden sollten. Dies betrifft Konventionen wie etwa die Strukturierung des Moduls.

Hinsichtlich der Kommunikation zwischen eTutor Core und dem RDBD-Modul muss angemerkt werden, dass das RDBD-Modul derzeit entfernte Methodenaufrufe über RMI (siehe Kapitel 5) nicht unterstützt und daher im Web-Container des eTutor-Systems installiert wird.

Das RDBD-Modul umfasst mehrere unterschiedliche Beispieltypen, u.a. für Normalisierung, Bestimmung von Schlüsseln oder Berechnung der Minimalen Überdeckung. Diese Beispieltypen sind aus Sicht des eTutor Core jeweils als eigenes Modul in das eTutor-System integriert sind. Damit stellt das RDBD-Modul einen speziellen Fall für die Implementierung eines Moduls dar, aufgrund seiner generischen Umsetzung kann das Modul aber dennoch als Referenz-Implementierung herangezogen werden.

Der Code des Moduls ist in den Dateien *core.zip* und *rdbd.zip* enthalten. In der Datei *core.zip* befinden sich die JAVA source files des eTutor Cores, die für das Evaluieren von Studenten-Abgaben und für die Administration von Übungsbeispielen benötigt werden.

Die Methode `evaluate` der Klasse `CoreManagerServlet` ist der Einstiegspunkt der Evaluierung von Studenten-Abgaben. Dieses Servlet des eTutor Core veranlasst das Analysieren, Bewerten und das Erstellen von Feedback durch die

vom Modul implementierte Komponente. Diese Komponente ist die Schnittstelle zum Modul, über die die Auswertung von Studenten-Lösungen ermöglicht wird. Dazu muss diese Komponente das Interface `Evaluator` implementieren, in dem die drei Methoden `analyze`, `grade` und `report` definiert sind.

Analog dazu befinden sich in der Klasse `ExerciseManagerServlet` die Einstiegspunkte für die Administration von Übungsbeispielen. Von dort werden durch den eTutor Core die entsprechenden Methoden der Komponente aufgerufen, die für einen bestimmten Beispieltyp des RDBD-Moduls das Interface `ModuleExerciseManager` implementiert.

In der Datei *rdbd.zip* befinden sich die JAVA-Quelldateien sowie HTML-, JavaScript- und CSS Dateien, die das RDBD-Modul umfasst.

6.1. Ordnerstruktur

Das Projekt wird auf dem Institut für Data & Knowledge Engineering über CVS (Concurrent Versions System) verwaltet. Die nachfolgende Ordnerstruktur dient einerseits der Strukturierung des Projekts entsprechend den Namenskonventionen für Web-Applikationen und andererseits einer einfachen Verwaltung der Quelldateien über *ant build files*. Ant ist ein Java basiertes Softwarewerkzeug welches mit dem UNIX Tool make verglichen werden kann und das Deployment und die Kompilierung eines verteilten Projekts erheblich vereinfacht.

Diese Struktur kann als Referenz-Struktur für die Implementierung von Modulen betrachtet werden.

Verzeichnis	**Beschreibung**
src	JAVA source files
lib	JAVA libraries
ui/modules/rdbd	HTML- und JSP source files
ui/modules/rdbd/js	JavaScript files
ui/modules/rdbd/css	CSS files

Tabelle 6.1: Ordnerstruktur des RDBD-Moduls

6.2. Java-Packages

Package	**Beschreibung**
etutor.modules.rdbd	Diverse Klassen des RDBD-Moduls.

etutor.modules.rdbd.algorithms	Klassen, die Algorithmen wie `Closure` oder `Cover` implementieren.
etutor.modules.rdbd.analysis	Klassen, die für das Analysieren einer Studenten-Abgabe benötigt werden.
etutor.modules.rdbd.exercises	Klassen, die für die Administration von Beispielen benötigt werden.
etutor.modules.rdbd.model	Klassen, die eine Studenten-Abgabe darstellen.
etutor.modules.rdbd.report	Klassen, die für das Erstellen von Feedback (Report) benötigt werden.
etutor.modules.rdbd.ui	Klassen, die ausschließlich für das User Interface benötigt werden.

Tabelle 6.2: Java-Packages des RDBD-Moduls

6.3. Anzeige des User Interface

Das User Interface des RDBD-Moduls stellt für die Auswertung von Studentenlösungen verschiedene Editoren (`showEditor.jsp`) zur Verfügung, die jeweils für einen bestimmten Beispieltyp gedacht sind (z.B. Bestimmung von Schlüsseln, Berechnung von Minimaler Überdeckung, etc.). Jeder dieser Editoren basiert auf dem gleichen JAVA-Modell. Deshalb implementieren die Editoren keinerlei Business Logic (Code zum Manipulieren des JAVA-Modells), sondern dienen lediglich zur Anzeige des jeweiligen User Interface. Die eigentliche Business Logic wird bei der Auswertung von Studentenlösungen von einem Servlet (`RDBDEditorServlet`) gemeinsam für alle Editoren bereitgestellt. Alle HTTP-Requests gehen somit zuerst an dieses Servlet, werden dort abgearbeitet (d.h. das JAVA-Modell wird entsprechend dem jeweiligen Request manipuliert) und abschließend an den jeweiligen Editor zur Visualisierung weitergeleitet.

Für die Administration von Beispielen stellt das RDBD-Modul analog dazu ebenfalls eigene Editoren für jeden Beispieltyp zur Verfügung (`specifyExercise.jsp`), sowie ein zentrales Servlet, das die Aufgaben der Business Logic übernimmt (`RDBDExerciseManagerServlet`). Das User Interface für die Administration von Übungsbeispielen ist so konzipiert, dass es keinen Unterschied macht, ob ein Beispiel neu zu erstellen ist, oder ob ein existierendes Beispiel bearbeitet werden soll.

6.4. Implementierung der Auswertungskomponente

Die Auswertungskomponente, die durch das vom eTutor Core vorgegebene Interface `Evaluator` spezifiziert ist, wird im RDBD-Modul durch die Klasse `RDBDEvaluator` implementiert.

6.4.1. Analysieren einer Studenten-Abgabe

Die Methode `analyze` implementiert die Analyse einer Studenten-Abgabe. Diese Methode veranlasst das Analysieren der Studenten-Abgabe entsprechend dem jeweiligen Beispieltyp (z.B. Bestimmung von Schlüsseln, etc.). Die konkreten Analysen werden von eigenen Klassen erstellt. Die Analyse für den Beispieltyp ‚Bestimmung von Schlüsseln' wird beispielsweise von der Klasse `KeysAnalyzer` erstellt und in einem Objekt der Klasse `KeysAnalysis` dargestellt. Das Ziel einer Analyse ist es, dass soviel Information wie möglich gewonnen wird, um möglichst aussagekräftiges Feedback geben zu können. In der momentanen Implementierung des RDBD-Moduls wird nur das Delta zwischen der richtigen Lösung und der Lösung des Studenten in der Analyse festgehalten. Bei der Entwicklung eines Moduls sollte der Fokus jedoch nicht allzu sehr auf dem Delta liegen, sondern auf der Darstellung der Studenten-Lösung und der richtigen Lösung.

Analysen können auch aus verschiedenen Teil-Analysen bestehen. So besteht beispielsweise die Analyse einer Studenten-Abgabe zum Beispieltyp Normalisierung (`NormalizationAnalysis`) unter anderem aus Analysen der Schlüssel (`KeysAnalysis`) der normalisierten Teilrelationen. Diesem Beispiel sollte gefolgt werden, um die Analysen, sofern es sinnvoll ist, modular und leicht wiederverwendbar als Teil-Analysen einsetzen zu können.

6.4.2. Bewerten einer Studenten-Abgabe

Die Methode `grade` implementiert das Bewerten einer Studenten-Abgabe. Im RDBD Modul ist die Bewertung einer Studenten-Abgabe rudimentär implementiert. D.h. ist die Studenten-Abgabe zu hundert Prozent korrekt, erhält der Student alle Punkte. Anderenfalls erhält der Student Null Punkte. Eine derartige Bewertungs-Funktionalität ist die Mindestanforderung an zu implementierte Module.

6.4.3. Erstellen von Feedback zu einer Studenten-Abgabe

Die Methode `report` implementiert das Erstellen von Feedback entsprechend einer bestimmten Analyse. Diese Methode veranlasst das Erstellen von Feedback entsprechend dem jeweiligen Beispieltyp. Das konkrete Feedback wird von eigenen Klassen erstellt. Z.B. wird das Feedback für den Beispieltyp ‚Bestimmung von Schlüsseln' von der Klasse `KeysReporter` erstellt und in einem Objekt der Klasse `Report` im Package `etutor.modules.rdbd.report` dargestellt. Im Unterschied zur Analyse, wo für jeden Beispieltyp eine eigene Klasse zur Repräsentation der jeweiligen Analyse implementiert wurde, wird beim Erstellen des Feedbacks im RDBD Modul immer die Klasse `Report` zur Repräsentation des Feedbacks verwendet, unabhängig vom konkreten Beispieltyp. Dadurch ist für die Anzeige des Feedbacks lediglich eine JSP Seite (`printReport.jsp`) erforderlich.

6.5. Implementierung der Administrationskomponente

Die Administrationskomponente, die durch das vom eTutor Core vorgegebene Interface `ModuleExerciseManager` spezifiziert ist, wird im RDBD-Modul durch die Klasse `RDBDExercisesManager` implementiert. Die modulspezifischen Informationen zu einem Übungsbeispiel werden durch ein Objekt der Klasse `SpecificationEditor` repräsentiert. Ein Objekt dieser Klasse wird einerseits als Input- und Output-Parameter der nachfolgend beschriebenen Methoden verwendet, und andererseits zur Anzeige und zur Speicherung der vom Benutzer eingegebenen Informationen in den JSP-Seiten, die vom RDBD-Modul für die Administration von Übungsbeispielen bereitgestellt werden.

6.5.1. Erstellen eines Übungsbeispiels

Die Methode `createExercise` implementiert die Funktionalität zur Erstellung eines Übungsbeispiels in der Datenbasis des Moduls. Die Informationen werden einem Objekt entnommen, das vom eTutor Core an diese Methode übergeben werden, und das vom Typ `SpecificationEditor` sein muss.

6.5.2. Ändern eines Übungsbeispiels

Die Methode `modifyExercise` implementiert die Funktionalität zur Änderung eines Übungsbeispiels in der Datenbasis des Moduls. Die Informationen werden einem Objekt entnommen, das vom eTutor Core an diese Methode übergeben werden, und das vom Typ `SpecificationEditor` sein muss.

6.5.3. Löschen eines Übungsbeispiels

Die Methode `deleteExercise` implementiert die Funktionalität zum Löschen eines Übungsbeispiels aus der Datenbasis des Moduls.

6.5.4. Abfrage eines neuen Beispiels

Die Methode `fetchExerciseInfo` wird vom eTutor Core aufgerufen um beim Erstellen eines neuen Beispiels ein Objekt zu ermitteln, das die modulspezifischen Informationen eines neu initialisierten Beispiels repräsentiert. Das vom RDBD-Modul übergebene Objekt ist vom Typ `SpecificationEditor` und wird vom eTutor Core an die modulspezifischen Views für die Administration von Übungsbeispielen, d.h. die in Abschnitt 6.3 erwähnten Servlets und JSP-Seiten, zur Manipulation übergeben. Nachdem alle Informationen eingegeben wurden, initiiert der eTutor Core die tatsächliche Erzeugung des Beispiels, indem es das Objekt über die entsprechende Methode (siehe Abschnitt 6.5.1) wieder an das RDBD-Modul übergibt.

6.5.5. Abfrage eines existierenden Beispiels

Die Methode `fetchExercise` wird vom eTutor Core aufgerufen um beim Ändern eines existierenden Beispiels ein Objekt zu ermitteln, das die modulspezifischen Informationen dieses Beispiels repräsentiert. Das vom RDBD-Modul übergebene Objekt ist vom Typ `SpecificationEditor` und wird vom eTutor Core an die modulspezifischen Views für die Administration von Übungsbeispielen, d.h. die in Abschnitt 6.3 erwähnten Servlets und JSP-Seiten, zur Manipulation übergeben. Nachdem alle Informationen eingegeben wurden, initiiert der eTutor Core die tatsächliche Änderung des Beispiels, indem es das Objekt über die entsprechende Methode (siehe Abschnitt 6.5.2) wieder an das RDBD-Modul übergibt.

6.5.6. Automatische Generierung eines Angabetextes

Durch Implementierung der Methode `generateHtml` unterstützt das RDBD-Modul die Funktionalität, aus der Spezifikation eines Übungsbeispiels automatisch einen Angabetext zu generieren. Die Informationen zum Übungsbeispiel werden dem übergebenen Objekt entnommen, das vom Typ `SpecificationEditor` sein muss. Über einen weiteren Parameter wird die gewünschte Sprache definiert, wobei das RDBD-Modul die Generierung in deutscher und englischer Sprache unterstützt.