1 Zielstellung des Projekts

Ziel des Projekts ist eine interaktiv web-basierte Lernumgebung zu entwickeln, die den Studierenden während der Selbstlernphase Hilfestellung beim Lösen von einfachen Programmier- und Spezifikationsaufgaben geben sollen. Das System soll sowohl im Rahmen von universitären Lehrveranstaltungen zur Bearbeitung von Übungsaufgaben genutzt werden, als auch exploratives Lernen unterstützt und fördert. Das System soll eher die Rolle eines Begleiters und weniger als Tutor spielen. Für diesen Zweck wird die Logikprogrammierung als Anwendungsszenario gewählt. Wir möchten zwei Teilprobleme der Logikprogrammierung untersuchen und Prototypen dafür entwickeln.

2 Projektplan und -Stand

2.1 Überblick

Das Projekt beginnt seit 01.11.2004 und wir versuchen, folgende Meilensteine zu erreichen:

- 01.11.2004 Bereitstellung einer Infrastruktur für eine interaktive Lernumgebung
- 01.05.2004 Erster Prototyp für Datenbankabfragen.
- 01.11.2004
  Vorbereitung für eine umfangreichere Evaluation mit Studierenden im Grundstudium.
  Prototyp für rekursive Prädikatsspezifikationen.
  Pilotuntersuchungen zur Übertragung der Diagnoseprinzipien auf Modellierungsaufgabe.
- 01.05.2005
  Evaluation in Kooperation mit der HAW.
  Portierungsversuch für das Anwendungsdomain UML.
- 31.08.2005: Projektdokumentationen.

Der Projektplan wurde bis zum heutigen Zeitpunkt eingehalten. Wir haben einen ersten Prototyp für die Datenbankabfragenaufgaben entwickelt. Dieser ist von einem Student im 6. Semester evaluiert worden und der Bericht für diese Evaluation liegt vor (`https://nats-www.informatik.uni-hamburg.de/view/INCOM/Dokumentation`).

Die Entwicklung eines Prototyps zur Lösung für rekursive Prädikatsspezifikationen läuft zur Zeit und parallel zu dieser Entwicklung wird es untersucht, ob die Diagnoseprinzipien auf Modellierungsaufgaben mit UML übertragen werden können.

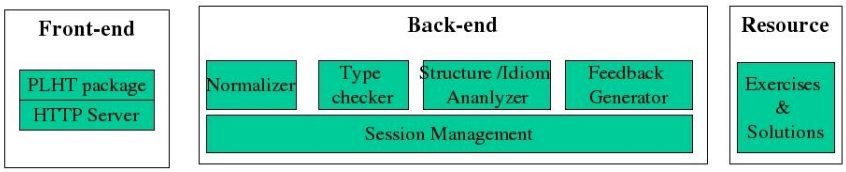


Figure 1: INCOM Architektur

2.2 Architektur und Infrastruktur der Lernumgebung für INCOM

Die Lernumgebung für INCOM ist ein Client-Server-System, durch das die lokale Arbeitsumbegung des Studenten mit einem zentralen Server verbunden wird. Der Client ist als dynamische HTML-Seiten realisiert, der Student kann dadurch auf die Lernumgebung Browsern zugreifen.

2.2 stellt die drei-Schichten-Architektur des Lernsystems dar. Die Präsentationsschicht dient zur Darstellung der Aufgabenbeschreibung, zum Eingeben der studentischen Lösung, und zur Vermittlung der Hinweise zur Lösungsverbesserung. Die Applikationslogikschicht verarbeitet die Benutzereingabe, analysiert sie anhand von Constraints, die von verschiedenen Wissensquellen extrahiert werden. Auf der Ressourcenschicht werden Wissensquellen wie Aufgabenbeschreibung, Musterlösungen, Patterns und Typdeklarationen gehalten.

Die Präsentationsschicht wird durch einen HTTP-Package von SWI-Prolog und eine PLHT-Bibliothek unterstützt. Das HTTP-Package ermöglicht uns, HTTP-Anfragen auf einem XPCE-Webserver zu verarbeiten. PLHT-Bibliothek stellt Techniken zur Verfügung, Aufrufe der Prolog-Prädikate auszuführen und dynamische HTML-Seiten zu generieren. Die Applikationslogikschicht besteht aus in Prolog geschriebene Module, die in den folgenden Abschnitten genauer beschrieben werden. Die Ressourcenschicht enthält Beschreibungen für Aufgabenstellung und deren Musterlösung, die in XML-Format gehalten werden.

2.3 Prototyp für Datenbankabfragen

In diesem Prototyp besteht die Benutzerschnittstelle für Studenten aus drei Bereiche. Der erste ist für die Aufgabenformulierung. Der zweite ist zum Eingeben der Lösungen und der letztere ist ein Platz für die Tipps, die vom Diagnosesystem zurückgegeben werden.

Die Eingabe des Student wird an die Applikationsschicht übergeben und wird durch die Komponente ***Normalizer*** in eine interne Darstellung transformiert. Das hat den Zweck, dass man die Eingabe als eine einheitliche Datenstruktur hat, um sie analysieren zu können. Die Diagnose durchläuft zwei Analyseschritte. Der erste ist semantische und der letztere ist aufgabenstellungsspezi-

fische Analyse. Die semantische Analyse ist durch den *TypeChecker* befürwortet. Der *StructureAnalyzer* durchführt den zweiten Analyseschritt und basiert auf dem Ansatz, die Strukturunterschiede zwischen der studentischen und der Musterlösung zu identifizieren. Im Prototyp für Datenbankabfragen wird die studentische Lösung nach folgenden Diagnoseinformationen gesucht:

- Typkonflikte
- Verwendung redundanter Unifikationsgleichungen
- Instanzzierung von Argumentpositionen
- Koreferenz
- Tippfehler in Prädikatsnamen
- Fehlende bzw. überflüssige Teilziele
- fehlerhafte Teilzielreihenfolge
- fehlende bzw. überflüssige Argumente
- fehlerhafte Argumentreihenfolge

Die o.g. Diagnoseinformationen sind durch Verletzung der Constraints hervorgerufen, die in den Analysekomponenten definiert sind und aus verschiedenen Wissensquellen wie Typdeklarationen, Musterdatenbank und Musterlösungen extrahiert werden.

Diagnoseinformationen, die durch Analyse gewonnen werden, werden an einen sog. *FeebackGenerator* übergeben. Die Feedbackgenerierung liefert zur Zeit Feedbacks mit drei Informationen: (1)wo ist der Fehler, (2) welche Art von Fehler liegt vor und (3) wie kann man ihn beseitigen. Diese Informationen sollen in der nächsten Version nacheinander zurückgegeben werden.

2.4 Prototyp für rekursive Prädikatsdefinitionen

Ein Prototyp für rekursive Prädikatsdefinitionen befindet sich in Entwicklung. Die Benutzerschnittstelle für Lernende in diesem Prototyp ist genauso strukturiert wie in dem Prototyp für Datebankabfragenaufgaben.

Ein Unterschied zwischen dem Prototyp für Datebankabfragen und diesem liegt in dem zweiten Analyseschritt. Hier verfolgen wir einen sog. Pattern-basierten Ansatz zur Identifizierung der Fehler in studentischer Lösung.

Wenn eine neue rekursive Aufgabenstellung definiert werden sollte, werden alle möglichen Lösungspatterns, die zum Lösen dieser Aufgabe verwendet werden können, in die Aufgabendatenbank eingetragen. Zu jedem Pattern wird eine

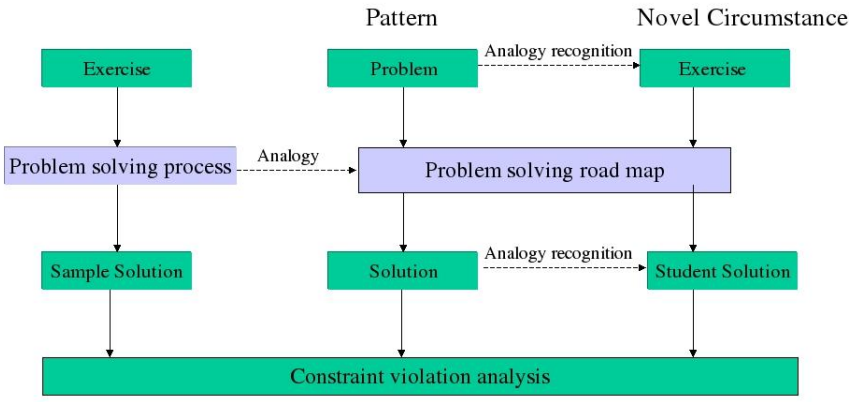


Figure 2: Pattern-basierte Diagnose

Musterlösung definiert. Ein Pattern beschreibt den Lösungsraum eines Problems, somit kann eine Musterlösung als eine spezielle Form eines Patterns betrachtet werden. Zuerst mit Hilfe eines Pattern-Erkenners wird es bestimmt, welchen Ansatz der Student gerade verfolgt. Zu einem Pattern werden sämtliche Constraints definiert, die dieses Pattern beschreiben. Die Musterlösung, die zu dem gefundenen Pattern gehört, beschreibt aufgabenspezifische Constraints . Der *IdiomAnalyzer* wird Constraint-Verletzungen in der studentischen Lösung diagnotizieren.

Zur Zeit leistet dieser Prototyp folgende Diagnose:

- fehlender bzw. überflüssiger Rekursionsabschluss
- fehlender Rekursionsschritt
- falsche Teilzielreihenfolge
- fehlerhafte Listenkomposition/-dekomposition
- fehlerhafte Argumentverwendung

3 Offene Probleme

Im Prinzip ist der für Datenbankabfragen angewandte Ansatz zum Vergleich einer studentischen und einer Musterlösung auf die Datenbankabfragesprache SQL übertragbar. Dies erfordert weitere Untersuchung.

Der für den Prototyp für rekursive Prädikatsspezifikationen eingesetzte Pattern-basierte Ansatz sollte zur Ableitung einer generalisierten Lösung erweitert werden, dass Die Portierung dieses Prototyps auf zusätzliche Aufgabenbiete wird.

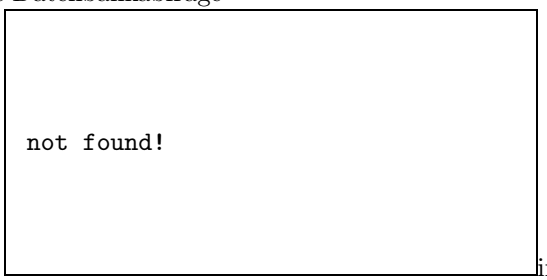
Die Modellierungssprache UML ist ein potentiales attraktives Anwendungsgebiet, das wir demnächst in Kooperation mit der TUHH untersuchen möchten.

4 Weitere Aufgaben

Zur Zeit liegt der Schwerpunkt unserer Arbeit bei der Fortsetzung der Entwicklung eines Prototyps für rekursive Prädikatsdefinitionen. Dabei verfolgen wir einen sog. Pattern-basierten Ansatz, der auch in anderen Anwendungsgebieten verwendet werden sollte.

Die Feedbackgenerator und die Benuterschnittstelle müssen erweitert werden, so dass der Lernende den grad der Unterstützung durch das System selbst bestimmen können und nicht fortlaufend vom System geschulmeistert werden.

Als nächstes werden wir eine Evaluation des Systems für die Datenbankabfrage-



naufgaben vorbereiten. Diese Evaluation ist für Studierende im Wintersemester 2005 geplant.

Zur Zeit müssen wir neue Aufgabenstellung, Musterlösung und Musterdatenbank direkt in XML oder Prolog-Code eingeben müssen. Eine Autorenumgebung zum Beschreiben neuer Aufgaben ist wünschenswert.

Zur Zeit erlaubt PLHT-Bibliothek keine thread-sichere Ausführung. Deshalb kann sie nicht mit einem multi-threading Webserver verwendet werden. Um das System öffentlich zugänglich machen zu können, müssen wir die PLHT-Bibliothek verbessern, so dass Prolog-Aufrufe Thread-sicher ausgeführt werden können.