eTutor

Modularchitektur und Moduldokumentation

XQuery und Datalog

Georg Nitsche

Version 1.1
Stand März 2006

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

dke
Data & Knowledge Engineering

Versionsverlauf:

Version	***Autor***	***Datum***	***Änderungen***
1.0	gn	14.02.2005	Fertigstellung der ersten Version
2.0	gn	23.02.2006	Überarbeitung
	gn	06.03.2006	Einfügen einer Referenz auf die Modularchitektur
	gn	07.03.2006	Anpassung nicht aktueller Inhalte

Inhaltsverzeichnis:

Abbildungsverzeichnis:

Tabellenverzeichnis:

Kurzfassung

In dieser Arbeit geht es um die Beschreibung der Entwicklung von Analyse- und Bewertungsmodulen, die in das *Electronic Tutor System* integriert werden sollen. Das System ist vor allem für Studenten konzipiert und wird in dieser Arbeit durch die Bezeichnung *eTutor* benannt. Mit den Modulen sollen spezielle Themengebiete abgedeckt werden, die aus den Vorlesungs- und Übungsinhalten des *Instituts für Data & Knowledge Engineering* in Linz entnommen sind. Bei den entwickelten Modulen, die in dieser Arbeit beschrieben werden, handelt es sich einerseits um ein Modul für den Themenschwerpunkt **XQuery**, sowie ein Modul für die Datenbank-Abfragesprache **Datalog**.

1. Einleitung

Mithilfe des eTutor-Systems soll es Benutzern ermöglicht werden, Lösungsvorschläge zu Problemstellungen aus Vorlesungen und Übungen online zu bearbeiten. Generelles Ziel ist dabei, den Lerneffekt bei der Bearbeitung von Themen aus dem Bereich des Data & Knowledge Engineering durch gezielt eingesetzte Hilfestellungen zu erhöhen. Studenten erhalten unmittelbar Rückmeldungen und Hinweise auf mögliche Fehler in den Ausarbeitungen, wobei die Detailliertheit dieses Feedbacks über Parameter konfiguriert werden kann.

Die Funktionen, die die Module realisieren sollen, lassen sich in die folgenden Kernfunktionen untergliedern:

- Analyse von Abfrage-Ausdrücken,
- Bewertung der Analyseergebnisse, sowie die
- Generierung eines Feedbacks über Analyse und Bewertung der Abgabe.

Im Falle des XQuery-Moduls davon XQuery-Ausdrücke betroffen, mit deren Hilfe XML-Daten abgefragt werden sollen. Der Vergleich einer vom Benutzer eingegebenen Abfrage mit einer Musterlösung stellt die Analyse dar. Dabei wird versucht, eventuelle Fehler und idealerweise die zugrunde liegenden Ursachen für die Fehler zu erkennen. Im nächsten Schritt kann die Korrektheit, bzw. können die gefundenen Fehler, bewertet werden. Letztendlich werden Abfrageergebnis, Analysen und Bewertungen für die Rückmeldung an den Benutzer aufgearbeitet und entsprechende Informationen generiert.

Analog zu dieser Funktionalität soll das zweite Modul für Datalog realisiert werden, mit dem es möglich sein soll, Ergebnisse von Datalog-Regeln und -Fakten zu vergleichen. Abhängig von Art und Umfang der Übereinstimmung sollen auch hier Hinweise auf mögliche Fehlerquellen gegeben werden.

In Kapitel 2 erfolgt vorerst eine allgemeine Beschreibung der Anforderungen an die Funktionalität der entwickelten Module. Diese Beschreibung trifft auf das

Datalog- und das XQuery-Modul gleichermaßen zu. Das Kapitel 3 enthält Erläuterungen zu den Lösungsansätzen, die verfolgt wurden, um die in Kapitel 2 beschriebenen Anforderungen zu erfüllen. Dabei werden in getrennten Abschnitten die Lösungsansätze des Datalog-Moduls und des XQuery-Moduls beschrieben, die in wesentlichen Punkten vergleichbar sind. Darauf folgend wird in 4 die Systemstruktur der entwickelten Module dokumentiert. Dies umfasst die Architektur und die Schnittstellen mit dem eTutor-System, sowie wesentliche Punkte der Implementierung, die mit der Programmiersprache *Java* durchgeführt wurde. Die Möglichkeiten, konkrete Übungsaufgaben für das Datalog- und das XQuery-Modul zu definieren, werden in Kapitel 5 behandelt, die grundlegenden Konfigurationsmöglichkeiten für die Module hingegen in Kapitel 6. Abgeschlossen wird diese Arbeit mit einer Zusammenfassung in Kapitel 7.

2. Anforderungen

Dieses Kapitel beschreibt die geforderte Funktionalität, die beide Module gleichermaßen erfüllen müssen. Dies schließt eine Beschreibung der Eingabe- und Ausgabedaten, sowie der zu behandelnden Systemfehler mit ein. Diese Fehler sind nicht zu verwechseln mit Fehlern, die in den Abgaben von Benutzern analysiert werden. Die Funktionalitäten lassen sich in die Kernfunktionen *Analyse*, *Bewertung* und *Feedback* untergliedern.

Der generelle Ablauf bei der Benutzung eines Moduls wird in Abbildung 2.1 dargestellt: Der Benutzer wählt über einen Browser eine konkrete Übungsaufgabe im eTutor-System aus (*showTask.jsp*). Die ausgewählte Aufgabe kann daraufhin auf einer Seite, die mit dem jeweiligen Modul mitgeliefert wird, ausgearbeitet werden (*showEditor.jsp*). Diese Seite wird wie alle übrigen dynamischen Seiten mittels JSP (Java Server Pages) realisiert. Durch die Auswahl der Übungsaufgabe werden an diese Seite Parameter übermittelt, von denen vor allem die folgenden für das Modul relevant sind:

- *typeID*: Dieser Parameter identifiziert das Modul (z.B. das Datalog- oder das XQuery-Modul).
- *exerciseID*: Dieser Parameter dient dazu, die Übungsaufgabe, die am Server gespeichert ist, zu identifizieren.
- *modeID*: Mit diesem Parameter wird festgelegt, ob die Beispielausarbeitung im Übungs- oder im Abgabemodus erfolgt.

Wenn der Benutzer auf der Seite *showEditor.jsp* die Abgabe bestätigt, werden die oben erwähnten Parameter und die ausgearbeitete Lösung an das *CoreManagerServlet* des eTutor-Systems übermittelt. Von dort werden die Daten an das Modul weitergeleitet, das dem Parameter *typeID* zugeordnet ist, und die Methoden für die Analyse, die Bewertung und das Feedback aufgerufen. Diese befinden sich in einer Klasse, die das eTutor-Interface *etutor.core.evaluation.Evaluator* implementiert. Die Ergebnisse werden für den Benutzer letztendlich auf einer Seite *printReport.jsp* dargestellt, die wiederum mit dem jeweiligen Modul mitgeliefert wird.

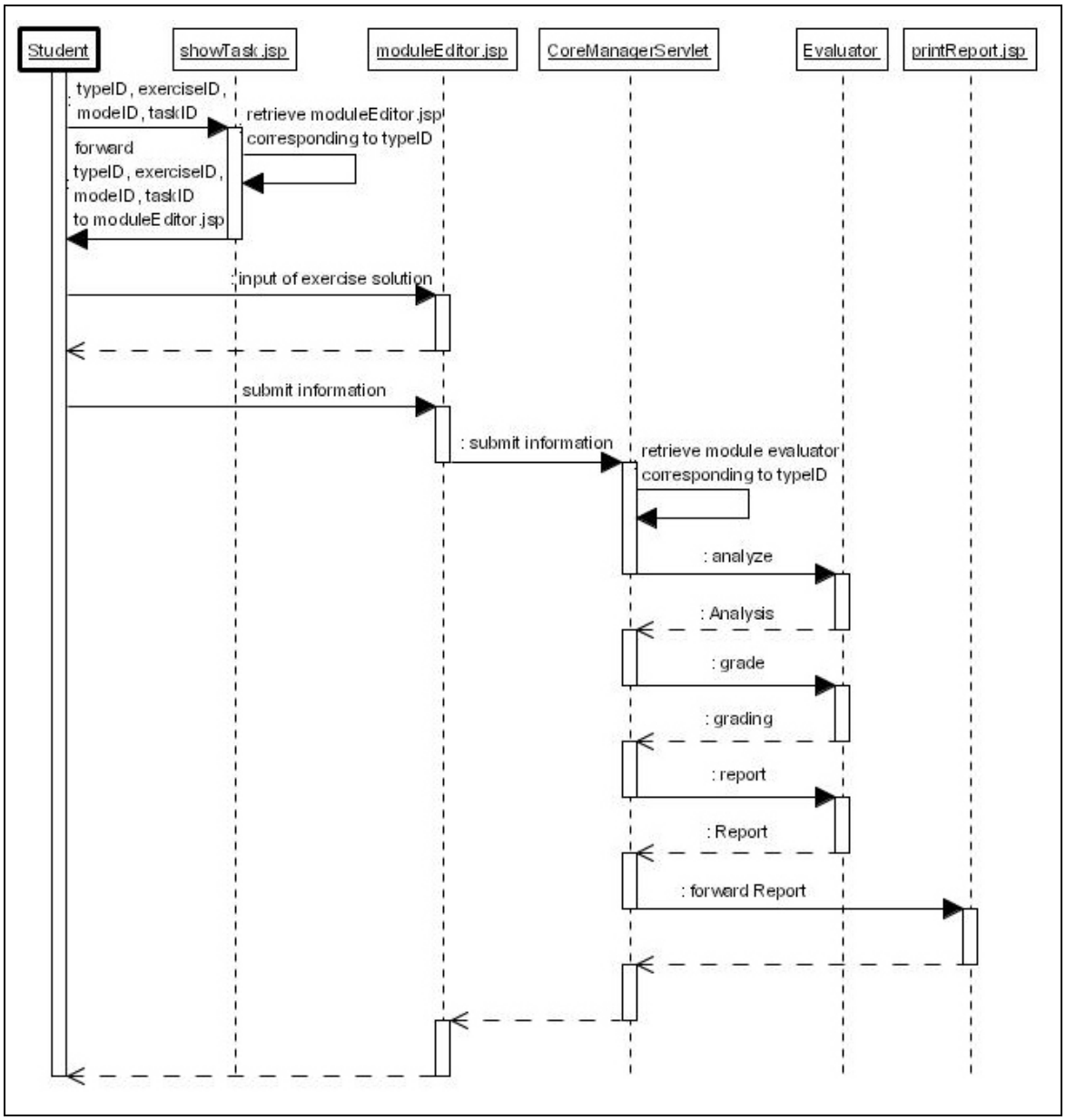

Abbildung 2.1: Aufruf eines Moduls im eTutor

2.1. Analyse

Die Analysefunktion stellt die Kernfunktion der Module dar und dient als Basis für die weiteren Funktionen. Die Funktionalität wird über die folgende Methode, die im eTutor-Interface *etutor.core.evaluation.Evaluator* definiert ist, aufgerufen:

analyze(int exerciseID, int userID, Map passedAttributes, Map passedParameters)

2.1.1. Funktionalität

Zwei Abfrage-Ausdrücke werden auf einem vorgegebenen Datenbestand ausgewertet: Einerseits der Ausdruck der die *Musterlösung* darstellt und mit dem Datenbestand am Server gespeichert ist, und andererseits der Ausdruck, der vom Benutzer eingegeben wird (die *Abgabe*). Das Auswertungsergebnis, das auf der Abgabe des Benutzers basiert, wird mit dem Auswertungsergebnis der Musterlösung verglichen und die Unterschiede zu definierten Fehlerkategorien zugeordnet.

Für die Module müssen demnach Fehlerkategorien und Zusammenhänge zwischen diesen Fehlerkategorien identifiziert werden, die für XQuery- bzw. Datalog-Abfragen typisch sind. Auf der Basis dieser Fehlerkategorien kann bei der Analyse eines Fehlers auf typische Fehlerursachen geschlossen werden.

Gegenstand der Analyse ist demnach weniger die Struktur des Abfrage-Ausdruckes selbst, sondern das Ergebnis, das durch die Auswertung der Abfrage geliefert wird. Die Identifizierung der Fehler basiert somit auf einem „heuristischen“ Verfahren, d.h. aus dem Auswertungsergebnis wird etwa geschlossen, dass der Abfrageausdruck mit hoher Wahrscheinlichkeit bestimmte Konstrukte enthält, die dazu führen, dass im Ergebnis zu viele Elemente geliefert werden.

2.1.2. Eingabedaten

Die Abfrage des Benutzers wird über eine Internetverbindung als Zeichenkette an das Modul übermittelt. Die Musterlösung für die Aufgabe wird aus der Datenbank gelesen, die vom Modul für die Verwaltung von modulspezifischen Informationen zu Übungsbeispielen verwendet wird. Die Datenbasis, auf die diese Abfragen angewandt wird, wird ebenso aus dieser Datenbank gelesen.

Im Falle des XQuery-Moduls liegt diese Datenbasis in Form eines oder mehrerer XML-Dokumente vor, die für die Auswertung einer XQuery-Abfrage benötigt werden. Die XML-Dokumente können in der Datenbank, die vom XQuery-Modul für die Verwaltung von Informationen zu Übungsbeispielen verwendet wird, gespeichert werden.

Diese Informationen werden aus den Eingabedaten ermittelt, die über die Methode *analyze* übergeben werden:

- *exerciseID*: identifiziert die Übungsaufgabe, d.h. die Musterlösung und die Datenbasis
- *userID*: identifiziert den Benutzer
- *passedAttributes*: Attribute, die im Session-, bzw. dem Request-Objekt gespeichert sind. Für die Analyse relevant ist hier der Abfrageausdruck, der vom Benutzer eingegeben wurde, und der in einem Attribut mit der Bezeichnung „submission“ übergeben wird.
- *passedParameters*: Parameter, die im Request-Objekt gespeichert sind, wobei diese für die Analyse im XQuery- und im Datalog irrelevant sind.

2.1.3. Ausgabedaten

Die Ergebnisse der Analyse werden in einem Objekt zusammengefasst, d.h. hier werden alle wichtigen Informationen über die beiden verglichenen Abfragen, die Datenbasis, die verwendeten Parameter und die gefundenen Fehler gespeichert. Die entsprechende Klasse implementiert das eTutor-Interface *etutor.core.evaluation.Analysis*.

2.1.4. Systemfehler

Fehler müssen behandelt werden wenn die Musterlösung oder die Abgabe in irgendeiner Form nicht ausführbar oder ungültig ist, wobei dies auf einen der folgenden Systemfehler zutrifft:

- Die Parameter, die einer eindeutigen Aufgabe zugeordnet sind, wie etwa Dateinamen von Dateien mit Musterlösungen und Datenbasis, können nicht ermittelt werden.
- Die Parameter haben ungültige Werte, wie etwa Referenzen auf nicht vorhandene Dateien.
- Die Datenbasis oder die Musterlösung selbst weisen syntaktische Fehler auf und können nicht angewendet werden.
- Bei der Ausführung der Abfrage einer Musterlösung tritt ein sonstiger unerwarteter Fehler auf, der die Analyse in weiterer Folge unmöglich macht.
- Bei der Ausführung einer vom Benutzer übermittelten Abfrage tritt ein unerwarteter Analysefehler auf.

Tritt einer dieser Systemfehler ein, muss die Analyse gestoppt werden und eine Fehlermeldung an das aufrufende Programm übermittelt werden.

2.2. Bewertung

Die Ergebnisse einer Analyse werden herangezogen, um die gefundenen Fehler anhand definierter Kriterien zu bewerten. Die Funktionalität wird über die folgende Methode aufgerufen:

grade(Analysis analysis, int taskID, Map passedAttributes, Map passedParameters)

2.2.1. Funktionalität

Die Analyseergebnisse, die in Form eines Analyseobjektes vorliegen, werden ausgewertet und die identifizierten Fehler nach Kriterien bewertet, die durch die Art des Fehlers und das Bewertungsgewicht bestimmt werden.

Jedes Modul muss Fehlerkategorien definieren, denen die analysierten Fehler zugeordnet werden. Die Bewertungskriterien können somit für jede Fehlerkategorie, die das Modul bei der Analyse und der Bewertung berücksichtigt, einzeln definiert werden.

Zu diesen Kriterien zählt die Punkteanzahl die für einen gefundenen Fehler abgezogen wird. Außerdem soll durch einen Parameter angegeben werden können, wie oft Fehler einer bestimmten Fehlerkategorie bei der Bewertung berücksichtigt werden. Auf diese Weise soll unterschieden werden können, ob jeder einzelne Fehler einer Fehlerkategorie Punkteabzüge nach sich führt oder ob Punkte höchstens einmal pro Fehlerkategorie abgezogen werden.

Als Beispiel sei hier die Sortierung von XML-Knoten im Ergebnis einer XQuery-Abfrage genannt. Hier kann definiert werden, ob jeder falsch sortierte Knoten zu Punkteabzügen führt, oder ob Punkte generell abgezogen werden, unabhängig davon, ob nur einer oder mehrere Knoten falsch sortiert sind. Damit ist eine flexible Bewertung möglich.

2.2.2. Eingabedaten

Die Informationen werden aus den Eingabedaten ermittelt, die über die Methode *grade* übergeben werden:

- *analysis*: Die Bewertung wird anhand des Analyseobjektes durchgeführt (siehe Abschnitt 2.1.3). Für Übungsbeispiele kann zusätzlich festgelegt

werden, in welcher Höhe Fehler einer bestimmten Fehlerkategorie bewertet werden, und wie oft Fehler pro Fehlerkategorie zu Punkteabzügen führen (siehe Kapitel 5). Diese Informationen werden aus der *exerciseID* abgeleitet, die bereits bei der Analyse im Analyseobjekt gespeichert wird.

- *taskID*: Identifiziert die Übungsaufgabe, die es zu bewerten gilt.
- *passedAttributes*: Attribute, die im Session-, bzw. dem Request-Objekt gespeichert sind. Für die Bewertung relevant ist hier ein Attribut mit der Bezeichnung „action“, das den Übungsmodus kennzeichnet. Mögliche Werte sind „submit“ (wenn eine Bewertung erfolgen soll), sowie „diagnose“ und „run“ (wenn keine Bewertung erfolgen soll).
- *passedParameters*: Parameter, die im Request-Objekt gespeichert sind, wobei diese für die Bewertung irrelevant sind.

2.2.3. **Ausgabedaten**

Das Ergebnis der Bewertung ist ein Bewertungsobjekt, in dem alle wichtigen Informationen über die Bewertung gespeichert sind. Die entsprechende Klasse implementiert das eTutor-Interface *etutor.core.evaluation.Grading*.

2.2.4. Systemfehler

Fehler können auftreten wenn Parameter, die für die Bewertung benötigt werden, nicht ermittelt werden können, oder ungültige Werte haben.

In diesem Fall muss die Bewertung gestoppt werden und eine Fehlermeldung an das aufrufende Programm übermittelt werden.

2.3. Feedback

Die Feedback-Funktion ist dafür vorgesehen, die Analyse- und Bewertungsergebnisse für den Benutzer aufzubereiten. Hier werden einzelne Elemente generiert, aus denen sich das Dokument zusammensetzt, das dem Benutzer präsentiert wird (Auswertungsergebnis, textuelle Beschreibung der Fehler, Punkteabzüge, …). Die Funktionalität wird über die folgende Methode aufgerufen:

report(Analysis analysis, Grading grading, Map passedAttributes, Map passedParameters)

2.3.1. Funktionalität

Die vom jeweiligen Modul gelieferten Analyse- und Bewertungsobjekte werden herangezogen, um einzelne Elemente zu generieren, aus denen die Rückmeldung an den Benutzer zusammengesetzt werden kann.

Dies umfasst die folgenden Elemente: das Ergebnis der Abfrage oder eventuelle Meldungen über syntaktisch ungültige Abfragen, Informationen über die Analyse und gefundene Fehler, Hinweise auf mögliche Fehlerquellen, eventuelle Verbesserungsvorschläge und Informationen über die Bewertung der Abgabe;

Aus den gefundenen Fehlern soll eine Rückmeldung generiert werden, die Rückschlüsse auf die Fehlerursache zulässt. Dadurch soll Benutzern Hilfestellung geboten werden, potentielle Fehler im XQuery-Ausdruck selbst zu entdecken und zu korrigieren. Zusätzlich soll der Grad der Detailliertheit für die Rückmeldung definierbar sein.

Da der Benutzer das Ergebnis der abgegebenen Lösung in einem Internet-Browser betrachtet, ist es außerdem wichtig, dass die generierten Elemente für die Einbettung in eine HTML-Seite geeignet sind.

2.3.2. Eingabedaten

Die Informationen werden aus den Eingabedaten ermittelt, die über die Methode *report* übergeben werden:

- *analysis*: Die Generierung der Rückmeldung wird anhand des Analyseobjektes durchgeführt (siehe Abschnitt 2.1.3).
- *grading*: Neben dem Analyseobjekt wird auch das dazugehörige Bewertungsobjekt übergeben (siehe Abschnitt 2.2.3).
- *passedAttributes*: Attribute, die im Session-, bzw. dem Request-Objekt gespeichert sind. Relevant ist hier ein Attribut mit der Bezeichnung „diagnoseLevel“, mit dem das Ausmaß an Information über die Bewertungs- und Analyseergebnisse, das an den Benutzer zurückgeliefert werden soll, festgelegt wird.

- *passedParameters*: Parameter, die im Request-Objekt gespeichert sind, wobei diese für die Bewertung irrelevant sind.

2.3.3. Ausgabedateien

Das Ergebnis ist ein Objekt, in dem alle wichtigen Feedback-Elemente gespeichert sind, und aus denen sich das Dokument zusammensetzen lässt, das dem Benutzer präsentiert wird. Die entsprechende Klasse implementiert das eTutor-Interface *etutor.core.evaluation.Report*.

2.3.4. Systemfehler

Wenn bei der Generierung der Feedback-Elemente unerwartete und schwerwiegende Fehler auftreten muss eine Fehlermeldung an das aufrufende Programm übermittelt werden.

3. Lösungsansatz

In diesem Kapitel werden die Lösungsansätze der beiden Module beschrieben. Dabei gibt es in der Architektur, der Konfiguration und in Programmstrukturen wesentliche Parallelen. Zunächst wird der Lösungsansatz des Datalog-Moduls erklärt. Die Beschreibung unterteilt sich gemäß den in Kapitel 2 beschriebenen Anforderungen in die Abschnitte *Analyse*, *Bewertung* und *Feedback*. Analog dazu wird im darauffolgenden Abschnitt der Lösungsansatz des XQuery-Moduls erläutert.

Eine konkrete Übungsaufgabe setzt sich im Kern aus einer Datenbasis zusammen, sowie einem Abfrageausdruck, der die Musterlösung darstellt und in Bezug auf die Datenbasis ausgewertet wird. Die Auswertung der Abfrage liefert ein Ergebnis in Form strukturierter Daten. Diese müssen gegebenenfalls weiterverarbeitet und für die Analyse so aufbereitet werden, dass zwei Ergebnisse miteinander verglichen werden können.

Um eine Analyse durchzuführen, wird das entsprechende Modul mit einem zu analysierenden Abfrage-Ausdruck aufgerufen. Ergänzende Parameter dienen in erster Linie dazu, die Übungsaufgabe zu identifizieren, und somit die Datenbasis und die Musterlösung zu ermitteln, die als Referenz für die Auswertung der Abfrage herangezogen werden muss.

3.1. Datalog-Modul

Eine Übungsaufgabe im Datalog-Modul setzt sich im allgemeinen aus Fakten und Regeln zusammen [Bihl04]. In Fakten werden konkrete Zusammenhänge zwischen Objekten definiert. Fakten repräsentieren demnach vorhandenes Wissen und sind die Ausprägung von *Relationen*. Die Objekte, deren Beziehungen zueinander in Fakten abgebildet werden, tragen die Bezeichnung *Term*. Diese Begrifflichkeiten werden beispielhaft in Abbildung 3.1 verdeutlicht, wo Ausprägungen der Relationen *person* und *kind* definiert werden. In der Relation *person* werden die Terme als Namen von Personen aufgefasst und in der Relation

kind werden Verwandtschaftsverhältnisse zwischen diesen Personen dargestellt, bei denen eine Person das Kind einer weiteren Person ist.

```
person(maria).
person(michael).
person(max).
person(berta).

kind(maria, michael).
kind(maria, max).
```

Abbildung 3.1: Datalog-Fakten

Auf der Basis einer Anzahl von Datalog-Fakten, wie sie hier gezeigt werden, können nun Regeln ausgeführt werden, um aus dem vorhandenen Wissen neues Wissen abzuleiten. Das Ergebnis sind wiederum Datalog-Fakten. In Abbildung 3.2 etwa wird eine neue Relation *verwandt* auf der Basis aller bekannten *kind*-Relationen bestimmt. Die Variablen *X*, *Y* und *Z* sind dabei Platzhalter für alle möglichen Ausprägungen der jeweiligen Regel.

```
vorfahre(X,Y) :- kind(X,Y).
vorfahre(X,Y) :- vorfahre(X,Z), vorfahre(Z,Y).

verwandt(X,Y) :- vorfahre(X,Y).
verwandt(X,Y) :- vorfahre(Y,X).
verwandt(X,Y) :- verwandt(X,Z), verwandt(Y,Z), X != Y.
```

Abbildung 3.2: Datalog-Regeln

Wenn die hier gezeigten Regeln und die beschriebenen Fakten die Musterlösung darstellen, so wird das Ergebnis der Auswertung (Abbildung 3.3) als Basis für die Analyse einer weiteren Anzahl von Datalog-Regeln herangezogen.

```
vorfahre(maria, michael).
vorfahre(maria, max).
verwandt(maria, michael).
verwandt(maria, max).
verwandt(michael, maria).
verwandt(michael, max).
verwandt(max, maria).
verwandt(max, michael).
```

Abbildung 3.3: Ableitung neuer Relationen durch Datalog-Regeln

In den folgenden Abschnitten wird nun die Funktionsweise des Datalog-Moduls in den Bereichen der Analyse, der Bewertung von Datalog-Regeln und der Generierung des Feedbacks beschrieben.

3.1.1. Analyse

Die Analyse von Datalog-Regeln wird durch Klassen im Package *etutor.modules.datalog.analysis* realisiert. In Abbildung 3.4 wird der Vorgang dargestellt, der die Auswertung von Datalog-Regeln durch einen bestehenden Datalog-Prozessor und die Aufbereitung des Ergebnisses für die Analyse betrifft. Diejenigen Aktionen, die durch den Datalog-Prozessor ausgeführt werden, sind in einer eigenen Spalte gekennzeichnet. Die abgebildeten Schritte werden vorerst für die Musterlösung durchgeführt und daraufhin für die Abgabe:

1. *Musterlösung*: Die Musterlösung wird ausgewertet und darauf überprüft, ob die Aufgabenstellung selbst keine Fehler enthält. Eine fehlerhafte Aufgabenstellung kann etwa durch syntaktisch inkorrekte Datalog-Regeln oder –Fakten bedingt sein, aber auch durch ein an sich gültiges Ergebnis, das jedoch nicht im Sinne einer Analyse verarbeitet werden kann. Letztendlich können Fehler auch durch ungültige Auswertungsparameter ausgelöst werden. Tritt einer dieser Fehler auf, so wird die Analyse an diesem Punkt abgebrochen und eine Exception ausgelöst, die darauf hinweist, dass die Aufgabenstellung inkonsistent ist. Die Musterlösung setzt sich grundsätzlich aus einer Anzahl von Datalog-Regeln zusammen, die auf Datalog-Fakten ausgewertet werden. Die Fakten stellen die Datenbasis dar. Das Ergebnis der Auswertung wird für die weitere Analyse und den Vergleich mit der Abgabe des Benutzers in eine interne Repräsentation transformiert – ein Objekt der Klasse *etutor.modules.datalog.analysis.DatalogResult*. Im Diagramm wird dieses Ergebnis als *Datalog result* bezeichnet.
2. *Abgabe*: Im nächsten Schritt werden nach dem gleichen Verfahren die Datalog-Regeln, die vom Benutzer abgegeben werden, ausgewertet. Als Datenbasis werden exakt dieselben Datalog-Fakten herangezogen wie bei der Auswertung der Musterlösung. Während syntaktische Fehler bei der Auswertung der Musterlösung zum Abbruch führen, werden diese hier in die Analyse miteinbezogen.

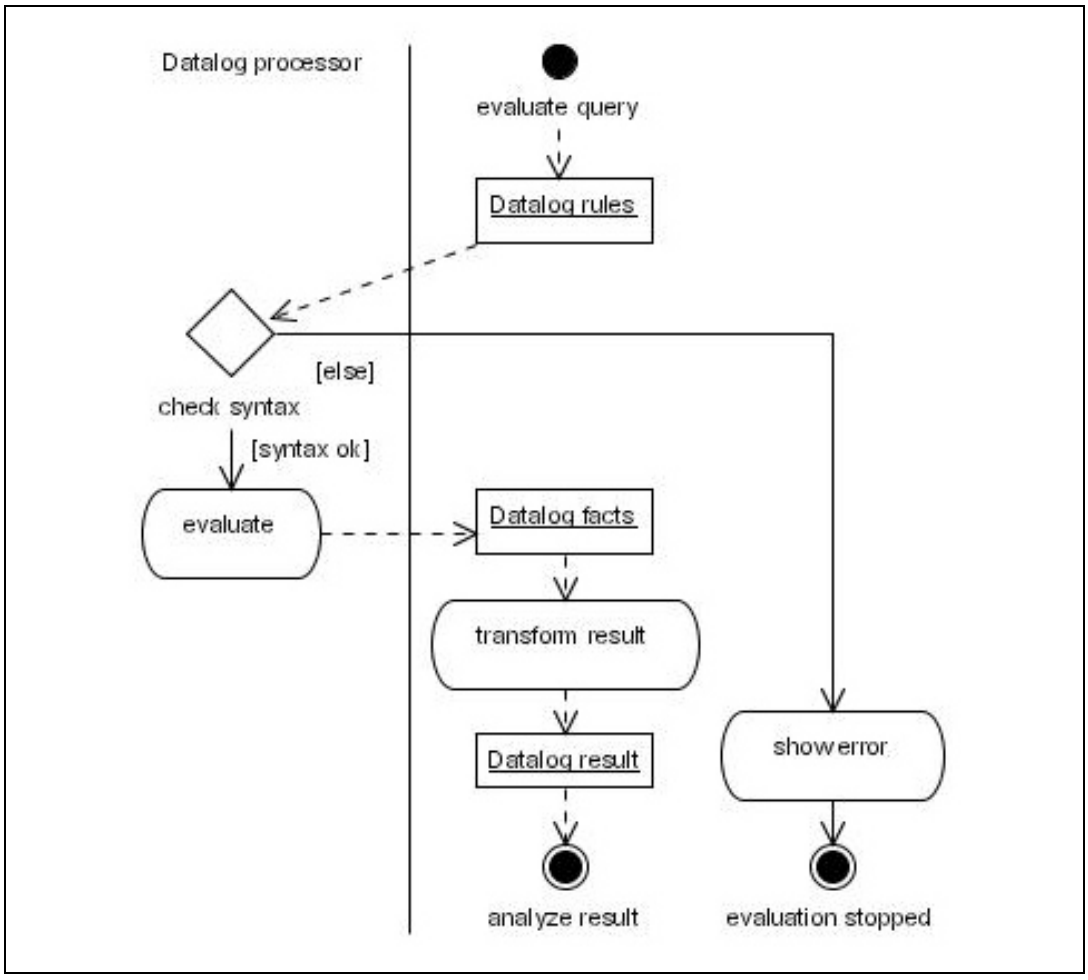

Abbildung 3.4: Auswertung von Datalog-Regeln

Nach erfolgter Auswertung der Musterlösung und der abgegebenen Lösung, bzw. nach Transformation der Ergebnisse, werden diese im nächsten Schritt verglichen und analysiert. Das Diagramm in Abbildung 3.5 stellt diesen Vorgang dar, wobei das Ergebnis der Musterlösung als *Datalog result 1* bezeichnet wird und das Ergebnis der abgegebenen Lösung als *Datalog result 2*. Das Analyseobjekt *analysis result* beinhaltet die Zuordnung von identifizierten Fehlern zu Fehlerkategorien und ist im Datalog-Modul ein Objekt der Klasse *etutor.modules.datalog.analysis.DatalogAnalysis*. Der Vorgang, der an die Analyse anknüpft – die Aufbereitung der Ergebnisse für den Benutzer – wird ebenfalls in diesem Diagramm dargestellt. Die Beschreibung dazu erfolgt in Abschnitt 3.1.3.

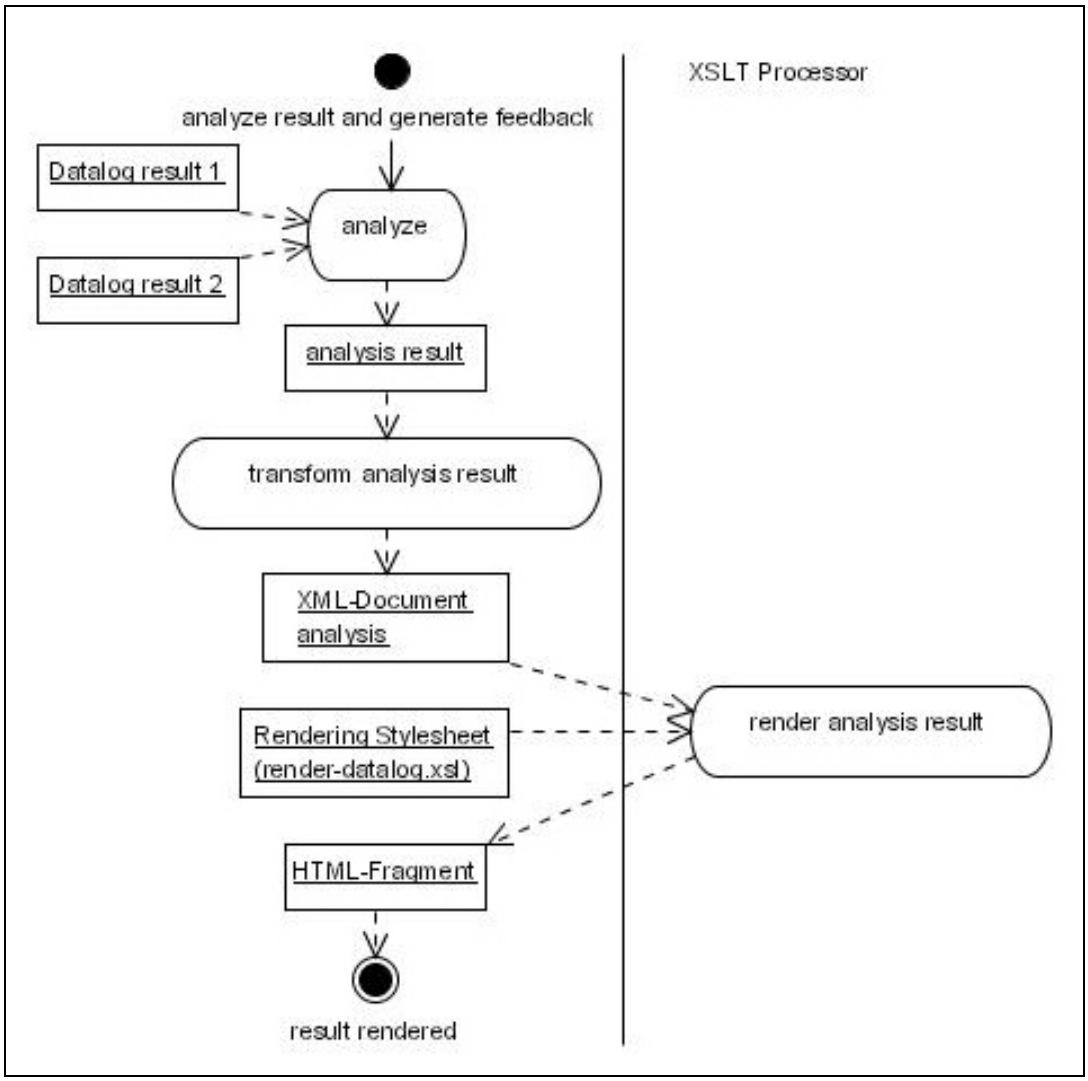

Abbildung 3.5: Analyse und Feedback-Generierung im Datalog-Modul

Einer der Parameter, die für die Definition einer Übungsaufgabe angegeben werden müssen, ist die Angabe derjenigen Fakten, die in der Abgabe eines Benutzers auf jeden Fall enthalten sein müssen. Dem Benutzer steht es prinzipiell offen, darüber hinaus beliebige Fakten zu definieren, die etwa als Zwischenergebnis für die Definition der verlangten Fakten dienen. So ist etwa die Relation *vorfahre* aus Abbildung 3.2 womöglich nicht verlangt, dient aber unter Umständen als Hilfskonstrukt, um letztendlich die Relation *verwandt* zu ermitteln.

Die folgenden Fehlerkategorien werden im Auswertungsergebnis von Datalog-Regeln analysiert, wobei sich die angeführten Beispiele auf Abbildung 3.3 beziehen:

- *Prädikate, bzw. Relationen, die in der Lösung gänzlich fehlen*: Dies ist etwa der Fall, wenn beide Relationen *verwandt* und *vorfahre* verlangt sind,

jedoch tatsächlich nur eine oder gar keine dieser Relationen im Ergebnis vorhanden sind.

- *Prädikate, die in der Lösung zwar enthalten sind, aber zu wenige Terme haben*: Dies ist etwa der Fall, wenn weniger als zwei Fakten in der Relation *verwandt* enthalten sind.
- *Prädikate, die in der Lösung zwar enthalten sind, aber zu viele Terme haben*: Dieser Fall stellt das Gegenteil des Fehlers dar, der im vorigen Punkt beschrieben wird.
- *Ausprägungen, die innerhalb eines Prädikats der Lösung fehlen:* Auf dieser Ebene wurde bereits festgestellt, dass das Prädikat, bzw. die Relation in der Lösung wie verlangt enthalten ist, wobei jedoch bestimmte Ausprägungen fehlen.
- *Ausprägungen, die innerhalb eines Prädikats der Lösung überflüssig sind*: Dieser Fall stellt das Gegenteil des Fehlers dar, der im vorigen Punkt beschrieben wird.
- *Fakten, die fälschlicherweise verneint sind:* Negierte Fakten werden durch Fakten dargestellt, denen ein ´-´ oder ein ´~´ Zeichen vorangestellt wird [Bihl04]. Unterscheidet sich ein Fakt der Musterlösung nur in diesem Punkt von einem Fakt der potentiellen Lösung, so wird dieser Fehler angezeigt.
- *Fakten, die fälschlicherweise nicht verneint sind*: Dieser Fall stellt das Gegenteil des Fehlers dar, der im vorigen Punkt beschrieben wird.

3.1.2. Bewertung

Die Bewertung von Analyseergebnissen erfolgt durch die Klassen im Java-Package *etutor.modules.datalog.grading*. Hier werden für die Fehler, die im Ergebnis der Auswertung von Datalog-Regeln gefunden wurden, Punkteabzüge vorgenommen, die in Datenbanktabellen für jede Fehlerkategorie definiert werden können (siehe Abschnitt 5.1.1 bis 5.1.3). Diese Informationen werden ausgelesen und dienen zur Konfiguration eines Objektes der Klasse *etutor.modules.datalog.grading.DatalogScores*. Dieses wird wiederum gemeinsam mit einem Objekt vom Typ *etutor.modules.datalog.analysis.DatalogAnalysis*, das das Ergebnis einer Analyse darstellt und alle Informationen über eventuell identifizierte Fehler enthält, dazu verwendet, um ein Objekt vom Typ *etutor.modules.datalog.grading.DatalogGrading* zu initialisieren. In diesem

Objekt werden die gefundenen Fehler anhand der Konfigurationsparameter bewertet.

Während Fehler der in Abschnitt 3.1.1 beschriebenen Fehlerkategorien je nach Aufgabendefinition gegebenenfalls einzeln bewertet werden, d.h. zu Punkteabzügen führen, werden in den folgenden Fehlerfällen von vornherein alle möglichen Punkte abgezogen:

- Die Regeln sind syntaktisch nicht korrekt formuliert.
- Die Regeln konnten innerhalb einer vorgegebenen Zeitspanne nicht ausgewertet werden (siehe Tabelle 6.1).
- Die Regeln sind so formuliert, dass das Ergebnis kein gültiges Modell enthält. Dies ist vereinfacht ausgedrückt die Folge von inkonsistenten oder widersprüchlichen Regeln, wie etwa in einer Regel *a :- not a*. Dieser Fall muss von Regeln unterschieden werden, die zwar keine neuen Fakten liefern, aber nichtsdestotrotz ein gültiges, wenn auch leeres Modell [Koch04]. Dies ist nur ein Spezialfall, der bei der Auswertung der Regeln auftreten kann: Der Datalog-Prozessor leitet anhand der definierten Regeln wiederholt neue Fakten aus den gegebenen und den schrittweise neu hinzugekommenen Fakten ab, bis sich am Ergebnis nichts mehr ändert. Dies wird auch als minimaler Fixpunkt bezeichnet.
- Die Regeln sind nicht *stratifizierbar*, d.h. eine eindeutige Reihenfolge für die Auswertung der Regeln kann nicht festgelegt werden. Das Verfahren, mit dem der Datalog-Prozessor nicht stratifizierbare Regeln auswertet, entspricht dem *Stable Model*-Ansatz [Koch04]. Als Ergebnis wird hier eine Anzahl mehrerer möglicher Modelle geliefert, die aus den verschiedenen Möglichkeiten resultieren, Regeln in einer bestimmten Reihenfolge auszuwerten [Loye04].

Anhand eines Beispiels in Abbildung 3.6 wird der letzte Fall verdeutlicht. Hier werden ausgehend von einer Relation *person*, für die ein Fakt vorliegt, zwei Regeln formuliert, die einerseits die Relation *ledig* und andererseits die Relation *verheiratet* erzeugen sollen. Allerdings enthält die Relation *ledig* im Regelrumpf eine negative Referenz auf die Relation *verheiratet*, die wiederum umgekehrt auf die Relation *ledig* verweist. Hier ist somit eine eindeutige Aussage nicht möglich, welche Regel zuerst ausgewertet werden muss, bevor die andere Regel auf der Basis des Zwischenergebnisses ausgewertet werden kann. Das bedeutet, dass diese Regeln nicht stratifizierbar sind. Der im Datalog-Modul verwendete

Datalog-Prozessor liefert in diesem Fall die beiden möglichen Modelle, die in der Abbildung mit (a) und (b) gekennzeichnet werden.

Nicht stratifizierbare Regeln müssen allerdings nicht unbedingt dazu führen, dass sich die Lösung aus mehr als einem Modell zusammensetzt. Ebenso wie bei inkonsistenten Regeln ist es auch möglich, dass kein einziges Modell gültig ist. In jedem Fall wird der Benutzer darauf hingewiesen, dass ein Fehler in der Formulierung der Regeln vorliegt, der mit der Stratifizierung zusammenhängen könnte. In Abbildung 3.7 wird ein Beispiel für nicht stratifizierbare Regeln gezeigt, bei dem der eingesetzte Datalog-Prozessor im Gegensatz zum vorigen Beispiel kein einziges konsistentes Modell liefert. Der Unterschied liegt hier lediglich darin, dass eine der negativen Kanten im Zyklus entfernt wurde.

```
person(max).

ledig(X) :- not verheiratet(X), person(X).
verheiratet(X) :- not ledig(X), person(X).
```

verheiratet(max).	ledig(max).
(a)	**(b)**

Abbildung 3.6: Nicht stratifizierbare Regeln, kein eindeutiges Modell

```
person(max).

ledig(X) :- verheiratet(X), person(X).
verheiratet(X) :- not ledig(X), person(X).
```

Abbildung 3.7: Nicht stratifizierbare Regeln, inkonsistentes Modell

3.1.3. Feedback

Für die Präsentation von Analyse- und Bewertungsergebnissen werden die Klassen im Java-Package *etutor.modules.datalog.report* eingesetzt. Die zentrale Klasse ist hier *etutor.modules.datalog.report.DatalogReport*, in der Methoden definiert sind, aus deren Ergebnissen sich die aufbereiteten Analyse- und Bewertungsergebnisse zusammensetzen. Das Ausmaß gewünschter Diagnosetiefe wird über einen Parameter bestimmt. Je höher dieses Diagnose-Level innerhalb des zulässigen Wertebereiches gesetzt wird, desto mehr Informationen über die Analyseergebnisse werden an den Benutzer zurückgeliefert. Auf der niedrigsten Stufe etwa wird nur das Auswertungsergebnis angezeigt, bzw. gegebenenfalls

Fehlermeldungen bei syntaktisch inkorrekten Datalog-Regeln. Die wichtigsten Methoden werden in Tabelle 3.1 zusammengefasst.

Methode	**Beschreibung**
getRenderedResult	Das Ergebnis, das vom Prozessor geliefert wurde, sofern die Musterlösung gültig und die Abgabe syntaktisch korrekt ist und ausgewertet werden konnte; In dieser Form wird das Ergebnis HTML-konform formatiert. Auf höchstem Diagnose-Level werden darüber hinaus alle Fehler direkt im Ergebnis angezeigt und als solche gekennzeichnet, was zum Beispiel auch fehlende Prädikate und Fakten umfasst, die an sich nicht im Ergebnis enthalten sind. Abbildung 3.8 zeigt den Unterschied zwischen Auswertungen auf hoher und niedriger Stufe. Im ersten Fall wird ein Auswertungsergebnis angezeigt, in dem gekennzeichnet ist, dass die Relation *verwandt* fehlt. Auf niedrigerem Diagnose-Level werden hingegen nur die Fakten angezeigt, die bei der Auswertung der Abfrage tatsächlich geliefert werden.
getRawResult	Im Gegensatz zur Methode *getRenderedResult* wird hier das Ergebnis ohne Formatierungen und ohne Kennzeichnung von fehlerhaften Stellen geliefert. Für Hinweise auf Fehler ist man hier somit auf textuelle Beschreibungen angewiesen.
getSyntaxErrors	Fehlermeldungen des Datalog-Prozessors zu syntaktischen Fehlern im Datalog-Ausdruck
getAnalysis	Eine textuelle Beschreibung der Analyseergebnisse, die unter anderem Hinweise auf Fehler im Auswertungsergebnis der Abgabe enthält
getGrading	Das Bewertungsergebnis der identifizierten Fehler, das über die maximal erreichbare Punkteanzahl und die tatsächlich erreichte Punkteanzahl informiert;
getGeneralAnalysis	Zusätzliche Meldungen an den Benutzer, die das Analyseergebnis allgemein in Worte fassen

Tabelle 3.1: Feedback-Bestandteile im Datalog-Modul

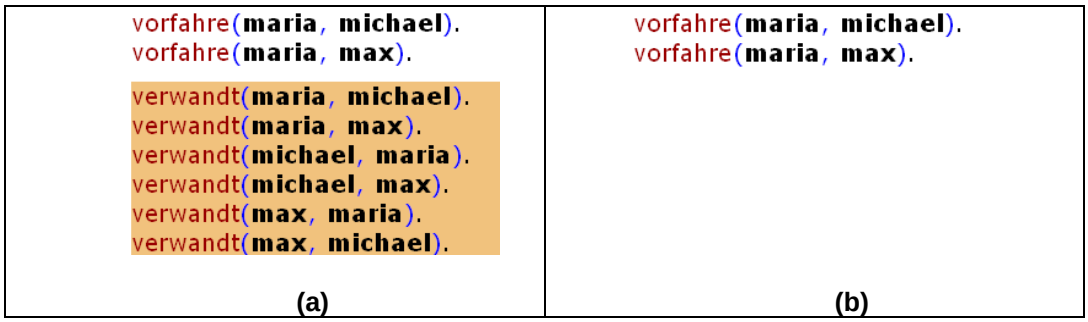

Abbildung 3.8: Datalog-Feedback auf hoher (a) und niedriger (b) Stufe

Sofern Datalog-Regeln syntaktisch korrekt sind, liefert der Datalog-Prozessor ein Ergebnis, das für den Benutzer aufbereitet und angezeigt werden muss. Dieses Ergebnis fließt in das Analyseobjekt ein, das im Diagramm in Abbildung 3.5 als *analysis result* bezeichnet wird und im Datalog-Modul durch ein Objekt der Klasse *etutor.modules.datalog.analysis.DatalogAnalysis* repräsentiert wird. In diesem Diagramm ist der Analyseprozess schematisch dargestellt, der in Abschnitt 3.1.1 erläutert wird. Darüber hinaus wird anhand des Diagramms ersichtlich, wie die Generierung des Feedbacks an die Analyse anknüpft. Da das Ergebnis formatiert dargestellt werden soll und als Mittel für die Aufbereitung der Analyseergebnisse XSL Transformationen gewählt wurde, werden alle Informationen, die im Analyseobjekt gespeichert sind, und die für die Darstellung relevant sind, in ein XML-Dokument transformiert (*XML-Document analysis*). Dieses Dokument wird nun mithilfe eines XSLT-Prozessors und eines XSL-Stylesheets in ein HTML-Fragment transformiert, das in die Seite eingebettet werden kann, die dem Benutzer präsentiert wird.

Die XML-Dokumente, in denen die Ergebnisse von Analysen zusammengefasst werden, entsprechen einem XML Schema, das in der Datei *datalog-result.xsd* definiert wird (siehe Abbildung 4.4). Die Elemente dieses Schemas werden anhand eines Modells in Abbildung 3.9 veranschaulicht und erfüllen die folgenden Zwecke:

- *datalog-result*: Wurzelelement, das XML-Attribute zu allgemeinen Informationen der Analyse enthält, wie die Aufgabennummer, erreichte Punkteanzahl und maximal erreichbare Punkteanzahl;
- *analysis*: Enthält Elemente für textuelle Beschreibungen des Analyseergebnisses;
- *summary*: Enthält eine Zusammenfassung der Analyseergebnisse;
- *grading*: Enthält eine Zusammenfassung der erreichten Punkteanzahl;

- *syntax*: Enthält eine Zusammenfassung eines eventuellen Syntaxfehlers in der analysierten Datalog-Regeln;
- *consistency*: Enthält eine Zusammenfassung eines eventuellen Fehlers in Bezug auf ein nicht konsistentes Modell im Auswertungsergebnis
- *error*: In diesem Element werden die analysierten Fehler auf detaillierterer Ebene beschrieben.
- *query*: Hier werden die analysierten Datalog-Regeln gespeichert.
- *model*: Ein, bzw. im Falle nicht stratifizierbarer Regeln gegebenenfalls mehrere Modelle, die das Ergebnis der ausgewerteten Regeln darstellen; In einem Attribut wird gespeichert, ob das jeweilige Modell konsistent ist.
- *predicate*: Prädikate, bzw. Relationen, aus denen sich ein Modell zusammensetzt; XML-Attribute enthalten Informationen über den Namen der Relation, die Anzahl der Terme, die ein Fakt dieser Relation hat, Informationen über einen eventuellen Fehler, der mit diesem Prädikat zusammenhängt, und Angaben darüber, ob das Prädikat in der Lösung vorhanden sein muss.
- *fact*: Fakten, aus denen ein Prädikat besteht; In XML-Attributen wird angegeben, ob das Fakt negiert ist, und welcher eventuelle Fehler mit diesem Fakt zusammenhängt.
- *term*: Terme, aus denen ein Fakt besteht, und deren Wert in einem XML-Attribut angegeben wird.

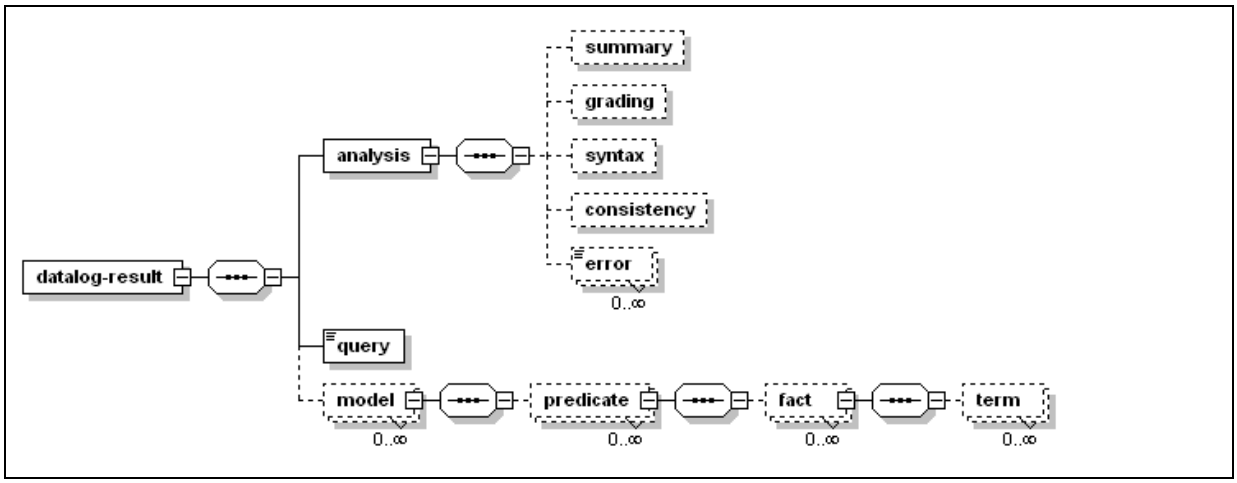

Abbildung 3.9: XML Schema für Analyseergebnisse im Datalog-Modul

3.2. XQuery-Modul

Eine Übungsaufgabe, die im XQuery-Modul definiert wird, setzt sich aus mindestens einem XML-Dokument zusammen, das die Datenbasis darstellt, sowie aus einem XQuery-Ausdruck – der Musterlösung. Abbildung 3.10 zeigt ein einfaches Beispiel eines XML-Dokumentes, das Informationen zu Personen enthält.

```
<db>
    <person nr="p1">
        <name>Maria</name>
        <stand>ledig</stand>
    </person>
    <person nr="p2">
        <name>Max</name>
        <stand>verheiratet</stand>
    </person>
    <person nr="p3">
        <name>Michael</name>
        <stand>ledig</stand>
    </person>
    <person nr="p4">
        <name>Berta</name>
        <stand>verwitwet</stand>
    </person>
</db>
```

Abbildung 3.10: XML-Dokument als Datenbasis für eine XQuery-Aufgabe

Ein ebenso einfaches Beispiel für einen XQuery-Ausdruck, der die Musterlösung darstellen könnte, wird in Abbildung 3.11 gezeigt. Ermittelt werden sollen alle Personen, deren Name mit dem Buchstaben *M* beginnt, wobei diese Personen nach ihrem Namen sortiert ausgegeben werden sollen. Über die *doc*-Funktion kann auf die XML-Dokumente zugegriffen werden, die als Datenbasis dienen. In diesem Beispiel wird also angenommen, dass das Dokument über die angegebene URL geladen werden kann.

```
let $db := doc('http://etutor.dke.uni-linz.ac.at/XML?id=1')/db,
        $personen := $db/person
return (
        for $p in $personen[starts-with(name,'M')]
        order by $p/name
        return $p)
```

Abbildung 3.11: XQuery-Ausdruck für eine XQuery-Aufgabe

Als Ergebnis dieser Abfrage werden drei XML-Knoten geliefert, die für die gewünschten Personen stehen. Das Ergebnis wird bei der Analyse immer als

XML-Fragment interpretiert, das jedoch selbst kein gültiges XML-Dokument ist, da es dazu in ein Wurzel-Element eingebettet sein müsste.

```
<person nr="p1">
    <name>Maria</name>
    <stand>ledig</stand>
</person>
<person nr="p2">
    <name>Max</name>
    <stand>verheiratet</stand>
</person>
<person nr="p3">
    <name>Michael</name>
    <stand>ledig</stand>
</person>
```

Abbildung 3.12: Auswertungsergebnis eines XQuery-Ausdruckes

In den folgenden Abschnitten wird nun die Funktionsweise des XQuery-Moduls in den Bereichen der Analyse, der Bewertung von Datalog-Regeln und der Generierung des Feedbacks beschrieben.

3.2.1. Analyse

Die Analyse im XQuery-Modul, die durch die Klassen im Package *etutor.modules.xquery.analysis* bewerkstelligt wird, setzt sich im allgemeinen bei jedem Aufruf der Analysefunktion des Moduls aus den folgenden Schritten zusammen:

1. Der XQuery-Ausdruck, der die Musterlösung darstellt, wird mit einem oder mehreren bestimmten XML-Dokumenten als Datenbasis ausgewertet. Das Ergebnis, das vom XQuery-Prozessor geliefert wird, muss so strukturiert werden, dass es mit einem weiteren Ergebnis verglichen werden kann. Aus diesem Grund wird aus dem Ergebnis ein wohlgeformtes XML-Dokument gebildet.
2. Der XQuery-Ausdruck, der analysiert werden soll, wird mit der selben Datenbasis wie die Musterlösung ausgewertet und das Ergebnis der Abfrage ebenso in ein wohlgeformtes XML-Dokument transformiert.
3. Die somit entstandenen XML-Dokumente werden verglichen.
4. Das Ergebnis, das durch den Vergleich geliefert wird, bildet die Grundlage für die tatsächliche Analyse. Aus diesem Ergebnis lassen sich die Unterschiede zwischen der abgegebenen Lösung und der Musterlösung erkennen und zu vordefinierten Fehlerkategorien zuordnen.

Die ersten beiden Schritte, in denen vorerst der korrekte und daraufhin der abgegebene XQuery-Ausdruck ausgewertet wird, erfolgen nach dem gleichen Schema und werden somit gleichermaßen durch das in Abbildung 3.14 gezeigte Diagramm skizziert. Beide Vorgänge werden mit einem XQuery-Ausdruck gestartet, der durch den XQuery-Prozessor ausgewertet werden soll. Die entsprechenden Aktivitäten, die diesen Prozessor betreffen, werden im Diagramm in einer eigenen Spalte gekennzeichnet. Wenn ein Ausdruck syntaktisch korrekt ist, liefert der XQuery-Prozessor ein Ergebnis, das wie im Beispiel in Abbildung 3.12 als XML-Fragment interpretiert wird. Um diese Ergebnisse vergleichbar und analysierbar zu machen, werden sie in ein standardmäßiges XML-Element eingebettet. Auf diese Weise wird ein XML-Dokument erzeugt, das für den Vergleich zweier Lösungen geeignet ist. Die Auswertung wird in zwei grundsätzlichen Fällen abgebrochen:

- Ein XQuery-Ausdruck ist syntaktisch nicht korrekt und kann daher nicht ausgewertet werden. Der XQuery-Prozessor liefert in diesem Fall eine Fehlermeldung zurück, die Hinweise auf die Fehlerquelle im Ausdruck enthält.
- Der XQuery-Ausdruck ist zwar syntaktisch korrekt, die Auswertung wird aber durch das XQuery-Modul verhindert, da er über *doc*-Funktionen Zugriffe auf XML-Dokumente enthält, die nicht für die Aufgabenstellung definiert sind. Dies hängt damit zusammen, dass die relevanten XML-Dokumente für jede Aufgabe vordefiniert werden. Im Falle der Musterlösung werden alle realen Pfade in den *doc*-Funktionen im XQuery-Ausdruck angegeben. Ein Fehler wird jedoch ausgelöst, wenn kein *Aliasname* für dieses Dokument definiert wurde, was etwa im Falle des Beispiels in Abbildung 3.10 die Kurzbezeichnung *personen.xml* sein könnte. Umgekehrt darf im XQuery-Ausdruck einer abgegebenen Lösung kein anderer als der definierte Aliasname in doc-Funktionen enthalten sein.
- Der XQuery-Ausdruck ist zwar syntaktisch korrekt und kann ausgewertet werden, das Ergebnis kann aber nicht als XML-Fragment interpretiert werden, da es nicht wohlgeformt ist. Zu beachten ist hier, dass dieses Ergebnis als gültige Auswertung durch den XQuery-Prozessor geliefert wird und demnach im eigentlichen Sinn nicht fehlerhaft ist. Der Fehler wird im Sinne der Analyse jedoch angezeigt, nachdem ein Vergleich mit einem weiteren Ergebnis nicht möglich ist. In Abbildung 3.13 wird ein Beispiel für einen XQuery-Ausdruck gezeigt, der zu einem Ergebnis führt,

in dem ein XML-Element zwei XML-Attribute mit derselben Bezeichnung *name* enthält.

```
let $db := doc(' http://etutor.dke.uni-
linz.ac.at/XML?id=1')/db,
        $personen := $db/person
return (
        for $p in $personen[starts-
with(name,'M')]
        order by $p/name
        return <person name='{$p/name}'
                name='M'/>)
```

(a)

```
<person name="Maria" name="M"/>
<person name="Max" name="M"/>
<person name="Michael" name="M"/>
```

(b)

Abbildung 3.13: Nicht wohlgeformtes XQuery-Ergebnis

Diese Ausnahmefälle haben eine unterschiedliche semantische Bedeutung, je nachdem ob die Musterlösung oder die abgegebene Lösung betroffen ist. Wenn die Analyse abgebrochen wird, weil bei der Auswertung der Musterlösung einer dieser Fehler aufgetreten ist, wird ein XQuery-Ausdruck eines Benutzers naheliegenderweise nicht mehr ausgewertet und er erhält allenfalls den Hinweis darauf, das die Musterlösung nicht konsistent ist und Auswertungen von XQuery-Ausdrücken, bzw. deren Analyse deswegen nicht möglich sind. Falls die Musterlösung korrekt ausgewertet werden kann, die abgegebene Lösung hingegen nicht, so enthält die Rückmeldung an den Benutzer den entsprechenden Hinweis auf die Art des Fehlers in dessen XQuery-Ausdruck. Dieser verhindert dementsprechend nicht, das der XQuery-Ausdruck und dessen Ergebnis analysiert und bewertet wird.

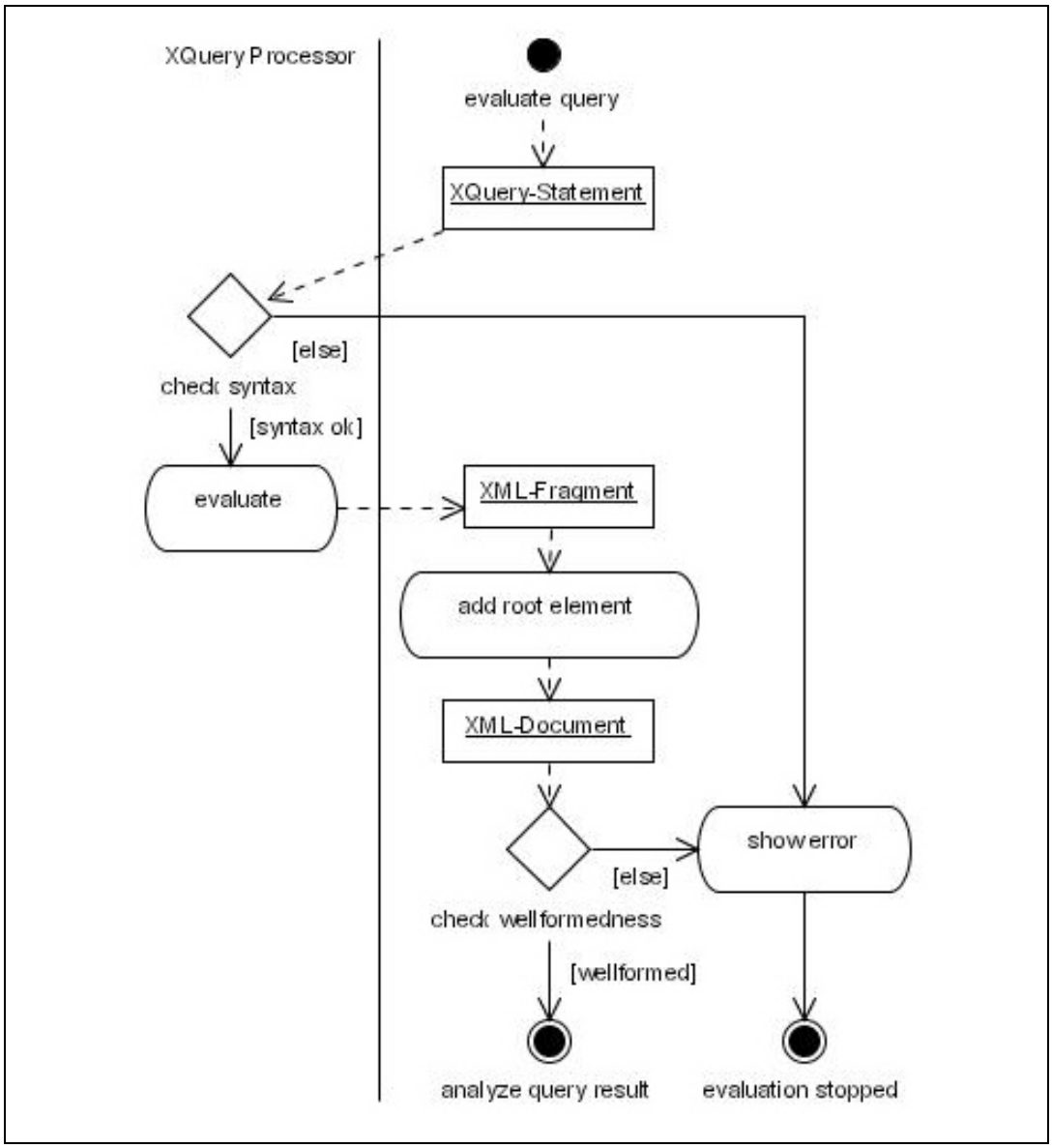

Abbildung 3.14: Auswertung von XQuery-Ausdrücken

Die entsprechende Java-Klasse im XQuery-Modul, die das Auswertungsergebnis eines XQuery-Ausdruckes definiert, ist die Klasse *etutor.modules.xquery.analysis.XQResult*. In einem Objekt dieser Klasse ist das XML-Dokument gespeichert, das im Diagramm in Abbildung 3.14 als *XML-Document* bezeichnet wird. Nachdem sowohl die Musterlösung als auch die abgegebene Lösung erfolgreich ausgewertet und in XML-Dokumente transformiert wurden, erfolgt im nächsten Schritt der Vergleich und die Analyse dieser Dokumente. Dieser Vorgang wird in einem Diagramm in Abbildung 3.15 dargestellt. Für den Vergleich wird eine Klassenbibliothek verwendet, mit der zwei XML-Dokumente verglichen werden können und die in Abschnitt 4.1.1 beschrieben wird. Im Diagramm ist die Vergleichsaktivität gekennzeichnet, die

durch dieses Tool erfolgt. Als Input für den Vergleich werden die XML-Dokumente herangezogen, die in den vorhergehenden Schritten für die Musterlösung und die abgegebene Lösung gemäß Abbildung 3.14 erzeugt wurden. In Abbildung 3.15 erhalten diese Dokumente die Bezeichnung *XML-Document 1* für die Musterlösung und *XML-Document 2* für die potentielle Lösung. Als Output wird ein XSL-Stylesheet generiert, das alle Unterschiede zwischen den beiden XML-Dokumenten repräsentiert. Dieses generierte Dokument dient als Grundlage für die tatsächliche Analyse. Das bedeutet, dass die Unterschiede, die in der potentiellen Lösung durch den Vergleich mit der Musterlösung identifiziert wurden, zu bestimmten Fehlerkategorien zugeordnet werden. Das Ergebnis in diesem Diagramm, das als *analysis result* bezeichnet wird, ist im XQuery-Modul ein Objekt der Java-Klasse *etutor.modules.xquery.analysis.XQAnalysis*.

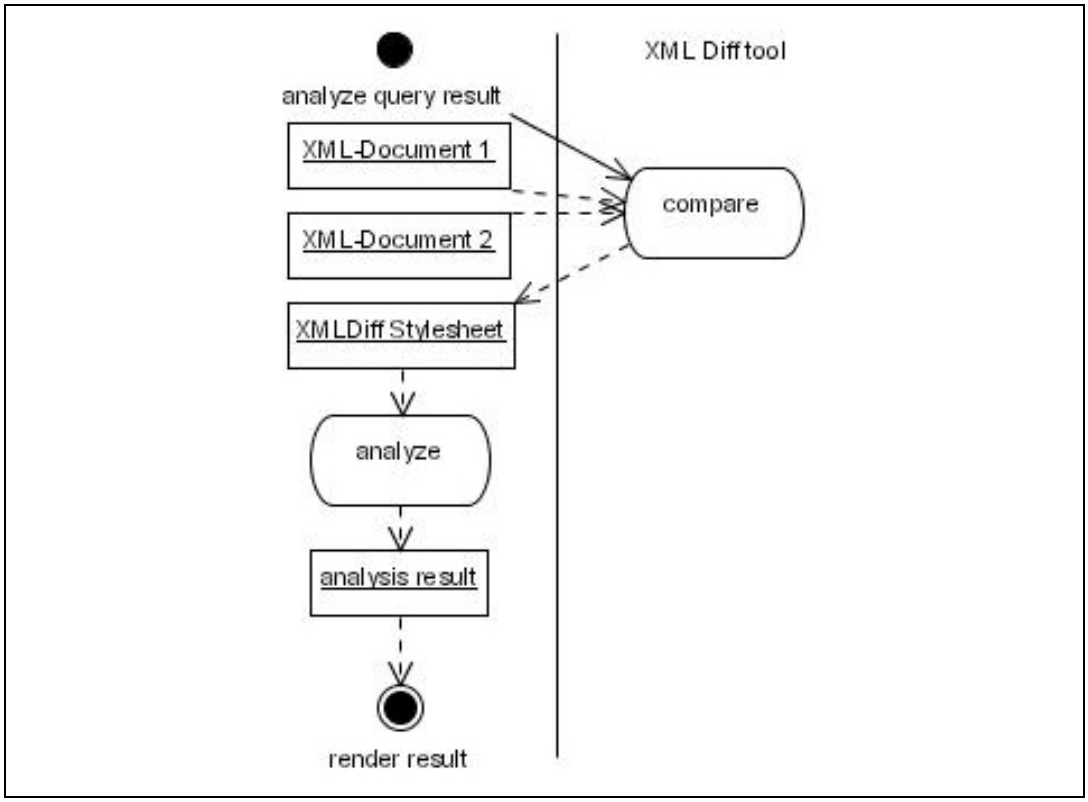

Abbildung 3.15: Analyse von XQuery-Ergebnissen

Die Analyse von XQuery-Ausdrücken wird durch die Angabe eines wesentlichen Parameters beeinflusst. Dabei handelt es sich um die Definition, bei welchen XML-Elementen im Ergebnis der Musterlösung auf die Sortierung Wert gelegt wird. Nachdem bei der Analyse der abgegebenen Lösung festgestellt wurde, dass bestimmte XML-Elemente korrekt sind, wird im letzten Schritt noch überprüft, ob bei diesen XML-Elementen eine bestimmte Sortierung gefordert ist, und ob diese mit der Sortierung in der Musterlösung übereinstimmt. Die korrekt zu

sortierenden XML-Elemente werden anhand von *XPath*-Ausdrucken definiert. Ein solcher Ausdruck wird auf das Ergebnis einer bestimmten Musterlösung, das wie beschrieben als XML-Dokument vorliegt, ausgewertet. Das Resultat sind je definiertem XPath-Ausdruck eine Anzahl von XML-Elementen. In Abbildung 3.16 wird mit (a) und (b) noch einmal das eingangs beschriebene Beispiel dargestellt, das anhand der Datenbasis in Abbildung 3.10 ausgewertet wird. Wenn sich nun bei der Analyse der abgegebenen Lösung (c) herausstellt, das ein XML-Element im Auswertungsergebnis (d) an sich mit dem entsprechenden XML-Element der Musterlösung übereinstimmt, so wird im nächsten Schritt untersucht, ob dieses XML-Element auch richtig sortiert ist, bzw. ob diese Sortierung überhaupt vorgeschrieben ist. Dazu werden alle definierten XPath-Ausdrücke ausgewertet und überprüft, ob sich das aktuelle XML-Element darunter befindet. Im gezeigten Beispiel ist das XML-Element mit der Nummer *p2* jedenfalls falsch sortiert. Ist nun der XPath-Ausdruck *//person* gegeben, so befindet sich dieses XML-Element unter den XML-Elementen, die durch die Auswertung dieses Ausdruckes geliefert werden, weshalb dieser Fehler tatsächlich bewertet wird.

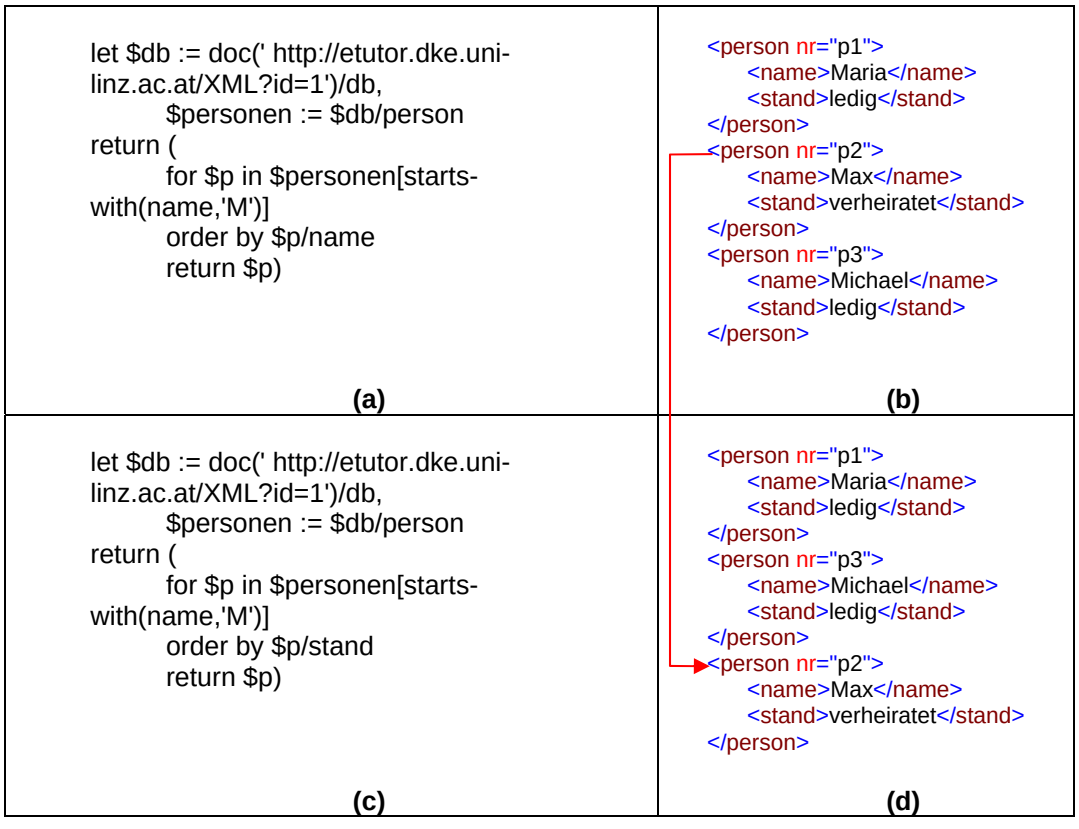

Abbildung 3.16: Analyse der Sortierung von XML-Elementen

In der folgenden Auflistung werden zusammenfassend alle Fehlerkategorien angeführt, auf deren Basis das Ergebnis eines XQuery-Ausdruckes analysiert werden.

- *Fehlende XML-Knoten*: Dies betrifft XML-Elemente und XML-Textknoten, die im Ergebnis der Musterlösung enthalten sind, aber im Ergebnis der abgegebenen Lösung in keiner Weise identifiziert werden konnten.
- *Überflüssige XML-Knoten*: Fehler dieser Fehlerkategorie stellen den umgekehrten Fall von fehlenden XML-Knoten dar.
- *Falsch sortierte XML-Elemente*: Von dieser Fehlerkategorie betroffen sind XML-Elemente, die zwar auf einer bestimmten Hierarchiestufe in der XML-Struktur korrekt vorhanden sind, allerdings nicht an der erwarteten Position. Dies wird nur als Fehler behandelt, wenn in der Aufgabenstellung definiert worden ist, dass der betreffende Knoten die selbe Sortierung aufweisen muss wie die entsprechenden Knoten in der Musterlösung.
- *Fehlende XML-Attribute*: Dieser Fehler betrifft XML-Elemente im Ergebnis einer abgegebenen Lösung, die als Pendant eines entsprechenden XML-Elements in der Musterlösung identifiziert wurden, die aber im Gegensatz zu diesem XML-Element ein bestimmtes XML-Attribut nicht enthalten.
- *Überflüssige XML-Attribute*: Fehler dieser Fehlerkategorie stellen den umgekehrten Fall von fehlenden XML-Attributen dar.
- *XML-Attribute mit falschen Werten*: In diesem Fall ist ein XML-Element identifiziert worden, das zwar ein entsprechendes XML-Element der Musterlösung darstellt, in dem aber ein gleichlautendes XML-Attribut einen anderen Wert hat.

3.2.2. Bewertung

Die Bewertung von Analyseergebnissen erfolgt durch die Klassen im Java-Package *etutor.modules.xquery.grading*. Hier werden für die Fehler, die in einem XQuery-Ausdruck gefunden wurden, Punkteabzüge vorgenommen, die für jede Fehlerkategorie in Datenbanktabellen definiert werden können (siehe Abschnitt 5.2.1 bis 5.2.3). Anhand dieser Tabellen wird ein Objekt *etutor.modules.xquery.grading.XQScores* konfiguriert, das dazu verwendet wird, um ein Objekt vom Typ *etutor.modules.xquery.grading.XQGrading* zu

initialisieren. Das zweite Objekt, das bei dieser Initialisierung benötigt wird, ist ein Objekt vom Typ *etutor.modules.xquery.analysis.XQAnalysis*, das das Ergebnis einer Analyse eines XQuery-Ausdruckes darstellt.

3.2.3. Feedback

Die Klassen, die für die Aufbereitung von Analyse- und Bewertungsergebnissen verwendet werden, befinden sich im Java-Package *etutor.modules.xquery.report*. Die Informationen, die der Benutzer über die Analyse und Bewertung eines XQuery-Ausdruckes erhält, werden von Methoden der Klasse *etutor.modules.xquery.report.XQReport* geliefert. Ebenso wie im Datalog-Modul wird das Ausmaß der Information über einen Parameter bestimmt, der Werte annehmen kann, die in Feldern der Klasse vordefiniert sind. Die wichtigsten Methoden werden in Tabelle 3.2 zusammengefasst.

Methode	**Beschreibung**
getRenderedResult	Das Ergebnis, das vom Prozessor geliefert wurde, sofern die Musterlösung gültig und die Abgabe syntaktisch korrekt ist und ausgewertet werden konnte; In dieser Form wird das Ergebnis HTML-konform formatiert, wodurch strukturelle Elemente im Ergebnis hervorgehoben werden. Auf höchstem Diagnose-Level werden außerdem alle Fehler direkt in diesem Ergebnis angezeigt und als solche gekennzeichnet, was zum Beispiel auch fehlende XML-Elemente oder –Attribute umfasst, die an sich nicht im Ergebnis enthalten sind.
getRawResult	Im Gegensatz zur Methode *getRenderedResult* wird hier das Ergebnis ohne Formatierungen und ohne Kennzeichnung von fehlerhaften Stellen geliefert. Für Hinweise auf Fehler ist man hier somit auf textuelle Beschreibungen angewiesen.
getSyntaxError *getWellformednessErrors*	Fehlermeldungen zu syntaktischen Fehlern im übermittelten XQuery-Ausdruck, zu Problemen mit der Wohlgeformtheit des Auswertungsergebnisses und zu ungültigen Referenzen auf XML-Dokumente in *doc*-Funktionen;

getUrlContentError	
getAnalysis	Informationen über die Fehler, die im Auswertungsergebnis der abgegebenen Lösung identifiziert wurden, und die den Unterschied zum Auswertungsergebnis der Musterlösung ausmachen;
getGrading	Das Bewertungsergebnis der identifizierten Fehler, das über die maximal erreichbare Punkteanzahl und die tatsächlich erreichte Punkteanzahl informiert,
getGeneralAnalysis	Zusätzliche Meldungen an den Benutzer, die das Analyseergebnis allgemein in Worte fassen

Tabelle 3.2: Feedback-Bestandteile im XQuery-Modul

Das Auswertungsergebnis eines XQuery-Ausdruckes kann in jedem Fall angezeigt werden, sofern der XQuery-Prozessor ein Ergebnis liefert. Dieses Ergebnis ist in Ausnahmefällen nicht als wohlgeformtes XML-Fragment interpretierbar. Für den Benutzer wird das Ergebnis in diesem Fall so angezeigt, wie es vom XQuery-Prozessor geliefert wurde, eine weitere Analyse ist jedoch nicht möglich. Wenn das Ergebnis als wohlgeformtes XML-Fragment interpretiert werden kann, wird aus dem XML-Fragment ein XML-Dokument generiert, das für die weitere Analyse verwendet wird. In Abschnitt 3.2.1 wurde beschrieben, dass die Unterschiede zwischen zwei analysierten XML-Dokumenten, die das Ergebnis der Musterlösung und der abgegebenen Lösung darstellen, aus einem XSL-Stylesheet ausgelesen werden können, das mithilfe eines Vergleichs-Tools erzeugt wird. Allerdings entspricht die Repräsentation nicht exakt den tatsächlichen Fehlern und der Art und Weise, wie das Ergebnis für den Benutzer dargestellt werden soll. Würde das XML-Dokument, das die potentielle Lösung darstellt, mit dem XSL-Stylesheet transformiert werden, das mithilfe des Vergleichs-Tools generiert wird, so würde exakt das XML-Dokument erzeugt, das die Musterlösung darstellt. Aus diesem Grund muss aus dem generierten XSL-Stylesheet ein adaptiertes XSL-Stylesheets erzeugt werden, das die folgenden grundsätzlichen Änderungen aufweist:

- Wenn die potentielle Lösung mit dem geänderten Stylesheet transformiert wird, soll nicht die Musterlösung erzeugt werden, vielmehr soll auch noch nach der Transformation das ursprüngliche Dokument zu erkennen sein.

- Die Elemente, die die Unterschiede zwischen der Musterlösung und der potentiellen Lösung ausmachen, sollen entsprechend gekennzeichnet werden. Um dies zu erreichen, müssen gewisse XSL-Templates des ursprünglichen Stylesheets adaptiert werden.

Die Vorgehensweise bei der Generierung des Feedbacks, die unter anderem diese Transformation umfasst, wird anhand eines Diagrammes in Abbildung 3.17 veranschaulicht. Das XSL-Stylesheet, das bei der Analyse generiert wurde (siehe Abbildung 3.15), wird mithilfe eines weiteren XSL-Stylesheets (*modify-xmldiff.xsl*) modifiziert. Das XML-Dokument, das im Diagramm als *XML-Document 2* bezeichnet wird, und das die Abgabe eines Benutzers darstellt, wird mit diesem nunmehr modifizierten XSL-Stylesheet transformiert. Das Ergebnis ist ein XML-Dokument, das letztendlich mithilfe eines weiteren XSL-Stylesheets (*render-xquery.xsl*) in ein HTML-Fragment transformiert wird, das in die Seite eingebettet werden kann, die dem Benutzer präsentiert wird.

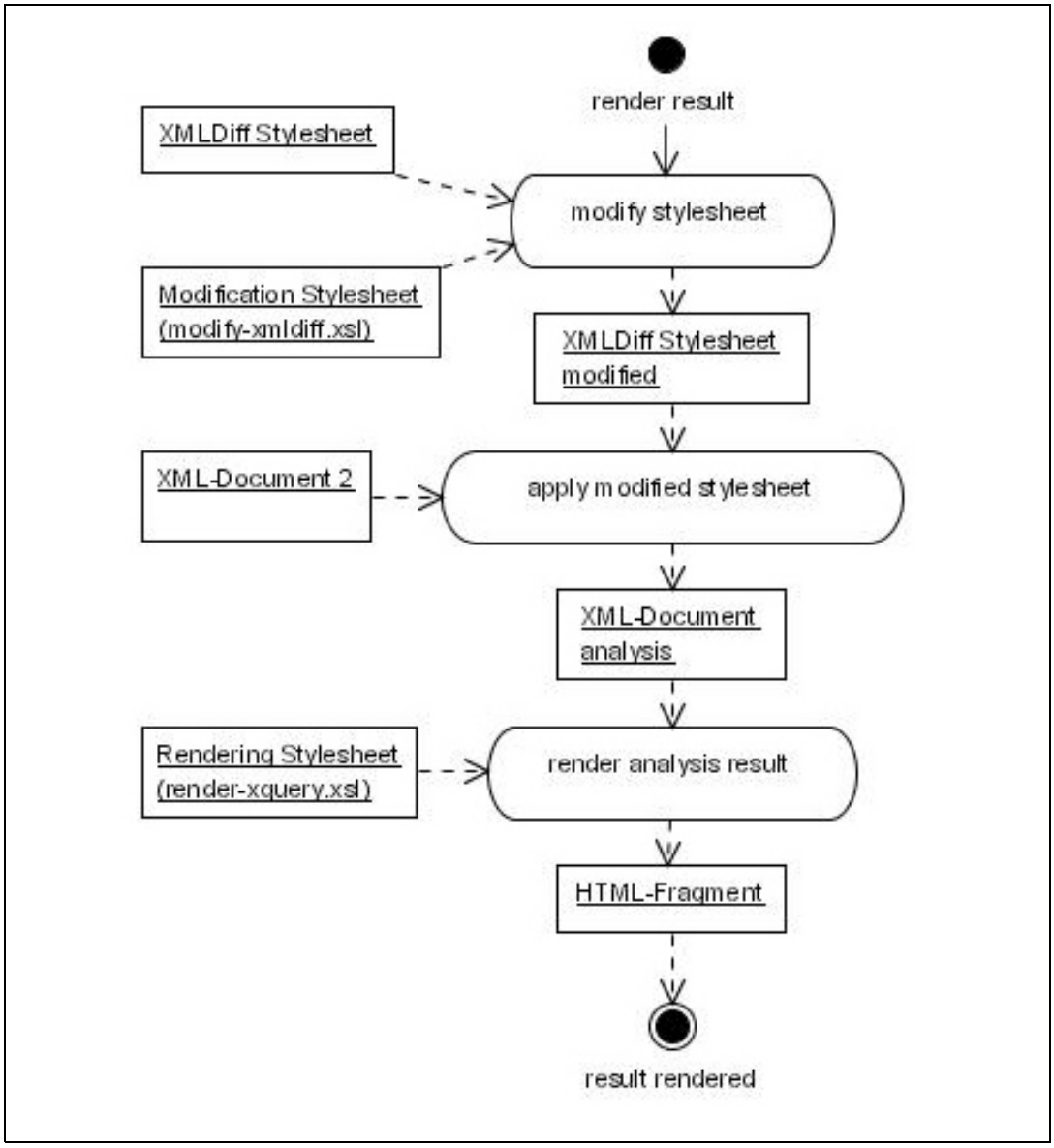

Abbildung 3.17: Feedback-Generierung im XQuery-Modul

Die XML-Dokumente, in denen die Ergebnisse von Analysen zusammengefasst werden, entsprechen einem XML Schema, das in der Datei *xquery-result.xsd* definiert wird (siehe Abbildung 4.5). Die Elemente dieses Schemas werden anhand eines Modells in Abbildung 3.18 veranschaulicht und erfüllen die folgenden Zwecke:

- *xquery-result*: Wurzelelement, in dem XML-Attribute für allgemeine Informationen zur Analyse enthält, wie die Aufgabennummer, erreichte Punkteanzahl und maximal erreichbare Punkteanzahl.
- *query*: Hier wird der analysierte XQuery-Ausdruck gespeichert.

- *result*: Dies ist das Standard-Element, in dem die Ergebnisse des XQuery-Ausdruckes eingebettet werden. Zusätzlich werden in XML-Attributen Informationen dazu gespeichert, ob das Ergebnis auf einem XQuery-Ausdruck basiert, der syntaktisch korrekt ist, und ob das Ergebnis als wohlgeformtes XML-Fragment interpretiert werden kann. Wenn der letzte Fall nicht zutrifft, wird das Ergebnis als Textknoten gespeichert.
- *analysis*: Enthält Elemente für die textuelle Beschreibung des Analyseergebnisses;

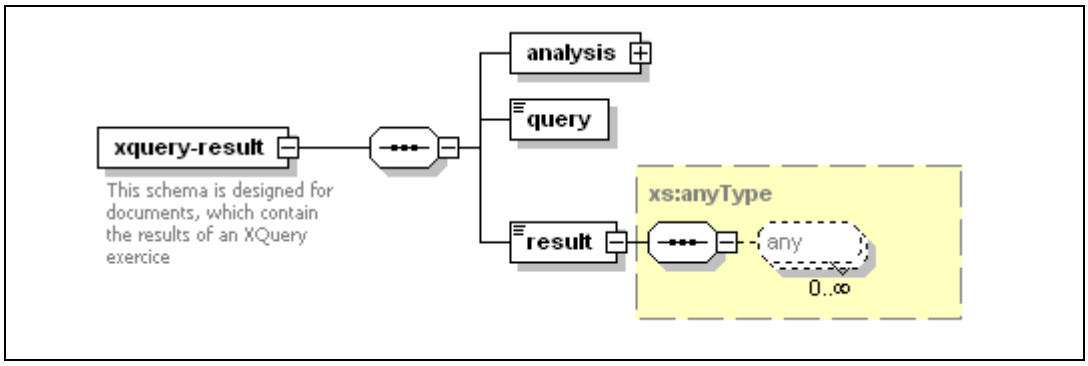

Abbildung 3.18: XML Schema für Analyseergebnisse im XQuery-Modul

4. Systemstruktur

In diesem Kapitel wird der Entwurf des Analyse- und Bewertungsmoduls beschrieben. Behandelt werden die Systemarchitektur der Module und der Aufbau der einzelnen Module in Form von Java-Packages.

4.1. Architektur

In diesem Abschnitt wird beschrieben, aus welchen Bestandteilen sich das System zusammensetzt, welche Aufgaben sie haben und wie sie zusammenwirken. Zunächst erfolgt eine detaillierte Beschreibung der Packages, der Abhängigkeiten zu Bibliotheken und der Beschreibung der einzelnen Funktionsblöcke.

Die Analyse- und Bewertungsmodule wurden für die Integration in das eTutor-System konzipiert und mit Version 1.4.2.04 der Java 2 Plattform API kompiliert. Um die Module im Sinne einer verteilten Architektur integrieren zu können, sind die Schnittstellen so implementiert, dass sie über Remote Method Invocation (RMI) aufgerufen werden können (siehe Abschnitt 4.1.4).

Die Analyse und Bewertung von Datalog-Regeln und XQuery-Ausdrücken und die Rückmeldung über die Ergebnisse wird durch eine Anzahl von externen Klassenbibliotheken unterstützt. Den Kern für die Auswertung von Datalog-Queries stellt hingegen eine ausführbare Datei dar (siehe Abschnitt 4.1.2).

Die Definitionen einzelner Datalog-Aufgaben werden in einer Datenbank gespeichert und dort mithilfe eines eindeutigen Schlüssels abgerufen.

Um die Konfiguration der Module, bzw. die Integration im eTutor-System zu veranschaulichen, wird hier auf die Modellierung in der Form von Verteilungsdiagrammen zurückgegriffen. Auch wenn es gemeinsame Aspekte gibt, so werden das Datalog-Modul (siehe Abbildung 4.1) und das XQuery-Modul (siehe Abbildung 4.2) in getrennten Diagrammen dargestellt. Die Systemkomponenten, die in beiden Diagrammen durch die Knoten *User: Client* und *E-Tutor Server: Server* repräsentiert werden, sind demzufolge als identisch zu

betrachten. Ersterer stellt den Rechner dar, von dem aus über eine Internet-Verbindung eine Anfrage an den zweiten Knoten, also den Web-Server auf dem das eTutor-System installiert ist, gesendet wird. Von diesem System wird nun über eine RMI-Verbindung das entsprechende Modul aufgerufen. Die Konfiguration des Datalog-Moduls sowie des XQuery-Moduls ist dabei weitgehend ähnlich, wie anhand der Diagramme ersichtlich wird. Beide Module benötigen eine Verbindung zu einer Datenbank. In dieser Datenbank werden Informationen gespeichert, aus denen sich einzelne Übungsaufgaben zusammensetzen (siehe Kapitel 5). Ähnlich ist außerdem die Abhängigkeit von einem speziellen XSL-Stylesheet (*feedback stylesheet*). Mit diesem werden HTML-Fragmente generiert, aus denen sich die HTML-Seiten zusammensetzen, in denen die Ergebnisse einer Anfrage an das Modul zusammengefasst werden.

Ein wesentlicher Unterschied besteht darin, dass das Datalog-Modul auf eine ausführbare Datei zurückgreift, mit der Datalog-Regeln ausgewertet werden können (siehe Abbildung 4.1). Für die Auswertung von Queries mit dem XQuery-Modul sind hingegen Java-Bibliotheken (*XQuery module*) verfügbar (siehe Abbildung 4.2). Zusätzlich zum XSL-Stylesheet, mit dem die Ergebnisse einer Anfrage an das Modul für die HTML-Seite generiert wird, wird ein weiteres Stylesheet benötigt. Dies hängt damit zusammen, dass für den Vergleich zweier Fragmente – die Auswertungsergebnisse der abgegebenen Lösung und der Musterlösung – im XQuery-Modul auf ein Tool zurückgegriffen wird, das die Unterschiede in Form eines XSL-Stylesheets abbildet. Dieses muss allerdings für die Zwecke des Moduls weiter angepasst werden, was wiederum mit einem XSL-Stylesheet erfolgt, das in der Abbildung als *xmlDiff stylesheet* bezeichnet wird.

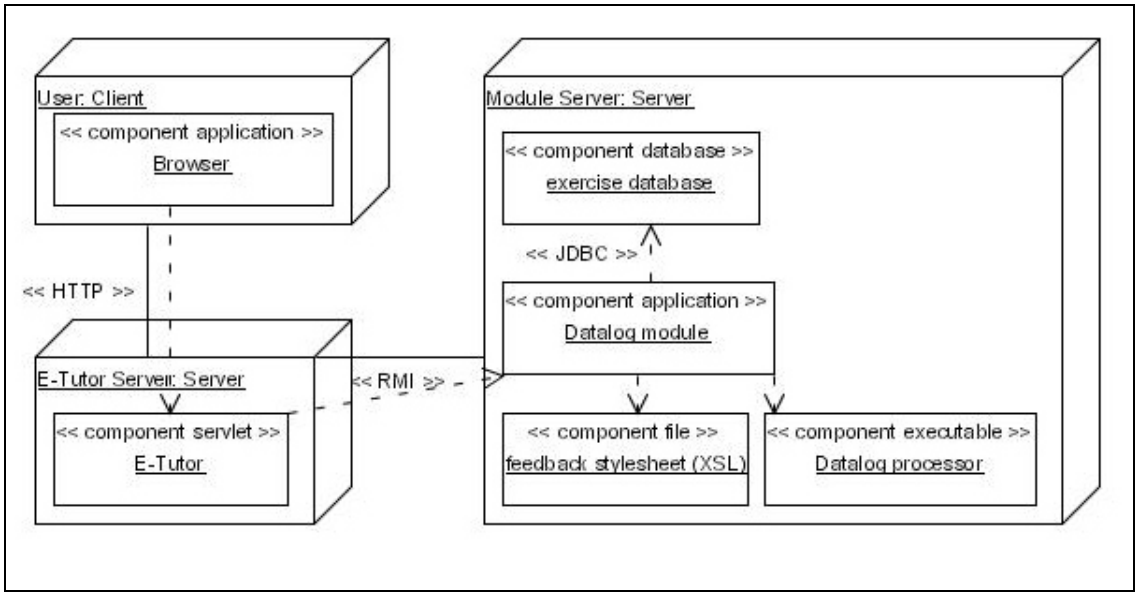

Abbildung 4.1: Verteilungsdiagramm für das Datalog-Modul

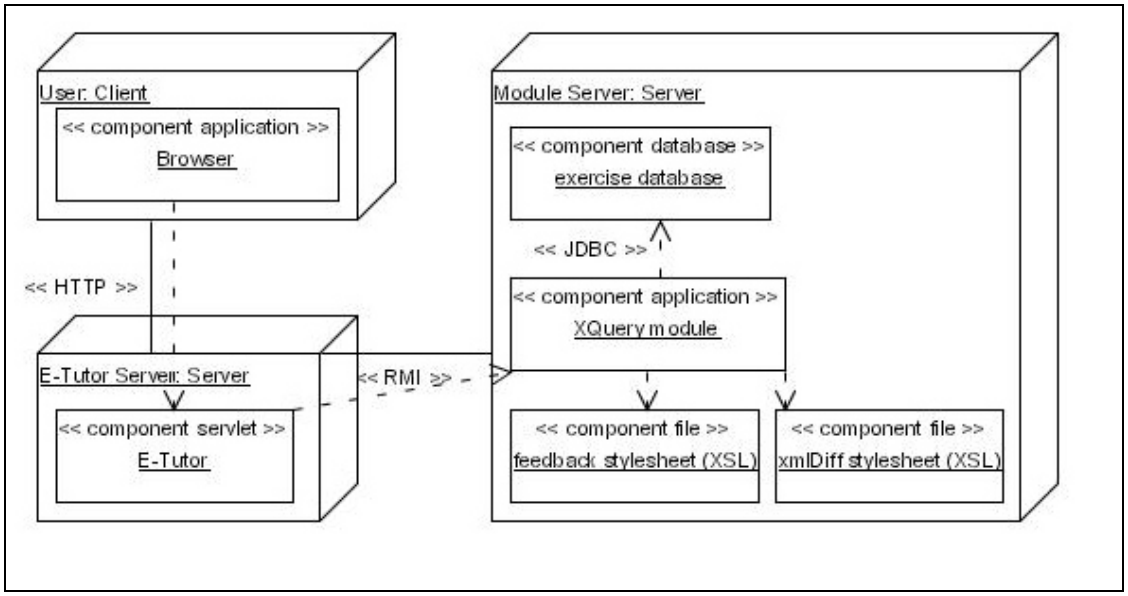

Abbildung 4.2: Verteilungsdiagramm für das XQuery-Modul

4.1.1. Klassenbibliotheken (Libraries)

Die im folgenden erwähnten Java-Klassenbibliotheken werden von den Modulen verwendet und müssen sich dementsprechend beim Aufruf eines Moduls im Klassenpfad befinden. Manche Klassenbibliotheken werden sowohl vom Datalog- als auch vom XQuery-Modul gleichermaßen benötigt, andere wiederum nur speziell von einem Modul. Informationen über die Zuordnung zu den einzelnen Modulen sowie Versionsnummer und Hersteller können der Übersicht in Tabelle 4.1 entnommen werden.

JAR-Datei	**Version**	**Quelle**	**Datalog**	**XQuery**
dlv.jar	2.0	[DLV04]	√	
dom4j.jar	1.4	[DOM04]	√	√
xmlparserv2.jar *xml.jar*	10.1.0.2.0	[Ora04]	√	√
log4j-1.2.8.jar	1.2.8	[Log04]	√	√
ipsi-xq.jar *ipsixq.ui.jar* *xalan.jar* *xercesImpl.jar* *xml-apis.jar* *xmlParserAPIs.jar*	1.3.2	[IPSI04]		√
DDbE.jar *Xerces.jar*		[DDBE99]		√
shiftone-arbor.jar *shiftone-oocjndi.jar*	1.2	[Shif06]	√	√
naming-factory-dbcp.jar	1.2.1	[Apac05]	√	√
ojdbc14.jar	9.0.2		√	√

Tabelle 4.1: Details zu Klassenbibliotheken

Eine Auflistung dieser Klassenbibliotheken mit einer kurzen Erklärung zum Einsatzzweck wird in Tabelle 4.2 gezeigt. Hier werden außerdem die Namen der Packages angeführt, in denen sich Klassen befinden, die im Datalog- bzw. dem XQuery-Modul importiert werden.

JAR-Dateien	**Zweck**	**Package**
dlv.jar	Dient als Java-Schnittstelle zum Datalog-Prozessor, um Datalog-Regeln und die Ergebnisse von Datalog-Regeln als Java-Objekte verarbeitet zu können.	*DLV*
dom4j.jar	Eine XML-Bibliothek für Java.	*org.dom4j*
xmlparserv2.jar *xml.jar*	Eine weitere XML-Bibliothek für Java (enthält u.a. eine XSLT-Engine).	*oracle.xml*
log4j-1.2.8.jar	Spezielle Bibliothek die für das Protokollieren von wichtigen Programmzuständen in einer Log-Datei verwendet wird.	*org.apache.log4j*
ipsi-xq.jar *ipsixq.ui.jar* *xalan.jar* *xercesImpl.jar* *xml-apis.jar* *xmlParserAPIs.jar*	Implementierung eines XQuery-Prozessors	*de.fraunhofer.ipsi*
DDbE.jar *Xerces.jar*	Ein Tool, mit dem sich aus einem vorhandenen XML-Dokument ein entsprechendes XML Schema oder eine DTD erzeugen lässt. Diese Möglichkeit wird im XQuery-Modul genützt, um zwei Lösungen, die als XML-Dokumente vorliegen, auf der	*com.ibm.DDbEv2*

	Basis ihrer Schemata zu vergleichen.	
shiftone-arbor.jar *shiftone-oocjndi.jar*	Wird verwendet, um ein lokal verfügbares JNDI-Verzeichnis zu erzeugen.	
naming-factory-dbcp.jar	Implementierung eines Connection-Pools	
ojdbc14.jar	JDBC-Datenbank-Treiber	

Tabelle 4.2: Übersicht der Klassenbibliotheken

Eine Ausnahme stellen in dieser Übersicht die Klassenbibliotheken *DDbE.jar* und *xerces.jar* dar, die sich im Ordner */etutor/resources/modules/xquery/reflect* befinden (siehe Abbildung 4.5) und deren Klassen zur Laufzeit durch das XQuery-Modul mit einem eigenen Klassenlader geladen werden. Diese Bibliotheken dürfen im Gegensatz zu den übrigen Bibliotheken nicht im Klassenpfad angegeben sein, da es zu Namenskonflikten, bzw. zu Versionskonflikten mit gleichlautenden Klassen der übrigen Bibliotheken kommt.

4.1.2. Query-Prozessor

Um *Datalog*-Queries ausführen zu können, wird auf ein Programm zurückgegriffen, das in Form einer einzelnen, ausführbaren Datei eingesetzt wird (siehe [DLV04]). Das Programm wird grundsätzlich von einer Konsole mit einer Anzahl von möglichen Parametern aufgerufen. In erster Linie können mithilfe dieser Parameter die Dateien angegeben werden, in denen auszuwertende Datalog-Regeln und –Fakten definiert sind. Darüber hinaus ist eine Vielzahl von Parametern definiert, mit denen die Bewertung der Regeln und Fakten beeinflusst werden kann. Im Datalog-Modul wird das Programm mit Hilfe einer Klassenbibliothek eingebunden, die als Schnittstelle zwischen Modul und ausführbarer Datei dient (siehe Abschnitt 4.1.1). Dateiname und Speicherort der ausführbaren Datei werden in einer *Property*-Datei angegeben (siehe Abschnitt 6.1.1).

Im Falle des *XQuery*-Moduls steht eine Java-Klassenbibliothek zur Verfügung, die die Implementierung eines XQuery-Prozessors darstellt. Im Gegensatz zum Datalog-Modul lässt sich der Prozessor daher leichter in das Modul integrieren,

weil dazu lediglich die Klassenbibliothek in den Klassenpfad aufgenommen werden muss.

4.1.3. Packages

Die Klassen der Module werden zu *Java Packages* zusammengefasst, wobei die Struktur und der Verwendungszweck dieser Packages für das Datalog-Modul und das XQuery-Modul vergleichbar sind (siehe Tabelle 4.3).

Packages	**Zweck**
etutor.modules.datalog *etutor.modules.xquery*	Dieses Package enthält Schnittstellen für die Auswertung von Übungen. Die Schnittstellen werden durch die Interfaces *DatalogEvaluator* und *XQEvaluator* und die dazugehörigen Implementierungen *DatalogEvaluatorImpl* und *XQEvaluatorImpl* repräsentiert. Weiters sind in diesen Packages die wichtigsten *Exception*-Klassen definiert.
etutor.modules.datalog.analysis *etutor.modules.xquery.analysis*	Dieses Package enthält die Klassen, mit deren Hilfe Queries vorverarbeitet und ausgewertet, sowie die Ergebnisse für eine eventuelle weitere interne Nutzung aufbereitet werden. Hier befindet sich somit die grundlegende Schnittstelle zum Datalog-Prozessor, bzw. zu den Klassenbibliotheken für den XQuery-Prozessor.
etutor.modules.datalog.exercise *etutor.modules.xquery.exercise*	Hier sind Schnittstellen zu den Modulen zu finden, die die Spezifikation von Übungen unterstützen. Die Schnittstellen werden durch die Interfaces *DatalogExerciseManager* und *XQExerciseManager* und die dazugehörigen Implementierungen *DatalogExerciseManagerImpl* und *XQExerciseManagerImpl* repräsentiert.
etutor.modules.datalog.grading	In diesem Package sind Klassen enthalten, mit

etutor.modules.xquery.grading	deren Hilfe die Beurteilung einer Query, bzw. des Ergebnisses einer Query durchgeführt werden kann.
etutor.modules.datalog.report *etutor.modules.xquery.report*	Hier findet die Aufbereitung der Analyse- und Bewertungsergebnisse für die Rückmeldung an den Benutzer statt.
etutor.modules.datalog.ui *etutor.modules.xquery.ui*	Enthält Klassen, die spezielle Aufgaben für die Umsetzung der Benutzerschnittstelle erfüllen (insbesondere *Servlet*-Implementierungen).
etutor.modules.datalog.util *etutor.modules.xquery.util*	Enthält zentrale Hilfsklassen, die in den übrigen Packages verwendet werden. Dies betrifft vor allem die Bearbeitung von XML-Strukturen und den Zugriff auf benötigte Ressourcen wie etwa *Property*-Dateien. In diesem Package befindet sich somit auch die *Property*-Datei, in der alle Nachrichten, die für Rückmeldungen an den Benutzer vorgesehen sind, zusammengefasst werden.

Tabelle 4.3: Java-Packages

4.1.4. RMI

Die entwickelten Module sollen die Möglichkeit bieten, in das eTutor-System im Sinne einer verteilten Architektur eingebunden zu werden. Zu diesem Zweck sind die Schnittstellen so implementiert, dass sie über *Remote Method Invocation* (RMI) aufgerufen werden können. Bei der Entwicklung der Module wurde daher darauf geachtet, dass die Klassen, die über die RMI-Schnittstelle übermittelt werden, serialisierbar sind.

Nachdem die für die Integration von Modulen in das eTutor-System zu implementierenden Interfaces den RMI-Aspekt noch nicht beinhalten, müssen sie durch modulspezifische Schnittstellen erweitert werden (siehe Abbildung 4.3). So wird etwa das eTutor-Interface *etutor.core.evaluation.Evaluator* im Falle des Datalog-Modules durch ein Interface *etutor.modules.datalog.DatalogEvaluator* erweitert, das gleichzeitig das für RMI vorgesehene Interface *java.rmi.Remote*

erweitert. Gleiches gilt im XQuery-Modul für das Interface *etutor.modules.xquery.XQEvaluator*.

Die Implementierung sowohl des Evaluator-Interfaces als auch des Remote-Interfaces erfolgt durch die Klassen *etutor.modules.datalog.DatalogEvaluatorImpl* (Datalog-Modul) und *etutor.modules.xquery.XQEvaluatorImpl* (XQuery-Modul). Von diesen Klassen wird mithilfe des Java-RMI-Compilers ein sogenannter Stub erzeugt, der vom eTutor-System verwendet wird, um entfernte Methodenaufrufe an das jeweilige Modul zu senden. Das Modul wird auf einem Rechner installiert, wo ein Prozess – die sogenannte RMI-Registry – über einen Port Methodenaufrufe entgegennimmt und an die entsprechenden Implementierungen des *Remote*-Interfaces weiterleitet. Diese Klassen werden in der RMI-Registry mit einer Bezeichnung registriert. Die Bezeichnung wird zusammen mit IP-Addresse und Port verwendet um die entsprechenden Klassen zu identifizieren.

Die Konfiguration der RMI-Registry ist nicht Bestandteil des Datalog-Moduls und des XQuery-Moduls. Dies betrifft v.a. die Wahl der Bezeichnungen und die Verwaltung der benötigten Klassen und Klassenbibliotheken beim Starten der RMI-Registry.

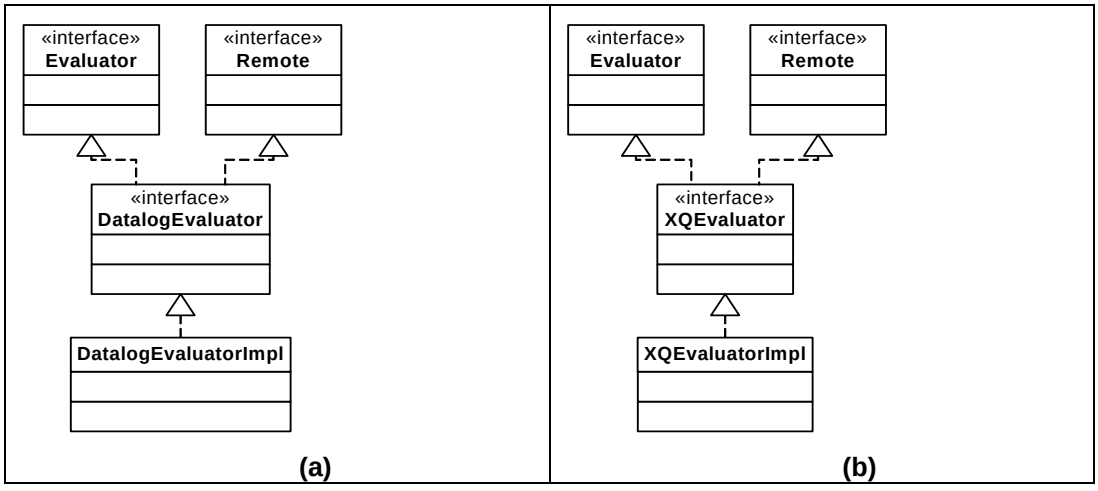

Abbildung 4.3: RMI-Implementierung

4.1.5. Datenbank

Während die wichtigsten Parameter, die für die Ausführung von Queries grundlegend und voraussetzend sind, in einer *Property*-Datei angegeben werden, können in einer Datenbank Informationen gespeichert werden, die eine konkrete Aufgabe hinsichtlich Analyse und Bewertung betreffen. Die Informationen die für

die Definition einer Datalog- oder XQuery-Übungsaufgabe erforderlich sind, werden in einer Datenbank abgespeichert und von dort anhand eines eindeutigen Schlüssels abgerufen.

Die Definition der Tabellen, in denen verschiedene Parameter zu Übungsaufgaben gespeichert werden können, sowie einige ausführliche Beispiele zur Speicherung werden in Kapitel 5 beschrieben.

4.2. Ordnerstruktur

In Abbildung 4.4 und Abbildung 4.5 wird die Struktur des Datalog- und des XQuery-Moduls dargestellt. Der Aufbau entspricht den Konventionen, die für die Integration von Modulen in das eTutor-System vorgegeben sind. Demnach gibt es ein gemeinsames Java-Package *etutor.modules*, in dem sich die untergeordneten Packages des Datalog- und des XQuery-Moduls befinden. Analog dazu befinden sich in *etutor.resources* alle modulspezifischen Ressourcen und in einem Ordner *lib* alle benötigten Klassenbibliotheken. Die getrennte Darstellung in den beiden Abbildungen kann auch als Hinweis darauf gedeutet werden, dass die Module voneinander unabhängig eingesetzt werden und sich an verschiedenen Speicherorten, bzw. auf verschiedenen Rechnern befinden können.

Die Packages des Datalog- und des XQuery-Moduls, die sich im gemeinsamen Package *etutor.modules* befinden, werden in Abschnitt 4.1.3 beschrieben. Die Bedeutung der Ressourcen im Package *etutor.resources* sowie der Datei *ooc-jndi.xml* wird in Kapitel 6 erläutert. Auf die Klassenbibliotheken des Ordners *lib* wird in Abschnitt 4.1.1 eingegangen.

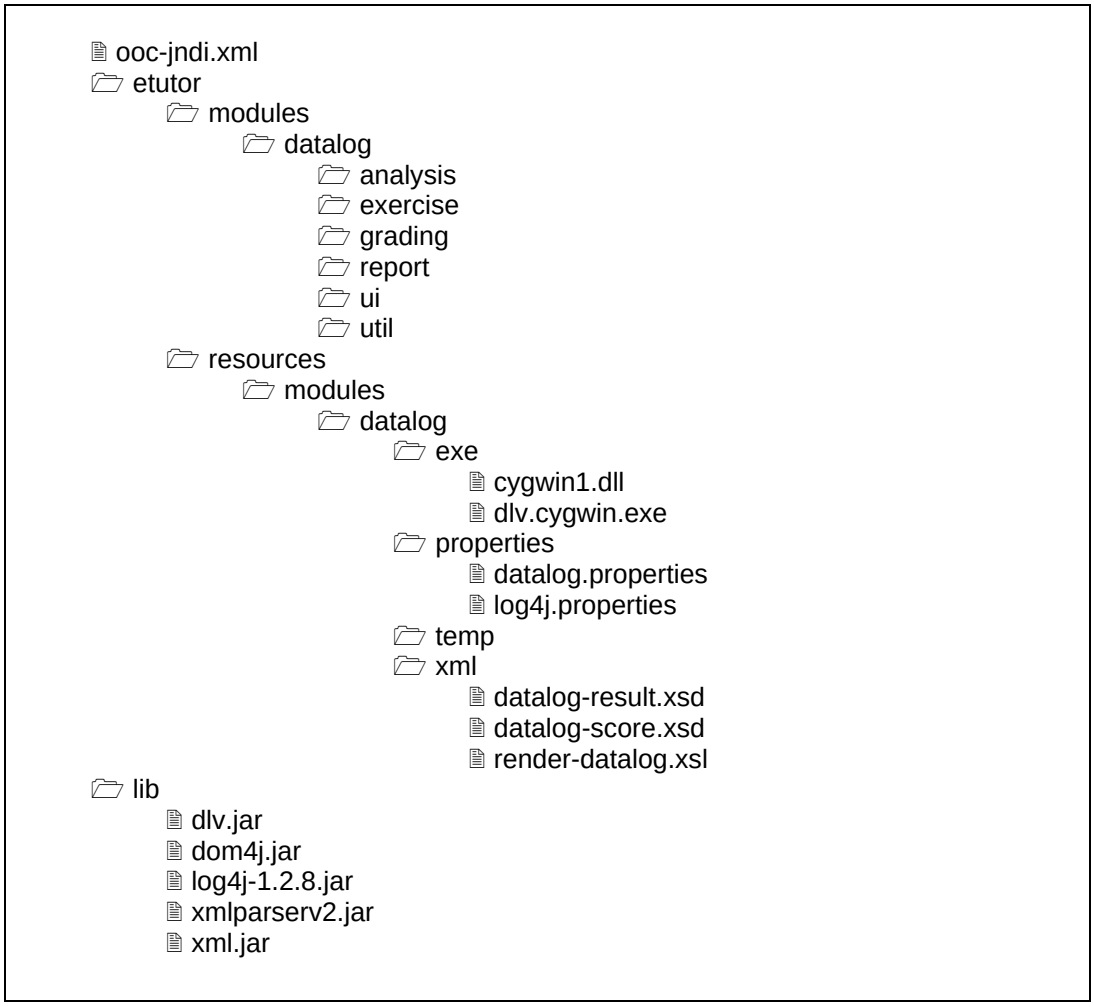

Abbildung 4.4: Ordnerstruktur des Datalog-Moduls

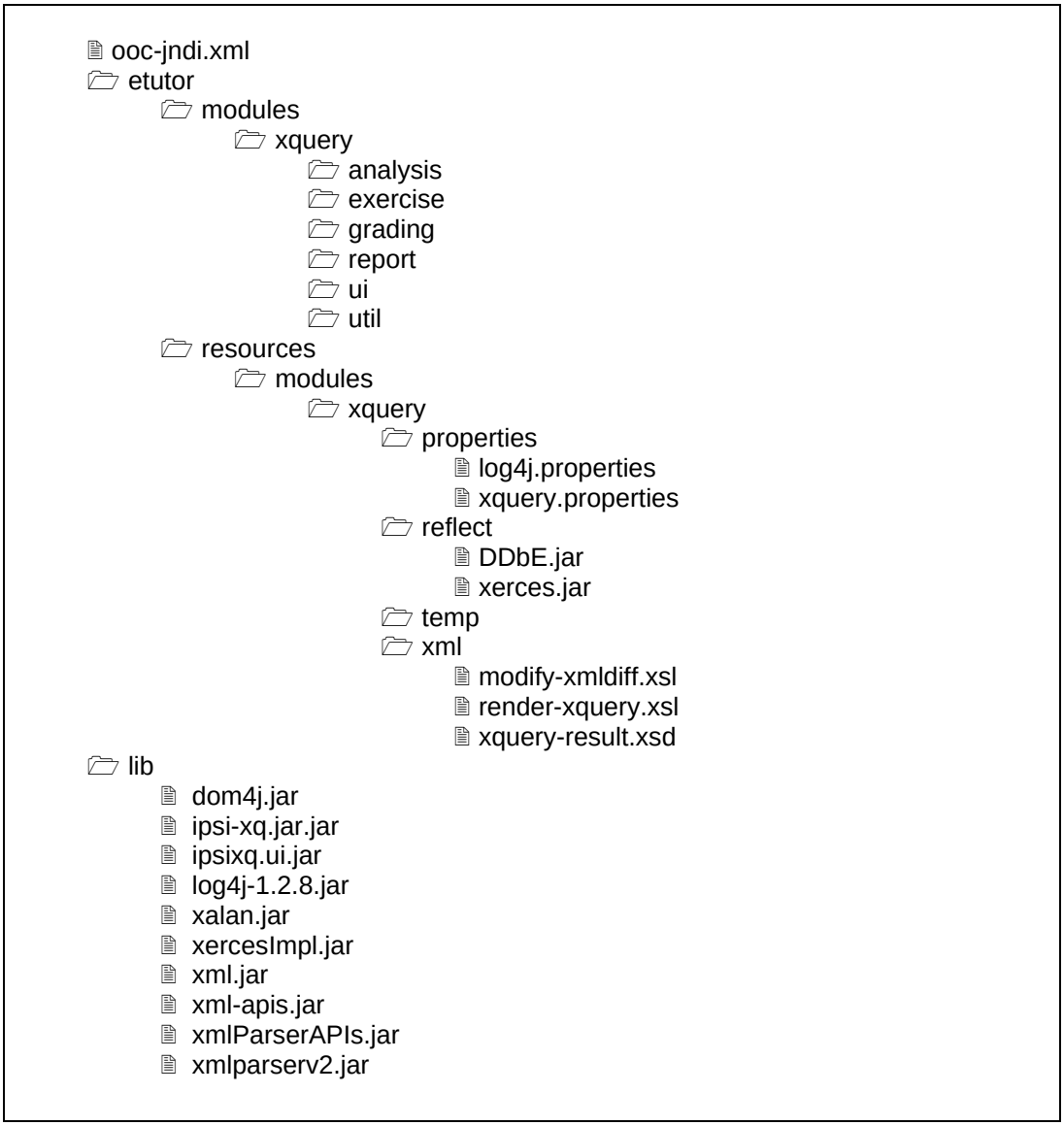

Abbildung 4.5: Ordnerstruktur des XQuery-Moduls

5. Definition von Übungsaufgaben

Im folgenden werden das Datenbankschema und die Bedeutung der Tabellen erläutert, die für die Definition von Übungsaufgaben verwendet werden. Die Erläuterungen werden anhand einiger Beispiele verdeutlicht. In gewissen Punkten ist die Bedeutung einzelner Tabellen zwischen dem Datalog- und dem XQuery-Modul vergleichbar, wobei an entsprechenden Stellen der folgenden Unterkapitel auf diesen Umstand hingewiesen wird.

Die Einhaltung des Schemas gewährleistet weitgehend die problemlose Verarbeitung der in der Datenbank enthaltenen Informationen durch das Modul. Wenn in den Tabellen Werte gespeichert sind, die ungültig sind und nicht verarbeitet werden können, wird die Ausführung durch das Modul abgebrochen und eine entsprechende Fehlermeldungen generiert. Die SQL-Statements, mit denen sich das Schema für das Datalog- und das XQuery-Modul anlegen lässt, sind im Anhang dieser Arbeit aufgelistet. Von Spaltenbezeichnungen, Datentypen und Referenzen abgesehen können die Bezeichnungen der Tabellen frei gewählt werden. Die Zuordnung zu den Tabellen erfolgt über eine *Property*-Datei (siehe Abschnitt 6.1.1).

5.1. Datalog

In diesem Abschnitt erfolgt die Beschreibung der Datenbanktabellen, die benötigt werden, um Informationen für konkrete Übungsaufgaben für das Datalog-Modul definieren zu können. Das dazugehörige Datenbankschema wird in Form eines UML-Diagramms in Abbildung 5.1 gezeigt.

Das Konstrukt, dem im Diagramm die Bezeichnung *exercise* zugewiesen wurde, repräsentiert die zentrale Tabelle, aus der die Informationen für eine Übungsaufgabe ausgelesen werden. Von dieser Tabelle wird auf Tabellen referenziert, die die folgenden Zwecke erfüllen:

- Spezifikation von Punkteabzügen für die Bewertung von Queries (*error_categories*, *error_gradings*, *error_grading_group*),
- Spezifikation der Fakten-Basis für eine Übungsaufgabe (*facts*),
- Spezifikation von Termen innerhalb der Fakten-Basis, die bei einer zusätzlichen Kontrolle der Gültigkeit einer Datalog-Query unbeachtet bleiben können (*unchecked_terms*), und
- Spezifikation von Prädikaten, deren Korrektheit im Ergebnis einer analysierten Lösung überprüft werden soll (*predicates*).

Zu beachten ist, dass Angaben zu Punkteabzügen und Spezifikationen der Fakten-Basis unabhängig existieren können, während die übrigen Informationen als Teil einer bestimmten Übungsaufgabe zu sehen sind.

Angemerkt sei darüber hinaus, dass das Datalog-Modul zwar eine weitgehend differenzierte Spezifikation von Punkteabzügen unterstützt, diese Möglichkeiten jedoch in der Praxis eher nicht ausgeschöpft werden sollten, da die Nachvollziehbarkeit und die Angemessenheit von Punkteabzügen durch komplexe Regeln negativ beeinflusst wird. Aus diesem Grund können die Tabellen für die Definition der Punkteabzüge unbeachtet gelassen werden, wenn die Standardregeln ausreichend sind.

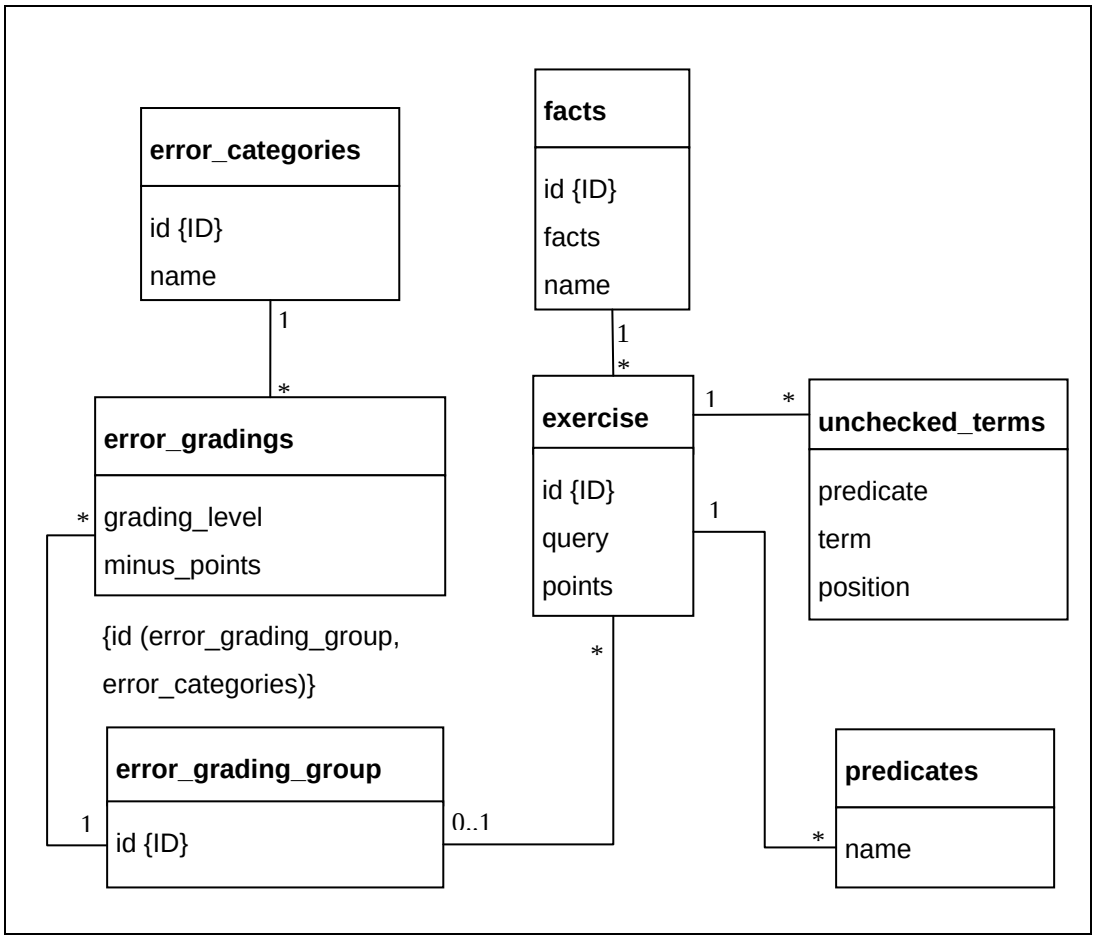

Abbildung 5.1: Datenbankschema für Datalog-Aufgaben

Die folgenden Abschnitte entsprechen sinngemäß den Datenbanktabellen, aus denen sich das Diagramm zusammensetzt. In den SQL-Statements, die für die Schemadefinition verwendet werden und die im Anhang aufgelistet sind, wird ein Schema *etutor_datalog* verwendet, in dem die Tabellen gespeichert werden.

5.1.1. Fehler-Kategorien

Die Tabelle *error_categories* enthält die Definition für mögliche Kategorien von Fehlern in Queries. Diese Fehler werden durch das Modul analysiert, sobald sichergestellt ist, dass sowohl die Musterlösung als auch die zu analysierende Lösung syntaktisch korrekt und ausführbar ist.

Spaltenbezeichnung	**Erklärung**
name	Bezeichnung für die Kategorie; Während die *id* als Schlüssel für Tabellen dient, die diese Tabelle referenzieren, dient *name* als Schlüssel für die Kategorie,

	wie sie vom Modul verarbeitet wird. Nicht zu letzt deshalb gibt es, abgesehen von den vordefinierten Kategorien, keine weiteren definierbaren Kategorien. Die Bezeichnung muss dabei genau eingehalten werden, andernfalls wird bei einem Aufruf des Moduls die Ausführung mit einer Fehlermeldung abgebrochen.
id	Eindeutiger Schlüssel für eine Kategorie, der frei gewählt werden kann.

Tabelle 5.1: Tabellenspalten für Fehlerkategorien

Die Kategorien, die das Modul verwendet, und durch den Kategorienamen identifiziert werden, werden in Tabelle 5.2 aufgelistet.

Kategoriename	**Erklärung**
missing-predicate	Prädikate, die in der Lösung gänzlich fehlen
redundant-predicate	Prädikate, die in der Lösung gänzlich überflüssig sind
low-term-predicate	Prädikate, die in der Lösung zwar enthalten sind, aber zu wenige Terme haben
high-term-predicate	Prädikate, die in der Lösung zwar enthalten sind, aber zu viele Terme haben
missing-fact	Fakten, die innerhalb eines Prädikats der Lösung fehlen
redundant-fact	Fakten, die innerhalb eines Prädikats der Lösung überflüssig sind
negative-fact	Fakten, die fälschlicherweise verneint sind
positive-fact	Fakten, die fälschlicherweise nicht verneint sind

Tabelle 5.2: Datalog-Fehlerkategorien

5.1.2. Bewertungseinheiten

Die Tabelle *error_grading_group* enthält den jeweiligen Schlüssel für eine Anzahl von Fehlern, die zu einer Datalog-Aufgabe zugeordnet werden können und somit bei der Bewertung berücksichtigt werden.

Spaltenbezeichnung	**Erklärung**
id	Eindeutiger Schlüssel für eine Gruppe oder Einheit von definierten Fehlern, der frei gewählt werden kann.

Tabelle 5.3: Tabellenspalten für Bewertungseinheiten

5.1.3. Fehler-Bewertungen

Die Tabelle *error_gradings* enthält die konkrete Definition von Fehlern wie sie bei der Bewertung einer Datalog-Query berücksichtigt werden.

Spaltenbezeichnung	**Erklärung**
grading_group	Fremdschlüssel, mit dem sich mehrere Einträge dieser Tabelle zu einer Einheit zusammenfassen lassen, die einer Datalog-Aufgabe zugeordnet werden kann.
grading_level	Gibt das Niveau an, mit dem der jeweilige Fehler bei der Bewertung berücksichtig wird. Die vom Modul vorgegebenen Werte sind - **1** wenn für jeden Fehler innerhalb der angegebenen Kategorie Punkte abgezogen werden, - **2** wenn Punkte für die Kategorie abgezogen werden, sobald zumindest ein Fehler innerhalb dieser Kategorie aufgetreten ist, - **3** wenn alle Punkte für die gesamte Aufgabe abgezogen werden sollen, sobald ein Fehler der angegebenen Kategorie aufgetreten ist (K.O. Kriterium). Ein anderer als die hier angegebenen Werte führt bei einem

	Aufruf des Moduls zum Abbruch der Ausführung mit einer Fehlermeldung.
grading_category	Referenz auf eine Fehlerkategorie (siehe Abschnitt 5.1.1)
minus_points	Gibt die Punkte an, die für einen Fehler der angegebenen Fehlerkategorie entsprechend dem angegebenen LEVEL abgezogen werden. Falls LEVEL 3 definiert wird, ist eine Angabe der Punkte irrelevant. Für die korrekte Interpretation ist es außerdem nicht von Bedeutung, ob negative oder positive Zahlen angegeben werden.

Tabelle 5.4: Tabellenspalten für Bewertungen

Im folgenden Beispiel wird eine Fehlerkategorie näher spezifiziert:

```
INSERT INTO etutor_datalog.error_gradings VALUES (1, 2, 5, 1.5);
```

Hier stellt 1 eine Nummer dar, unter der mehrere Einträge dieser Tabelle zusammengefasst werden können.

Mit der darauf folgenden Ziffer 2 wird das LEVEL definiert, d.h. ein Fehler der betreffenden Kategorie wird so behandelt, dass die Punkte einmal abgezogen werden, sobald dieser Fehler auftritt.

Die Ziffer 5 bezieht sich in diesem Fall auf die ID einer Fehlerkategorie aus der Tabelle *error_categories*, die z.B. für fehlende Fakten steht (*missing-fact*).

Mit 1.5 wird der Punkteabzug für ein Fakt definiert, das im Ergebnis der Query fehlt, wobei diese Punkte dem LEVEL entsprechend nur höchstens einmal abgezogen werden.

5.1.4. Fakten (Datenbasis)

Die Tabelle *facts* enthält die Definition für Fakten die als Datenbasis für eine Query dienen.

Spaltenbezeichnung	**Erklärung**
facts	Datalog-Fakten, die als Text gespeichert werden.
id	Eindeutiger Schlüssel.

name	Kurze Beschreibung der Datalog-Fakten

Tabelle 5.5: Tabellenspalten für Datalog-Fakten

5.1.5. Nicht überprüfte Terme

Bei der Formulierung von Datalog-Queries gibt es eine Vielzahl von Möglichkeiten, um zu dem in einer Übungsaufgabe gefragten Ergebnis zu gelangen. Darunter befindet sich auch die Möglichkeit, die gewünschten Fakten einfach selbst zu deklarieren anstatt durch eine Query abzuleiten. Nachdem sich dieser Lösungsansatz aber nicht mit den Lernzielen im Datalog-Modul deckt, wird versucht, eine auf diese Weise zustandegekommene Lösung als nicht korrekt zu entlarven.

Die Datenbasis wird vor der Ausführung einer Query so manipuliert, dass sich das Ergebnis mit hoher Wahrscheinlichkeit von dem Ergebnis unterscheidet, das auf Basis der ursprünglichen Datalog-Fakten geliefert wird. Ist dies nicht der Fall, so wird angenommen, dass die gefragten Fakten oder Teile davon vom Benutzer selbst deklariert wurden. Für den Benutzer ist weder die Tatsache, dass die Fakten geändert werden, noch die Art und Weise, wie sie geändert werden, bekannt. Das Ergebnis darf dem Benutzer daher bei dieser Art der Überprüfung nicht angezeigt werden, da es für ihn nicht nachvollziehbare Werte enthält.

Die Manipulation erfolgt automatisch, indem alle Terme in den Datalog-Fakten geändert werden. Die Tabelle *unchecked_terms* dient dazu, einzelne Terme von dieser Manipulation auszuschließen. Dies ist u.a. dann notwendig, wenn bestimmte Terme in einer Query direkt angesprochen werden (z.B. Suche nach dem Mitarbeiter mit der ID ‚15').

Spaltenbezeichnung	**Erklärung**
predicate	Name des Prädikats (z.B. '*employee*'), das die entsprechenden Datalog-Fakten umfasst.
position	Position des Terms innerhalb der Fakten
term	Ausprägung des Terms innerhalb der Fakten
exercise	Fremdschlüssel der Übungsaufgabe, für die ein Term definiert wird.

Tabelle 5.6: Tabellenspalten für nicht überprüfte Terme

5.1.6. Erforderliche Prädikate

In der Tabelle *predicates* werden einzelne Prädikate definiert, die im Ergebnis einer Datalog-Query enthalten sein müssen, um im Sinne der Aufgabenstellung korrekt zu sein. Solange keine Prädikate zu einer Übungsaufgabe definiert sind, ist jede analysierte Query korrekt.

Spaltenbezeichnung	**Erklärung**
name	Name des Prädikats (z.B. '*employee*'), wobei eine eventuelle Analyse gestoppt wird, falls ein Prädikat im Zusammenhang mit einer Query angegeben wird, in deren Lösung das Prädikat nicht enthalten ist.
exercise	Fremdschlüssel der Übungsaufgabe, für die das Prädikat definiert wird.

Tabelle 5.7: Tabellenspalten für Prädikate

5.1.7. Datalog-Übungsaufgabe

In der Tabelle *exercise* werden schlussendlich konkrete Übungsaufgaben definiert.

Spaltenbezeichnung	**Erklärung**
id	Eindeutiger Schlüssel für eine Übungsaufgabe.
query	Datalog-Query, die die Musterlösung zu einer Übungsaufgabe repräsentiert.
facts	Fremdschlüssel für die Tabelle, in der die Fakten für Übungsbeispiele enthalten sind.
gradings	Fremdschlüssel für eine Gruppe von definierten Fehlerkategorien und den jeweiligen Informationen über die Punkteabzüge. Wenn keine Gruppe angegeben wird oder innerhalb der Gruppe keine Kategorien definiert wurden, wird bei der Bewertung einer Query

	standardmäßig das K.O. Kriterium (LEVEL **3**) für jede Kategorie herangezogen (siehe Abschnitt 5.1.3).
points	Die maximal zu erreichende Punkteanzahl für diese Aufgabe.

Tabelle 5.8: Tabellenspalten für Datalog-Übungsaufgaben

5.2. XQuery

In diesem Abschnitt erfolgt die Beschreibung der Datenbanktabellen, die benötigt werden, um Informationen für konkrete Übungsaufgaben für das XQuery-Modul definieren zu können. Das dazugehörige Datenbankschema wird in Form eines UML-Diagramms in Abbildung 5.2 gezeigt.

Die Tabelle mit der Bezeichnung *exercise* repräsentiert hier ebenso wie im Datenbankschema für das Datalog-Modul die zentrale Tabelle, aus der die Informationen für eine Übungsaufgabe ausgelesen werden. Von dieser Tabelle wird auf Tabellen referenziert, die die folgenden Zwecke erfüllen:

- Spezifikation von Punkteabzügen für die Bewertung von Queries (*error_categories*, *error_gradings*, *error_grading_group*),
- Speicherung von XML-Dokumenten in der Datenbank (*xmldocs*),
- Spezifikation von URLs zu XML-Dokumenten, die als Datenbasis für eine konkrete XQuery-Übungsaufgabe dienen (*exercise_urls*), und
- Spezifikation von XML-Knoten im Ergebnis einer XQuery, auf deren Sortierung Wert gelegt wird (*sortings*).

Zu beachten ist, dass Angaben zu Punkteabzügen unabhängig existieren können, während Informationen über URLs und Sortierungen als Teil einer bestimmten Übungsaufgabe zu sehen sind. Die Tabelle *xmldocs* nimmt eine Sonderstellung ein, da im Datenbankschema keine Abhängigkeiten mit den übrigen Tabellen vorgesehen sind.

Angemerkt sei darüber hinaus, dass das XQuery-Modul ebenso wie das Datalog-Modul zwar eine weitgehend differenzierte Spezifikation von Punkteabzügen unterstützt, diese Möglichkeiten jedoch in der Praxis eher nicht ausgeschöpft werden sollten, da die Nachvollziehbarkeit und die Angemessenheit von Punkteabzügen durch komplexe Regeln negativ beeinflusst wird. Aus diesem

Grund können die Tabellen für die Definition der Punkteabzüge unbeachtet gelassen werden, wenn die Standardregeln ausreichend sind.

Die folgenden Abschnitte entsprechen den Datenbanktabellen, aus denen sich das Diagramm zusammensetzt. In den SQL-Statements, die für die Schemadefinition verwendet werden und die im Anhang aufgelistet sind, wird ein Schema *etutor_xquery* verwendet, in dem die Tabellen gespeichert werden.

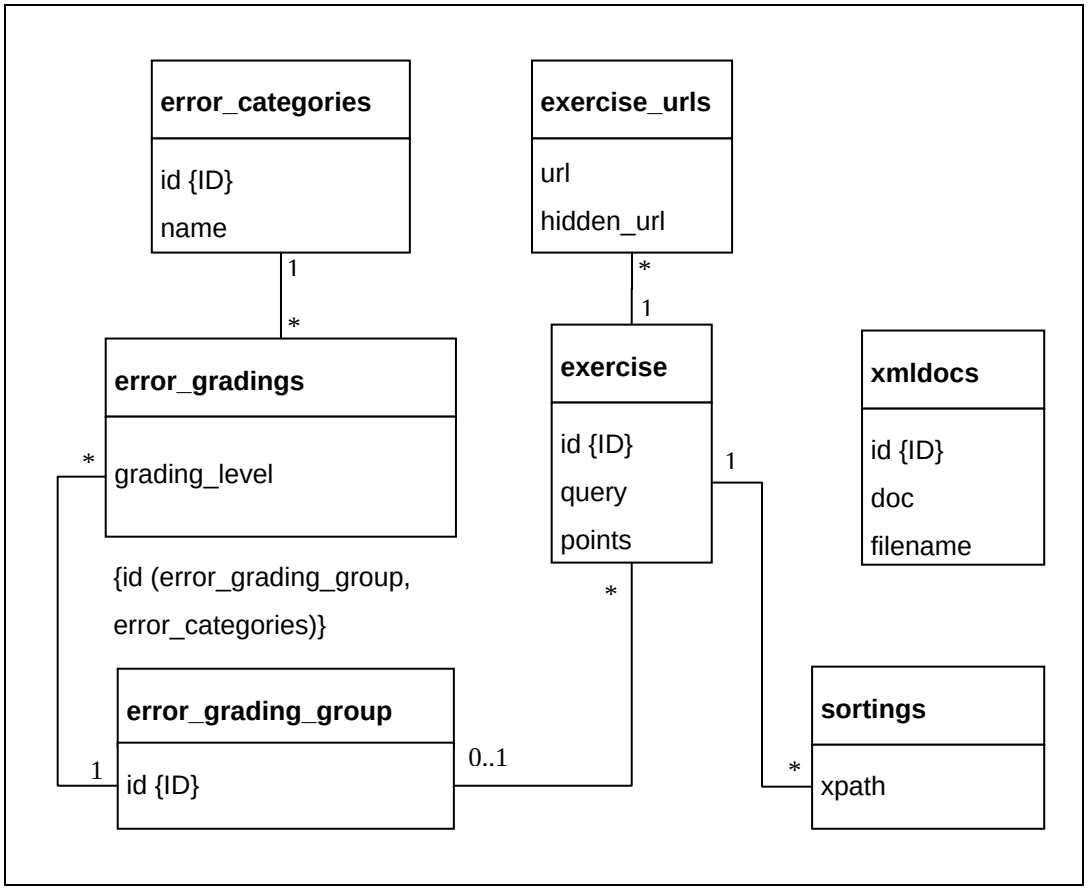

Abbildung 5.2: Datenbankschema für XQuery-Aufgaben

5.2.1. Fehler-Kategorien

Sinn und Zweck der Tabelle *error_categories* entsprechen der Beschreibung in Abschnitt 5.1.1, wobei der entscheidende Unterschied zur Tabelle für das Datalog-Modul in den Fehlerkategorien liegt, die für das XQuery-Modul vorgesehen sind. Eine Übersicht der Kategorien, die vom Modul erkannt werden, wird in Tabelle 5.9 gezeigt.

Kategoriename	**Erklärung**

missing-node	XML-Knoten, die an einer bestimmten Stelle in der XML-Struktur einer Lösung gänzlich fehlen
redundant-node	XML-Knoten, die an einer bestimmten Stelle in der XML-Struktur einer Lösung gänzlich überflüssig sind
displaced-node	XML-Knoten, die zwar auf einer bestimmten Hierarchiestufe in der XML-Struktur korrekt vorhanden sind, allerdings nicht an der erwarteten Position; Dies wird nur als Fehler behandelt, wenn in der Aufgabenstellung definiert worden ist, dass der betreffende Knoten zwingend sortiert sein und somit die selbe Ordnung aufweisen muss wie die entsprechenden Knoten in der Musterlösung.
missing-attribute	XML-Attribute, die in einem bestimmten XML-Element gänzlich fehlen.
redundant-attribute	XML-Attribute, die in einem bestimmten XML-Element gänzlich überflüssig sind.
incorrect-value	XML-Attribute, die zwar in einem bestimmten XML-Element erwartet werden und vorhanden sind, allerdings mit einem falschen Wert

Tabelle 5.9: XQuery-Fehlerkategorien

5.2.2. Bewertungseinheiten

Die Tabelle *error_grading_group* wird analog zur der entsprechenden Tabelle des Datalog-Moduls, die in Abschnitt 5.1.2 beschrieben wird, verwendet.

5.2.3. Fehler-Bewertungen

Auch die Tabelle *error_gradings* wird benötigt, um die Bewertung für Queries festlegen zu können, und wird wiederum analog zur entsprechenden Tabelle des Datalog-Moduls, die in Abschnitt 5.1.3 beschrieben wird, verwendet.

5.2.4. Verweis auf XML-Dokumente einer Übungsaufgabe

In der Tabelle *exercise_urls* können mehrere XML-Dokumente definiert werden, die als Datenbasis für eine XQuery-Aufgabe dienen.

Die Datenbasis wird in Form von URLs angegeben, die in einem XQuery-Ausdruck eingebaut werden können. Auch wenn davon ausgegangen wird, dass in der Regel pro Übungsaufgabe nicht mehr als ein XML-Dokument benötigt wird, so kann eine theoretisch unbegrenzte Anzahl von Dokumenten als Datenbasis festgelegt werden.

In Abschnitt 5.2.6 wird eine Möglichkeit beschrieben, wie über URLs auf die XML-Dokumente in der dafür vorgesehenen Datenbanktabelle zugegriffen werden kann.

Spaltenbezeichnung	**Erklärung**
url	Eine URL, über die das XML-Dokument abgerufen werden kann. Die URL muss dabei exakt in derselben Form angegeben werden, in der sie im XQuery-Ausdruck der Musterlösung verwendet wird (z.B. '*http://etutor.dke.uni-linz.ac.at/etutor/XML?id=1*'). Bleibt in einem Ausdruck eine URL undefiniert, so wird bei der Analyse einer Abgabe zu dieser Übungsaufgabe die Ausführung mit einer Fehlermeldung abgebrochen. Gleiches gilt für eine URL, die für ein und die selbe Übungsaufgabe mehrfach definiert wird.
hidden_url	Eine URL, über die ein beliebiges weiteres XML-Dokument abgerufen werden kann. Die in der Spalte *url* definierte URL wird bei einer Abgabe mit dieser URL ersetzt, um zu überprüfen, ob ein potentiell richtiges Ergebnis darauf beruht, dass die Daten aus der Datenbasis abgeleitet wurden, oder ob sie vom Benutzer in seiner Query lediglich selbst deklariert wurden. Die Details dieser Überprüfung werden dem Benutzer nicht erkennbar gemacht.
exercise	Fremdschlüssel der Übungsaufgabe, für die die Angaben gelten.

Tabelle 5.10: Tabellenspalten für Verweise auf XML-Dokumente

5.2.5. Sortierte XML-Knoten

In der Tabelle *sortings* können nun einzelne XML-Knoten definiert werden, die in einer bestimmten Aufgabe korrekt sortiert sein müssen. Die Sortierung richtet sich dabei an der Anordnung der entsprechenden Knoten in der Musterlösung.

Spaltenbezeichnung	**Erklärung**
xpath	Hier wird ein XPath-Ausdruck angegeben, der die Knoten im Ergebnis der Musterlösung angibt, auf deren Sortierung Wert gelegt wird, z.B.: - //wohnung - //personen/person - //*
exercise	Fremdschlüssel der Übungsaufgabe, für die die Angaben gelten.

Tabelle 5.11: Tabellenspalten für sortierte XML-Knoten

5.2.6. XML-Dokumente (Datenbasis)

Im allgemeinen können für eine Übungsaufgabe beliebige URLs spezifiziert werden, solange durch diese URLs XML-Dokumente lokalisiert werden, die im Internet verfügbar sind.

Um im speziellen Fall die XML-Dokumente aus der Tabelle *xmldocs* verwenden zu können, wird ein Servlet *etutor.modules.xquery.ui.XMLServlet* implementiert, das anhand des Identifikations-Parameters *id* das entsprechende Dokument ausliest und zurückgibt. Eine URL für eine Übungsaufgabe kann somit zwar auf ein XML-Dokument in der Tabelle *xmldocs* verweisen, die Abhängigkeit wird im Datenbankschema jedoch nicht explizit überprüft.

Das Servlet bietet zusätzlich die Funktionalität, das angeforderte XML-Dokument automatisch zu verfälschen, indem es in allen enthaltenen XML-Elementen ein Test-Attribut einfügt. Diese Funktionalität kann für den in Abschnitt 5.2.4 beschriebenen Einsatzzweck genutzt werden, um das XML-Dokument zu

deklarieren, das zusätzlich zum eigentlichen XML-Dokument zur Überprüfung der Plausibilität einer Lösung verwendet wird. In diesem Fall wird der Identifikations-Parameter in der URL um den Zusatz ‚_enc' erweitert, wie das folgende Beispiel zeigt:

http://etutor.dke.uni-linz.ac.at/etutor/XML?id=1_enc

In diesem Beispiel wird angenommen, dass die Web-Applikation, in der das eTutor-System läuft, die an die URL gesendete Anfrage an das Servlet *etutor.modules.xquery.ui.XMLServlet* weiterleitet.

Spaltenbezeichnung	**Erklärung**
id	Eindeutiger Schlüssel eines XML-Dokuments
doc	Das XML-Dokument, das in der Datenbank gespeichert wird.
filename	Dateiname, unter dem das XML-Dokument gespeichert wird. Dieser Dateiname wird u.a. bei einem HTTP-Request zusammen mit dem ausgelesenen XML-Dokument als Meta-Information zurückgeliefert.

Tabelle 5.12: Tabellenspalten für XML-Dokumente

5.2.7. XQuery-Übungsaufgabe

In der Tabelle *exercise* können konkrete XQuery-Übungsaufgaben definiert werden.

Spaltenbezeichnung	**Erklärung**
id	Eindeutiger Schlüssel für eine Übungsaufgabe.
query	XQuery-Ausdruck, der die Musterlösung zu einer Übungsaufgabe repräsentiert.
gradings	Fremdschlüssel für eine Gruppe von definierten Fehlerkategorien und den jeweiligen Informationen über die Punkteabzüge. Wenn keine Gruppe angegeben wird oder innerhalb der Gruppe keine Kategorien definiert wurden, wird bei der Bewertung einer Query

	standardmäßig das K.O. Kriterium (LEVEL **3**) für jede Kategorie herangezogen (siehe Abschnitt 5.1.3).
points	Die maximal zu erreichende Punkteanzahl für diese Aufgabe.

Tabelle 5.13: Tabellenspalten für XQuery-Übungsaufgaben

6. Konfiguration

Das Modul stellt diverse Möglichkeiten zur Konfiguration zur Verfügung, die in den folgenden Abschnitten erläutert werden sollen. Die Dateinamen und –pfade sind für die Dateien vorgegeben und müssen sich daher in den in Abbildung 4.4 (XQuery-Modul), bzw. Abbildung 4.5 (Datalog-Modul) angegebenen Ordnern befinden. Um den Zugriff auf diese Ressourcen über deren Dateipfade zentral zu verwalten, werden die relativen Pfade in Attributen der folgenden Klassen zusammengefasst:

etutor.modules.datalog.DatalogCoreManager
etutor.modules.xquery.XQCoreManager

Für die unmittelbare Konfiguration sind vor allem die Property-Dateien relevant, mit denen Parameter, die für die Ausführung der Module von Bedeutung sind, eingestellt werden können. Diese Dateien werden im folgenden Abschnitt beschrieben. Darauf folgend werden die Dateien erläutert, die in den Modulen mittels XSLT (Extensible Stylesheet Language Transformations) für die interne Transformation von XML-Dokumenten verwendet werden. Da diese Dateien den Programmablauf und die Ergebnisse grundlegend beeinflussen, sind Änderungen innerhalb dieser Dateien mit Bedacht zu wählen.

6.1. Property-Dateien

Während in der Datenbank konkrete Aufgaben formuliert werden, dienen die Property-Dateien dazu, allgemeine Parameter zu definieren, die für das Ausführen von Datalog-Queries, für das Bewerten von Analyseergebnissen und die Präsentation der Ergebnisse grundsätzlich herangezogen werden.

6.1.1. Parameter für Analyse und Bewertung

Die betreffende Property-Datei befindet sich für das Datalog-Modul im folgenden Verzeichnis:

/etutor/resources/modules/datalog/properties/datalog.properties

Für das XQuery-Modul ist die entsprechende Datei im folgenden Verzeichnis zu finden:

/etutor/resources/modules/xquery/properties/xquery.properties

Die zu definierenden Parameter werden modulspezifisch in Tabelle 6.1 (Datalog-Modul) und Tabelle 6.2 (XQuery-Modul) zusammengefasst:

Key	**Bedeutung**
java.naming.factory.initial	Eine der wichtigsten Properties, die von der Java Plattform für die Verbindung zu einem JNDI-Verzeichnisdienst vorgesehen sind (siehe Abschnitt 6.2). Zusätzliche JNDI-Properties können angegeben werden.
dlg.java.naming.datasource	Pfad im JNDI-Verzeichnis, in dem das *DataSource*-Objekt konfiguriert ist, das einen Connection-Pool verwaltet (siehe Abschnitt 6.3).
datalog.exe	Angabe des vollständigen Dateipfades der ausführbaren Datei, mit der Datalog-Regeln und –Fakten ausgewertet werden können (siehe Abschnitt 4.1.2).
datalog.timeout	Zeitangabe in Millisekunden, die das Limit definiert, das bei der Ausführung von Queries mithilfe des

	Datalog-Prozessors nicht überschritten werden darf. Wenn das Limit bei der Initialisierung der Musterlösung überschritten wird, wird intern eine Exception ausgelöst. Wenn das Limit hingegen bei der Ausführung einer zu analysierenden Query überschritten wird, erhält der Benutzer eine dementsprechende Fehlermeldung.
datalog.interval	Zeitangabe in Millisekunden, die definiert, in welchen Abständen bei der Ausführung einer Query überprüft werden soll, ob das Zeitlimit überschritten worden ist.
datalog.max.int	Dieser Parameter wird vom Datalog-Prozessor dann benötigt, wenn mit numerischen Werten gerechnet werden muss. Wenn das angegebene Maximum für Zahlen überschritten wird, wird vom Prozessor ein Fehler angezeigt. Ein relativ hoher Integer-Wert innerhalb des vom Datalog-Prozessor vorgegebenen Wertebereich ist daher zu empfehlen.
table.error.categories	Siehe Abschnitt 5.1.1
table.error.grading.group	Siehe Abschnitt 5.1.2
table.error.grading	Siehe Abschnitt 5.1.3
table.facts	Siehe Abschnitt 5.1.4
table.terms.encryption	Siehe Abschnitt 5.1.5
table.predicates	Siehe Abschnitt 5.1.6
table.exercise	Siehe Abschnitt 5.1.7
modus.debug	Wenn der Wert auf *true* gesetzt wird, werden Zwischenergebnisse, die intern für die Analyse verwendet werden, in Dateien in einem Ordner */etutor/resources/modules/datalog/temp* gespeichert.

Tabelle 6.1: Konfigurationsparameter für das Datalog-Modul

Key	Bedeutung
java.naming.factory.initial	Eine der wichtigsten Properties, die von der Java Plattform für die Verbindung zu einem JNDI-Verzeichnisdienst vorgesehen sind (siehe Abschnitt 6.2). Zusätzliche JNDI-Properties können angegeben werden.
xq.java.naming.datasource	Pfad im JNDI-Verzeichnis, in dem das *DataSource*-Objekt konfiguriert ist, das einen Connection-Pool verwaltet (siehe Abschnitt 6.3).
table.error.categories	Siehe Abschnitt 5.2.1
table.error.grading	Siehe Abschnitt 5.2.3
table.urls	Siehe Abschnitt 5.2.4
table.sortings	Siehe Abschnitt 5.2.5
table.exercise	Siehe Abschnitt 5.2.7
modus.debug	Wenn der Wert auf *true* gesetzt ist, werden Zwischenergebnisse, die intern für die Analyse verwendet werden, in Dateien in einem Ordner */etutor/resources/modules/xquery/temp* gespeichert.

Tabelle 6.2: Konfigurationsparameter für das XQuery-Modul

6.1.2. Logging

Die betreffende Property-Datei befindet sich für das Datalog-Modul im folgenden Verzeichnis:

/etutor/resources/modules/datalog/properties/log4j.properties

Für das XQuery-Modul ist die entsprechende Datei im folgenden Verzeichnis zu finden:

/etutor/resources/modules/xquery/properties/log4j.properties

Diese Datei wird für die Konfiguration des Loggers verwendet, mit dem während der Programmausführung Informationen über bestimmte Ereignisse und Zustände

in eine Log-Datei geschrieben werden (siehe http://logging.apache.org/log4j/docs).

Zu den Konfigurationsmöglichkeiten zählt in erster Linie der Speicherort der Log-Datei und die Einstellung des Log-Levels.

6.1.3. Messages

In einer weiteren Property-Datei können die Nachrichten formuliert werden, die für Rückmeldungen an den Benutzer sowie für Exceptions verwendet werden. Dabei wird das Format verwendet, das in der Klasse *java.text.MessageFormat* der Java 2 Platform, Standard Edition, beschrieben ist.

Im Falle des Datalog-Moduls ist dies die Datei *DLResources.properties*. Diese Datei ist an die Java-Klasse *etutor.modules.datalog.util.DLResources* gebunden, die alle *Keys* der Property-Datei definiert und die Schnittstelle zwischen der Property-Datei und dem Programm darstellt. Die Property-Datei befindet sich dementsprechend im selben Package wie die kompilierte Klasse.

Im XQuery-Modul lautet die entsprechende Property-Datei *XQResources.properties*, die sich im selben Package wie die kompilierte Klasse *etutor.modules.xquery.util.XQResources* befindet.

6.2. JNDI

JNDI (Java Naming and Directory Interface) definiert Schnittstellen, über die Ressourcen in einem Verzeichnis standardisiert verwaltet und abgefragt werden können. Mithilfe eines solchen Verzeichnisdienstes kann eine Anwendung weitgehend unberührt von Konfigurationsdetails gehalten werden. Im speziellen wird JNDI im Datalog- und im XQuery-Modul für die Konfiguration des Datenverbindungs-Pools eingesetzt (siehe 6.3).

Es gibt unterschiedliche Implementierungen von JNDI-Verzeichnisdiensten, die diverse Einsatzgebiete abdecken. Als Beispiele für Einsatzzwecke seien hier LDAP (Leightweight Data Access Protocol) und DNS (Domain Name Service) erwähnt (siehe [Mahm03]). Als Grundvoraussetzung für die Einbindung eines JNDI-Verzeichnisdienstes in einer Anwendung gilt einerseits, dass die herstellerspezifischen Klassendefinitionen, die die Kommunikation mit dem Verzeichnisdienst ermöglichen, im Klassenpfad verfügbar sein müssen, und

andererseits die Definition bestimmter Java System Properties. Dazu zählen in erster Linie die Properties *java.naming.provider.url* für die Verbindung zum Verzeichnisdienst, sowie *java.naming.factory.initial* für die herstellerspezifische Klasse, die den Einstiegspunkt für den Aufbau der Kommunikation mit dem JNDI-Verzeichnis darstellt.

Die folgende Unterscheidung wird getroffen, um verschiedene Szenarien zu beschreiben, wie ein JNDI-Verzeichnisdienst in Modulen des eTutor-Systems eingebunden werden kann:

- *Zentraler JNDI-Verzeichnisdienst:* Ein zentraler JNDI-Verzeichnisdienst läuft als eigenständige und unabhängige Komponente in einem System. Anwendungen, die den JNDI-Verzeichnisdienst benutzen, benötigen wie oben beschrieben lediglich die Klassendefinitionen der JNDI-Implementierung, sowie Festlegung der entsprechenden Java System Properties. Der Vorteil eines zentralen JNDI-Verzeichnisdienstes liegt in der einfachen Konfiguration systemweiter Ressourcen. Im Idealfall können so die Datenbankverbindungen aller eTutor-Komponenten an einer zentralen Stelle konfiguriert werden. Dies würde z.B. die Konfiguration des Datenbankschemas für Testzwecke und für den Produktiveinsatz erleichtern.
- *Integrierter JNDI-Verzeichnisdienst*: JNDI stellt eine der Komponenten dar, aus denen sich ein J2EE-konformes System zusammensetzt. Innerhalb eines Tomcat-Containers ist beispielsweise ein JNDI-Verzechnisdienst integriert, der mithilfe von herstellerspezifischen Konfigurationsdateien mit Ressourcen befüllt werden kann, die dann in Web-Applikationen zur Verfügung stehen [Apac06]. Der JNDI-Verzeichnisdienst selbst ist hier somit im Gegensatz zum zentralen JNDI-Verzeichnisdienst automatisch verfügbar. Dies eignet sich im eTutor-System für Module, die direkt im Web-Container ausgeführt werden.
- *Lokaler JNDI-Verzeichnisdienst*: Einfache JNDI-Implementierungen ermöglichen die Erzeugung eines lokal verfügbaren JNDI-Verzeichnisses, das für eine Anwendung exklusiv und nur für die Dauer der Programmausführung zur Verfügung steht. Meistens stellt dies eine Möglichkeit dar, einzelnen Komponenten einen einfachen Ersatz zu bieten, solange ein bestimmter Verzeichnisdienst nicht verfügbar ist. Ein solcher Ersatz ist zum Beispiel sinnvoll, wenn ein eTutor-Modul für den

Betrieb im Web-Container konzipiert ist, aber außerhalb des Web-Containers getestet wird.

Hinsichtlich der Konfigurierbarkeit ist grundsätzlich jede dieser Alternativen für die Integration eines JNDI-Verzeichnisdienstes mit dem Datalog- und dem XQuery-Modul möglich. Der integrierte JNDI-Verzeichnisdienst eignet sich insofern nicht, als dass die Module eigenständige Komponenten darstellen, die außerhalb des Web-Containers ausgeführt und über RMI integriert werden. Ein zentraler JNDI-Verzeichnisdienst ist für eine einfache, zentrale Konfigurierung der Ressourcen des eTutor-Systems in jedem Fall von Vorteil. Für den Fall, dass kein zentraler JNDI-Verzeichnisdienst zur Verfügung steht, aber auch für Testzwecke, sieht sowohl das Datalog- als auch das XQuery-Modul einen eigenen, lokalen JNDI-Verzeichnisdienst vor.

Die für den lokalen Verzeichnisdienst eingesetzten Klassenbibliotheken werden in Abschnitt 4.1.1 beschrieben. Für die verwendete Implementierung muss das Property *java.naming.factory.inital* in den in Abschnitt 6.1.1 beschriebenen Property-Dateien auf den Wert *org.shiftone.occ.InitialContextFactoryImpl* gesetzt werden. Zur Konfiguration des JNDI-Verzeichnisses benötigt das Tool eine Datei *ooc-jndi.xml*, die sich im Klassenpfad befinden muss (siehe Abschnitt 4.2). Mithilfe einer vom Tool vorgegebenen Syntax wird in dieser Datei eine Verzeichnisstruktur vorgegeben, in der eine Ressource durch die Angabe eines Pfades identifiziert werden kann. Entsprechend einer JNDI-Konvention befinden sich JDBC-Ressourcen im Verzeichnis *java:comp/env/jdbc*.

6.3. Datenbankverbindungen

Datenbankverbindungen werden über einen sogenannten Connection-Pool bereitgestellt. In einem Connection-Pool wird eine bestimmte Anzahl von Datenbankverbindungen verwaltet, die exklusiv angefordert werden können. Die Datenbankverbindung muss vom Empfänger, d.h. vom aufrufenden Objekt, explizit freigegeben werden damit sie wieder verwendet werden kann.

Mit JDBC (Java Database Connectivity) wird von der Java-Plattform eine Schnittstelle definiert, die diesen Mechanismus unterstützt. Ein Connection-Pool wird durch eine Klasse realisiert, die des Interface *javax.sql.DataSource* implementiert. Informationen, die für die Datenbankverbindung notwendig sind, sind bereits in solch einem Objekt konfiguriert, deshalb müssen sie bei der

Anforderung eines Datenverbindungsobjektes nicht mehr angegeben werden. Für das Datalog- und das XQuery-Modul bedeutet dies im speziellen, dass lediglich ein *DataSource*-Objekt verfügbar sein muss. Die Module sind dadurch von der Konfiguration der Parameter für die Datenbankverbindungen und für die Initialisierung des Connection-Pools relativ unabhängig.

Ein Connection-Pool ist ein Beispiel für eine Ressource, die typischerweise in einem JNDI-Verzeichnis konfiguriert wird (siehe Abschnitt 6.2). Für das Datalog- und XQuery-Modul wird dementsprechend jeweils ein *DataSource*-Objekt konfiguriert und unter einem bestimmten Pfad im Verzeichnis abgelegt. Im Datalog-, sowie im XQuery-Modul muss nun lediglich die Verbindung zu einem JNDI-Verzeichnis hergestellt werden und anhand des bekannten Pfades das *DataSource*-Objekt identifiziert werden, über das in weiterer Folge die Datenbankverbindungen abgefragt werden können.

Die Klassenbibliotheken des Connection-Pools und des datenbankspezifischen JDBC-Treibers werden in Abschnitt 4.1.1 beschrieben. Dies Klassenbibliotheken müssen sich im Klassenpfad des jeweiligen Moduls befinden, damit das *DataSource*-Objekt für das jeweilige Modul im JNDI-Verzeichnis konfiguriert werden kann. Dazu wird der Name der Klasse benötigt, die *DataSource*-Objekte erzeugen kann.. Von der konkreten Implementierung hängt ab, welche Initialisierungsparameter festgelegt werden können. Wichtig ist hier vor allem die Anzahl der im Connection-Pool zu verwaltenden Datenbankverbindungen. Für die Datenbankverbindungen muss der Name der Klasse, die das JDBC-Interface *java.sql.Driver* implementiert, sowie Informationen über die Datenbankverbindung (Benutzername, Passwort, Datenbank-URL), angegeben werden.

6.4. XSL-Stylesheets

Das Ergebnis der Analyse und Bewertung einer Query wird sowohl vom Datalog- als auch vom XQuery-Modul in einem speziellen XML-Dokument abgebildet. XSL-Dateien werden an diesem Punkt vor allem dafür eingesetzt, um aus solch einem Dokument HTML-Fragmente zu generieren, die in die HTML-Seite eingebettet werden, die dem Benutzer als Ergebnis seiner Anfrage präsentiert wird. Dies wird im folgenden Abschnitt erläutert. Darüber hinaus wird ein weiteres, spezielles XSL-Stylesheet im XQuery-Modul dazu verwendet, um ein

internes Zwischenergebnis der Analyse zweier XQuery-Ausdrücke zu modifizieren.

6.4.1. HTML Rendering

XSL-Stylesheets werden in erster Linie dazu verwendet, um Analyse- und Bewertungsergebnisse, die in Form eines XML-Dokumentes abgebildet werden, für die Präsentation als HTML-Seite aufzubereiten.

Das XSL-Stylesheet für das Datalog-Modul befindet sich in der folgenden Datei:

/etutor/resources/modules/datalog/xml/render-datalog.xsl

Die entsprechende Datei für das XQuery-Modul ist unter dem folgenden Pfad zu finden:

/etutor/resources/modules/xquery/xml/render-xquery.xsl

Mithilfe der XSL-Templates, die in diesen Stylesheets enthalten sind, werden die Analyse- und Bewertungsergebnisse von Queries so transformiert, dass sie in die HTML-Seite eingebettet werden können, die dem Benutzer präsentiert wird.

6.4.2. Modifikation der XQuery-Analyse

XSL-Stylesheets kommen außerdem speziell im XQuery-Modul zum Einsatz, um ein internes Zwischenergebnis der Analyse für die Darstellung für den Benutzer anzupassen (siehe Abschnitt 3.2.3). Die Transformationsregeln für die Modifikation des Zwischenergebnisses, das als XSL-Stylesheet vorliegt, sind wiederum in einem XSL-Stylesheet definiert, das unter dem folgenden Pfad zu finden ist:

/etutor/resources/modules/datalog/xml/render-datalog.xsl

7. Zusammenfassung

Es wurden zwei Module entwickelt, um in das eTutor-System integriert zu werden. Dieses soll die Möglichkeit bieten, Übungsaufgaben zu Themen auf dem Gebiet des *Data & Knowledge Engineering* interaktiv und online lösen zu können. Mit dem ersten Modul, das auf Datalog-Aufgaben ausgerichtet ist, können demzufolge Datalog-Regeln und –Fakten analysiert werden. Die Analyseergebnisse wiederum werden bewertet und für die Präsentation für den Benutzer aufbereitet. Mit dem zweiten Modul wurde analog dazu die Möglichkeit geschaffen, XQuery-Ausdrücke automatisch analysieren und bewerten zu lassen.

Es hat sich bei der Entwicklung herausgestellt, dass einige zusätzliche Angaben bei der Definition von Übungsaufgaben erforderlich sind, um die Details der Analyse festzulegen. Diese Informationen werden in einer Datenbank gespeichert und von dort bei jedem Aufruf der Analysefunktion des jeweiligen Moduls ausgelesen.

Die Analyse von Datalog-Regeln und XQuery-Ausdrücken konzentriert sich hauptsächlich auf die konkreten Auswertungsergebnisse dieser Abfragen, d.h. die Analyse basiert auf einem „heuristischen“ Verfahren. Vergleiche zwischen Datalog-Regeln bzw. XQuery-Ausdrücken selbst würden zwar genauere Aussagen über mögliche Fehlerquellen ermöglichen, allerdings würde dies eine weiterführende Syntaxanalyse erfordern, die beim Datalog- bzw. dem XQuery-Modul bereits durch die eingesetzten Tools bei der Auswertung von Abfragen erfolgt. Darüber hinaus muss bedacht werden, dass im allgemeinen die Möglichkeiten, auf verschiedene Art und Weise zum richtigen Ergebnis zu kommen, vielfältig sind.

Literaturverzeichnis

[Apac05] Apache Software Foundation. Commons DBCP. http://jakarta.apache.org/commons/dbcp/, Februar 2006.

[Apac06] Apache Software Foundation. Apache Tomcat. http://tomcat.apache.org/, Februar 2006.

[Bihl04] R. Bihlmeyer, W. Faber, G. Ielpa, V. Lio, G. Pfeifer. DLV - User Manual. http://www.dbai.tuwien.ac.at/proj/dlv/man/, November 2004.

[DDBE99] IBM alphaWorks. Data Descriptors by Example. http://www.alphaworks.ibm.com/tech/DDbE, Juni 1999.

[DLV04] The DLV Project - A Disjunctive Datalog System. http://www.dbai.tuwien.ac.at/proj/dlv/, November 2004.

[DOM04] dom4j. http://www.dom4j.org/, November 2004.

[Gott99] G. Gottlob et al. Expertensysteme. Springer Verlag, 1999.

[IPSI04] Fraunhofer IPSI Darmstadt. IPSI-XQ. http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/index_e.html, November 2004.

[Koch04] C. Koch. The DLV Tutorial. http://chkoch.home.cern.ch/chkoch/dlv/dlv_tutorial.html, November 2004.

[Log04] log4j. http://logging.apache.org/log4j/docs/index.html, November 2004.

[Loye04] Y. Loyer, U. Straccia. The Stable Model Semantics under the Any-World Assumption. http://dienst.iei.pi.cnr.it/Dienst/Repository/2.0/Body/ercim.cnr.isti/2004-TR-02/pdf, Februar 2004.

[Mahm03] Q. Mahmoud. Developing JNDI-based Applications. http://java.sun.com/developer/technicalArticles/Programming/jndi/, Februar 2006.

[Ora04] Oracle XML Developer's Kit 10g, http://www.oracle.com/technology/tech/xml/xdkhome.html, November 2004.

[Shif06] ShiftOne. ShiftOne OOC JNDI. http://oocjndi.sourceforge.net/index.html, Februar 2006.

Glossar

Fakt	Datalog-Elemente, aus denen sich ein Prädikat zusammensetzt und das mit einem Tupel einer Tabelle in relationalen Datenbanken vergleichbar ist. Ein Fakt besteht aus dem Namen des Prädikats, das dieses Fakt enthält, sowie aus einer Anzahl von Termen. Die Menge der Terme, die ein Fakt umfasst, kann auch leer sein.
Prädikat	Ein Datalog-Element, das mit dem Tabellenschema in relationalen Datenbanken vergleichbar ist. Die Bezeichnung *Relation* wird daher als Synonym verwendet. Ein Prädikat besteht aus einem oder mehreren Fakten und wird durch einen Namen identifiziert.
Property	Bezeichnung für ein Schlüssel/Wert-Paar, mit dem eine Eigenschaft festgelegt werden kann, die von einem Programm benötigt wird. Eine Property-Datei enthält eine Auflistung von solchen Schlüssel/Wert-Paaren.
Query	Dieser Begriff wird in dieser Arbeit gleichermaßen als Bezeichnung für ein XQuery-Statement verwendet, als auch für eine Anzahl zusammengehöriger Datalog-Regeln.
Term	Kleinste Datalog-Einheit, aus denen sich ein Fakt zusammensetzt.
URL	Uniform Resource Locator. Textuelle Repräsentation einer Ressource, die über das Internet verfügbar ist

Anhang

Anlegen der SQL-Tabellen für das Datalog-Modul

```
CREATE TABLE etutor_datalog.error_categories (
        name VARCHAR2(25) NOT NULL,
        id NUMBER NOT NULL,
        CONSTRAINT error_cat_id PRIMARY KEY(id));

CREATE TABLE etutor_datalog.error_grading_group (
        id NUMBER NOT NULL,
        CONSTRAINT group_grad_id PRIMARY KEY(id));

CREATE TABLE etutor_datalog.error_gradings (
        grading_group NUMBER NOT NULL,
        grading_level NUMBER NOT NULL,
        grading_category NUMBER NOT NULL,
        minus_points NUMBER(4, 1) NOT NULL,
        CONSTRAINT grading_id FOREIGN KEY(grading_group)
                REFERENCES etutor_datalog.error_grading_group(id)
                ON DELETE CASCADE),
        CONSTRAINT grading_cat_id FOREIGN KEY(grading_category)
                REFERENCES etutor_datalog.error_categories(id),
        CONSTRAINT grading_grad_unique UNIQUE(grading_group,
                grading_category));

CREATE TABLE etutor_datalog.facts (
        facts VARCHAR2(1500) NOT NULL,
        id NUMBER NOT NULL,
        name VARCHAR2(20) NOT NULL,
        CONSTRAINT fact_id PRIMARY KEY(id));

CREATE TABLE etutor_datalog.predicates (
        name VARCHAR2(35) NOT NULL,
        exercise NUMBER(10) NOT NULL,
        CONSTRAINT predicates_fk_exercise FOREIGN KEY(exercise)
                REFERENCES etutor_datalog.exercise(id)
                ON DELETE CASCADE);

CREATE TABLE etutor_datalog.unchecked_terms (
        predicate VARCHAR2(35) NOT NULL,
        term VARCHAR2(35 byte) NOT NULL,
        position NUMBER(10) NOT NULL,
        exercise NUMBER(10) NOT NULL,
        CONSTRAINT unchecked_exercise_id FOREIGN KEY(exercise)
                REFERENCES etutor_datalog.exercise(id));
```

```
CREATE TABLE etutor_datalog.exercise (
        id NUMBER NOT NULL,
        query VARCHAR2(1500) NOT NULL,
        facts NUMBER NOT NULL,
        gradings NUMBER,
        points NUMBER(4, 1),
        CONSTRAINT ex_id PRIMARY KEY(id),
        CONSTRAINT exercise_fact_id FOREIGN KEY(facts)
                REFERENCES etutor_datalog.facts(id),
        CONSTRAINT exercise_grad_id FOREIGN KEY(gradings)
                REFERENCES etutor_datalog.error_grading_group(id));
```

Anlegen der SQL-Tabellen für das XQuery-Modul

```
CREATE TABLE etutor_xquery.error_categories (
        name VARCHAR2(25) NOT NULL,
        id NUMBER NOT NULL,
        CONSTRAINT cat_id PRIMARY KEY(id));

CREATE TABLE etutor_xquery.error_grading_group (
        id NUMBER NOT NULL,
        CONSTRAINT grad_group_id PRIMARY KEY(id));

CREATE TABLE etutor_xquery.error_gradings (
        grading_group NUMBER NOT NULL,
        grading_level NUMBER NOT NULL,
        grading_category NUMBER NOT NULL,
        minus_points NUMBER(4, 1) NOT NULL,
        CONSTRAINT cat_grad_id FOREIGN KEY(grading_category)
                REFERENCES etutor_xquery.error_categories(id),
        CONSTRAINT grad_id FOREIGN KEY(grading_group)
                REFERENCES etutor_xquery.error_grading_group(id)
                ON DELETE CASCADE,
        CONSTRAINT grad_unique UNIQUE(grading_group,
                grading_category));

CREATE TABLE etutor_xquery.exercise_urls (
        url VARCHAR2(100) NOT NULL,
        hidden_url VARCHAR2(100) NOT NULL,
        exercise NUMBER(10) NOT NULL,
        CONSTRAINT exercise_urls_fk_exercise FOREIGN KEY(exercise)
                REFERENCES etutor_xquery.exercise(id)
                ON DELETE CASCADE);

CREATE TABLE etutor_xquery.sortings (
        xpath VARCHAR2(50) NOT NULL,
        exercise NUMBER(10) NOT NULL,
        CONSTRAINT sortings_fk_exercise FOREIGN KEY(exercise)
                REFERENCES etutor_xquery.exercise(id)
                ON DELETE CASCADE);

CREATE TABLE etutor_xquery.xmldocs (
        id NUMBER(10) NOT NULL,
        doc SYS.XMLTYPE NOT NULL,
        filename VARCHAR2(20) NOT NULL,
        CONSTRAINT xmldoc_id PRIMARY KEY(id));
```

```
CREATE TABLE etutor_xquery.exercise (
        id NUMBER NOT NULL,
        query VARCHAR2(1000) NOT NULL,
        gradings NUMBER,
        points NUMBER(4, 1),
        CONSTRAINT ex_grad_id FOREIGN KEY(gradings)
                REFERENCES etutor_xquery.error_grading_group(id),
        CONSTRAINT ex_id PRIMARY KEY(id));
```