Der Virtuelle Campus in Osnabrück

Helmar Gust   Christoph Peylo   Claus Rollinger
Wilfried Teiken

April 1999

1 Einleitung

Vorliegender Aufsatz geht auf einen Vortrag zurück, der während des Workshops „Lehr-Lern-Systeme im Virtueller Campus“ (10. - 12.3.1999) in Goslar gehalten wurde. Im folgenden wird das Projektmodul einer netzbasierten tutoriellen Prolog-Umgebung, das am Institut für semantische Informationsverarbeitung an der Universität Osnabrück entwickelt wurde, vorgestellt sowie seine Aufgabe und Zielsetzung motiviert. Vorangestellt wird ein kurzer Überblick über Entstehung und Entwicklung „intelligenter Lehr- und Lernumgebungen“, um die eigentliche Herkunft und Ausrichtung dieses Bereichs zu verdeutlichen und das Projektmodul innerhalb des Forschungsbereichs „intelligente tutorielle Systeme“ (ITS) einordnen zu können.

Die begriffliche Fassung dessen, was intelligente Lehr- und Lernumgebungen eigentlich darstellen sollen, wird von zwei Seiten vorgenommen: zunächst anhand didaktischer Leistungen, die ein Tutor – unabhängig ob Mensch oder Maschine – mindestens erbringen muß, um das Attribut „intelligent“ für sich reklamieren zu können, ferner aus der Perspektive des Forschungsbereichs selbst. Dann werden verschiedene Vorgehensweisen und Strategien solcher Systeme vorgestellt und diskutiert. Vor diesem Hintergrund wird im letzen Teil der Beitrag des Instituts für Semantische Informationsverarbeitung, der VC-Prolog-Tutor, dargestellt. Da der Workshop zur Halbzeit des Projekts stattfand, wird im wesentlichen auf die bisherige Entwicklung und die bis dorthin realisierte Funktionalität des VC-Prolog-Tutors eingegangen und stellt somit nur eine Momentaufnahme dar. Die Entwicklung des Systems wird in den nächsten anderthalb Jahren weitergeführt.

2 Motivation intelligenter Lehr- und Lernumgebungen

Die Idee, Maschinen zum Lehren und Lernen einzusetzen, ist vor dem Hintergrund der Entwicklung psychologischer Theorien zu sehen, die die Verbindung von Inhalten des Bewußtseins durch Assoziation als wichtigstes Erklärungsprinzip annahmen. Die ersten Ansätze waren daher stark von der behavioristischen Lerntheorie und der Skinner'schen Variante, der operanten Konditionierung (Skinner, 1968), beeinflußt. Aus diesem Modell der Verhaltens und Lernprozesse heraus entwickelte sich die „Programmierte Instruktion“. Die Lernmaschinen der 50er und 60er Jahre präsentierten auf einer Art Microfiche-Bildschirm Texte und stellten Fragen zu den Texten, die von den Lernenden über eine Spezialtastatur mit einer geringen Anzahl von Knöpfen beantwortet wurden (Schulmeister, 1997, S. 97).

Brunner (Brunner, 1966) markiert den Wandel von den behavioristischen Lerntheorien zu den Instruktionstheorien (vgl. Seels (1989) und Schulmeister (1997)). Die Absicht der Instruktionstheorie lag im wesentlichen darin, die Atomisierung des Lernens durch Stimulus-Response-Verbindungen zu vermeiden und die Planung computerunterstützter Lehreinheiten durch verschiedene Methoden variabel zu gestalten.

Daneben bildete sich eine weitere Richtung, in der Computer zur Erforschung und Modellierung kognitiver Fähigkeiten eingesetzt wurden (ein Überblick bei Park et al. (1987), S. 11). Das Spektrum dieses Ansatzes reicht von der Modellierung kognitiver Leistungen des Menschen zur Validierung von Theorien (beispielsweise Schank und Abelson (1977)) bis zur Verwendung des Computers als Katalysator zur Entwicklung kognitiver Fähigkeiten (z.B. Papert (1980)). Die Möglichkeit, Computer zur Erforschung menschlicher Kognition einzusetzen, motivierte Wissenschaftler aus dem kognitionswissenschaftlichen Bereich und dem der künstlichen Intelligenz, sich mit dieser Thematik zu beschäftigen (Wenger, 1987). Das Interesse der KI und Kognitionswissenschaft ist zum einen erkentnistheoretisch motivert. Die Modellierung von Lehrtechniken soll Aspekte des Lernens, wie Motivation und Performanz verstehen helfen. Zum anderen kann der Bereich der Lehre und Wissensvermittlung als Anwendungsgebiet für klassische Gebiete der KI, wie Wissensrepräsentation, aufgefaßt und genutzt werden (Woolf, 1992).

Der Komplex KI und Unterricht hat sich bereits seit Jahren als Forschungsbereich – mit einer Fülle von Spezialisierungen – etabliert und KI-Technologien und Arbeitsweisen sind in eine Vielzahl von Systemen eingeflossen. Im weiteren sollen vor allem die Systeme näher in Betracht gezogen werden, die sich mit dem Erlernen von Programmiersprachen, oder zumindest der Vermittlung von Kenntnissen aus dem Bereich der Programmierung befassen, da der VC-Prolog-Tutor in der Tradition dieser Systeme zu sehen ist.

3 Anforderungen an intelligente Lehr- und Lernumgebungen

An dieser Stelle soll zunächst nicht gefragt werden, wie intelligente Lehr- und Lernumgebungen normalerweise, d.h. im Bereich der Forschungsrichtung, die sich damit beschäftigt, definiert werden, sondern grundsätzlich der Frage nachgegangen werden, welche Merkmale ein System aufweisen sollte, damit es tatsächlich als „intelligentes“ Lehrsystem bezeichnet werden kann.

3.1 Didaktische Anforderungen

Im wesentlichen sollten alle Leistungen erwartet werden können, die man von einem menschlichen Lehrer ebenfalls erwarten würde. Dabei handelt es sich zunächst um die Art und Weise wie der Lehrstoff organisiert und präsentiert wird. Dies muß einerseits dem Lehrstoff angemessen sein, sich aber auch an dem allgemeinen Kenntnisstand der Lernenden orientieren. Daneben müssen die Lernenden beim Wissenserwerb unterstützt und angeleitet werden. Auch dies beinhaltet eine ganze Reihe einzelner Aufgabenfelder, wie Darbietung verschiedener Übungsmöglichkeiten, Beantwortung auftretender Fragen, Hilfestellung bei der Bearbeitung von Aufgaben, etc. Schließlich erwartet man von einem Tutor, daß er aus seinen Erfahrungen mit der Rezeption einer Lehreinheit den Lernfortschritt und Kenntnisstand der jeweiligen Lernenden beurteilen kann und – im Idealfall – dieses Wissen bei der weiteren Auswahl und Präsentation von Lehrinhalten berücksichtigt.

Die genannten Punkte lassen sich für ein Anforderungsprofil an intelligente Lehr und Lernumgebungen hinsichtlich grundlegender didaktischer Fähigkeiten sowie der Präsentation von Wissen und tutorieller Strategien verwenden.

3.2 Aufbau und Struktur

Im folgenden sollen diese eher introspektiv gewonnenen Kriterien, die an Intelligente Lehr- und Lernumgebungen anzulegen sind, mit den in diesem Bereich üblichen kontrastiert und verglichen werden. Daß es keine präzise Terminologie und „harte“ Definitionen gibt, ist in Forschungsgebieten, in dem sich Angehörige verschiedener Disziplinen mit verschiedenen Zielen und Ausrichtungen betätigen, eher die Regel als die Ausnahme. So sind die Grenzen zwischen *Computer Based Instruction* (CBI) und *Computer Aided Instruction* einerseits und *Intelligent Computer Aided Instruction* (ICAI), *Intelligent Tutoring Systems* (ITS) und *Intelligent Learning Environment* andererseits unscharf.

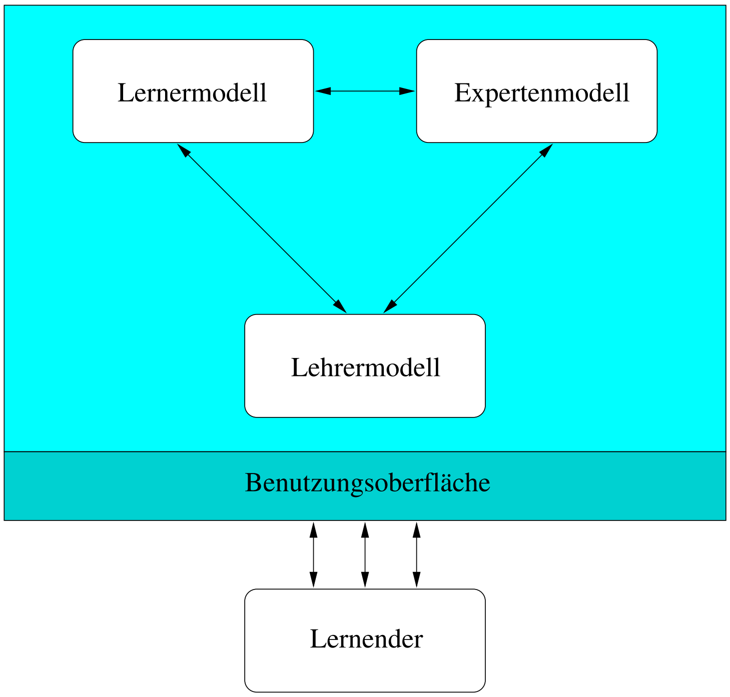

Abbildung 1: Klassische Architektur eines ITS. Abbildung nach Schröder (1996),S. 290

Ohne an dieser Stelle die Schwierigkeit geeigneter Abgrenzungskriterien weiter vertiefen zu wollen (hierzu Park et al. (1987)), wird im weiteren betrachtet, wie Aufbau und Struktur eines Systems dessen intelligentes Verhalten ermöglicht und unterstützt. Beispielsweise sind CAI- und CBI-Systeme in der Regel skriptähnlich aufgebaut: alle Antworten des Rechners, Texte oder Graphiken werden in Bildschirmseiten organisiert und Verzweigungsregeln definieren Thema und Ausgabe. Das bedeutet, daß das System über sein Wissen nicht inferieren kann, sondern jede Antwort und jeder Pfad vom Autor explizit angelegt und hart kodiert ist (Woolf, 1992, S. 435), (Park et al., 1987, S. 31). Ein intelligentes System muß hingegen die Möglichkeit besitzen, auf der Grundlage seines Wissens Schlußfolgerungen ziehen zu können. Allgemeiner ausgedrückt: intelligentes Verhalten in einem Lehrkontext setzt die Repräsentation des Domänenwissens in einer Form voraus, mit der das System auf verschiedenen Ebenen (z.B. Präsentation von Lehreinheiten, Übungszusammenstellung, Fehlerdiagnose etc.) umgehen kann. Hierzu Wenger (Wenger, 1987, S. 5):

> [...] it is the knowledge itself that is explicitly represented so that it can be used in computer-based systems. It is then the responsibility of programs to compose instructional interactions dynamically, making decisions by reference to the knowledge with which they have been provided.

Im folgenden wird die Sichtweise zugrunde gelegt, daß ein ITS die eigentliche Aufgabe hat, einem Lerner die Erarbeitung eines Gegenstandsbereichs ohne weitere Hilfsmittel zu ermöglichen (Schröder (1996), Anderson et al. (1990)). Dies soll durch ein Zusammenspiel mehrerer Module, die das Lernermodell, das Expertenmodell und das Lehrermodell enthalten sowie einer weiteren Komponente, die die Kommunikation des Lernenden mit dem System regelt, erreicht werden (Winkels und Breuker (1992), Schulmeister (1997), vgl. Abbildung 1). Ein intelligentes Tutorsystem besitzt somit grundsätzlich die erforderlichen Komponenten, um den didaktischen Anforderungen, wie sie im letzten Abschnitt (3.1) definiert wurden, gerecht werden zu können. Daraus ergibt sich die klassische Architektur eines ITS, wie sie in Abbildung

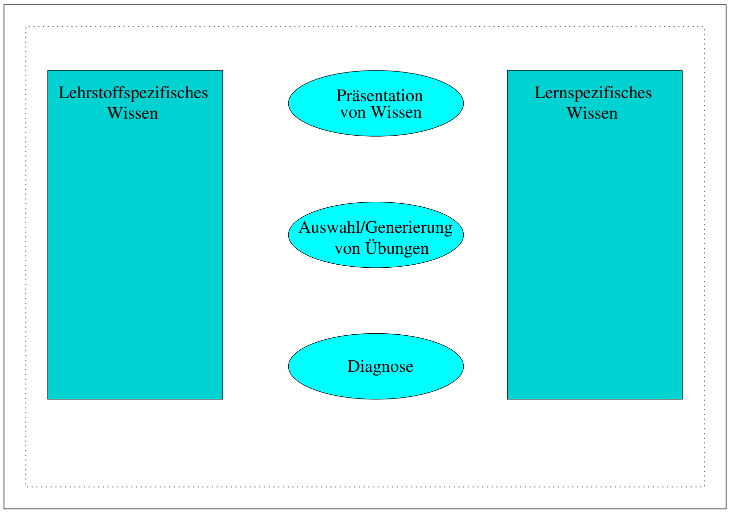

Abbildung 2: Allgemeine Architektur für Lehr- und Lernsysteme. Unterschieden wird zwischen den Datenbeständen (lehrstoff- und lernspezifisches Wissen) und den Prozessen (durch Ovale dargestellt), die auf diesen Daten operieren.

1 dargestellt ist. Der Nachteil dieser Architekturdarstellung (Abbildung 1)

ist offensichtlich: Zwischen den einzelnen Modellen soll irgendeine Form der Kommunikation stattfinden aus der Skizze geht aber weder hervor wie diese aussehen könnte noch wie das, was dem Lernenden präsentiert wird, mit den einzelnen Modellen zusammenhängt. Darüberhinaus wäre es wünschenswert, eine allgemeinere Architektur zur Verfügung zu haben, die nicht nur intelligenten Lehrsystemen, sondern auch intelligenten Lernumgebungen zugrunde gelegt werden könnte. [1] Wir schlagen daher eine allgemeinere Basisarchitektur (siehe Abbildung 2) vor, mit der sich alle Systeme in diesem Bereich darstellen lassen. Bei dieser allgemeinenen Architektur wird zwischen zwei Wissensbeständen unterschieden: dem lehrstoff- und dem lernspezifischen Wissen. Das lehrstoffpezifische Wissen beinhaltet das Wissen, das dem Lernenden vermittelt werden soll sowie das fachspezifische Wissen, das für die Überprüfung des Lernerfolges relevant ist. Um dies an einem Beispiel zu demonstrieren: beim VC-Prolog-Tutor (siehe Abbildung 3, Seite 10) umfaßt das lehrstoffspezifische Wissen ein Domänenmodell, d.h. konkretes Wissen über Prolog, Übungsaufgaben und ein Anwendungsdomänenmodell, das das notwendige Hintergrundwissen beinhaltet, das benötigt wird, um die Übungsaufgaben zu korrigieren.[2] Das lernspezifische Wissen umfaßt Wissen über die Vermittlung von Wissen. Dies umfaßt allgemeineres Wissen, wie Kenntnis verschiedener Lehrstrategien, Diagnosewissen und kann auch sehr spezielle Informationen, wie Modelle der einzelnen Benutzer enthalten. Auch hier kann als Beispiel wieder auf den VC-Prolog-Tutor in Abbildung 3 verwiesen werden.

[1] Der Unterschied zwischen einem ITS und einer intelligenten Lernumgebung wird an der explizit tutoriellen Ausrichtung eines intelligenten Tutorsytems festgemacht: d.h. im Gegensatz zu einem ITS besitzt eine intelligente Lernumgebung keine explizite tutorielle Komponente, d.h. kein eigentliches Lehrermodell (Schröder, 1996). Dahinter steht der Ansatz des „explorativen“ Lernens. Der Lernende soll nicht von einem Tutormodul ständig überwacht werden und sich nur auf den vorgegebenen Lösungswegen bewegen, sondern sich die Domäne selbst erschließen können. Tutorsysteme, die dem Lernenden wenig eigenen Spielraum lassen, können Abweichungen des Lerners vom vorgegebenen Lösungsweg schnell diagnostizieren und dem Lernenden einen entsprechenden Hinweis geben (z.B. Anderson et al. (1990)). In dem Maße, in dem den Lernenden mehr Freiheiten eingeräumt werden, erhöht sich die Wahrscheinlichkeit nicht antizipierter Fehler. Der Verzicht auf ein explizites Lehrermodell erfordert somit eine wesentlich elaboriertere Fehlerdiagnose.

[2] Beispielsweise muß das System, um eine Sortieraufgabe korrigieren zu können, Wissen darüber besitzen, wie sich Dinge in der Anwendungsdomäne anordnen lassen. Wenn Zahlen zu sortieren sind, ist dafür anderes Wissen erforderlich als zum Sortieren von Bauklötzen.

4 Aufgaben intelligenter Lehr- und Lernumgebungen

4.1 Analyse von Problemlöseaufgaben

Programmieraufgaben lassen sich ganz allgemein dem Bereich der Problemlöseaufgaben zuordnen. Hinsichtlich der Analyse der Problemlösekompetenz eines Lernenden können grundsätzlich zwei Strategien unterschieden werden: erstens die Analyse von Lösungen bzw. Teillösungen und zweitens die Analyse des Problemlöseprozesses.

Beide Vorgehensweisen werden bei Systemen eingesetzt, die sich mit dem Lehren von Programmiersprachen oder der Vermittlung von Teilaspekten aus diesem Bereich beschäftigen.

Beispielsweise wurde die erste Strategie bei LAURA (Adam und Laurent, 1980), dem MENO-II-Projekt (Wenger, 1987, S. 236) und dem Prolog-Tutor von Hong (Hong, 1997) angewendet. Hingegen wurde beispielsweise bei BRIDGE (Bonar und Solloway, 1985), PROUST (Johnson, 1990) und einem LISP-Tutor (Anderson et al., 1990) die zweite Strategie zugrunde gelegt. Daneben gibt es weitere Ansätze, die eine Kombination beider Vorgehensweisen verwenden (z.B. Weber (1994), Specht und Weber (1997)).

Die Analyse von Lösungen basiert auf dem Ansatz die vom Lerner produzierte Lösung mit einer oder mehreren korrekten Lösungen, sogenannten Musterlösungen, zu vergleichen und die semantisch relevanten Divergenzen zu erklären.[3] Der Vergleich kann entweder direkt auf dem Code des Lerners stattfinden (wie bei MENO-II) oder auf ziemlich abstrakter Ebene durchgeführt werden. So wurden beispielsweise in LAURA die zu vergleichenden Programme in Graphen transformiert, die durch verschiedene Operationen ineinander überführt werden sollten (Adam und Laurent, 1980, S. 83 ff). Bei prozeduralen Programmiersprachen gibt es allerdings so viele Möglichkeiten, Lösungen zu produzieren, daß ohne zusätzliches Wissen über Vorgehen und Anwendungsdomäne eine zuverlässige automatische Korrektur praktisch ausgeschlossen ist. Durch ihre Nähe zur Logik ist bei deklarativen Programmiersprachen wie Prolog die Transformation verschiedener – aber semantisch gleicher – Programme ineinander einfacher möglich.

Bei der Überwachung des Problemlöseprozesses wird davon ausgegangen, daß der Benutzer mit der Bearbeiten einer Programmieraufgabe einen Plan verfolgt, der vom System erkannt und in seiner Ausführung verfolgt werden

[3] Jede Vertauschung von Operationen kann semantisch relevant sein. Hier muß jeder Einzelfall analysiert werden. Diese Analyse ist bei prozeduralen Sprachen nicht in jedem Falle leistbar.

kann. Dieses Wissen kann zur Fehlererklärung sinnvoll eingesetzt werden, um einem Lernenden erklären zu können *warum* er etwas falsch gemacht hat. Die zur Diagnose erforderlichen Fähigkeiten des Systems liegen dann vor allem im Bereich der Planerkennung und Prozeßüberwachung (Winkels und Breuker, 1992). Planen und Planerkennung ist allerdings, im Rahmen nicht-trivialer Domänen, von hoher Komplexität. Bei Aufgaben, die auf unterschiedlichen Wegen gelöst werden können, ist dieses Vorgehen problematisch, da der Suchraum zur Planerkennung durch die steigende Anzahl alternativer Handlungsmöglichkeiten, die berücksichtigt werden müssen, förmlich explodiert (Yang, 1997). Dieses Vorgehen bietet sich somit entweder bei überschaubaren Suchräumen an, oder wenn aufgrund von Anleitungen des Benutzers oder der Integration zusätzlichen Wissens der Suchraum beschnitten werden kann. Beispielsweise basiert der LISP-Tutor von Anderson et al. (1990), auf konkreten Annahmen darüber, wie ein idealer Lösungsweg für eine gestellte Aufgabe aussieht. Abweichungen des Benutzers von einem antizipierten Lösungsplan, führen zu sofortigen Rückmeldungen des Systems. Hier wird die Planerkennung im wesentlichen auf die Überwachung antizipierter Pläne reduziert, was von einem Benutzer als Gängelung durch das Systems empfunden wird, wenn er eine dem System unbekannte Lösung zu realisieren versucht. Wenn zusätzliche Daten über Arbeitsweisen und Präferenzen eines Lerners vorliegen, kann durch die Integration von Techniken aus dem maschinellen Lernen eine Zuordnung von Situationen zu möglichen Handlungssträngen erleichtert werden (Bauer, 1996). Dieses Vorgehen ist aber nur dann möglich, wenn schon Daten über den Benutzer und seine üblichen Handlungsweisen vorliegen. Bei der Bearbeitung von neuen Aufgaben, wie es bei dem Erlernen von Programmierfähigkeiten in der Regel der Fall ist, liegen diese Daten daher meistens nicht vor.

4.2 Fehlerdiagnose und Korrektur

Unabhängig davon, ob nur das Ergebnis des Problemlöseprozesses oder der Prozeß selbst vom System analysiert werden, müssen die Fehler des Lernenden gefunden und in einer Weise erklärt werden, die ein Lernen aus den Fehlern ermöglicht. Fehlerbehandlung ist somit ein zentraler Punkt jedes Systems im Bereich von intelligenten Lehr- und Lernumgebungen. Wenn ein Fehler vorliegt, lassen sich mehrere Fälle unterscheiden:

- Der Fehler kann vom System eigenständig entdeckt und erklärt werden. Dies setzt voraus, das das System Wissen über die korrekte Lösung, die Domäne sowie eine gewisse Analysefähigkeit besitzt, die es ihm ermöglicht, Differenzen zwischen einer korrekten Lösung und der vom

Lerner produzierten Lösung zu finden und zu erklären.

- Es handelt sich um einen antizipierten Fehler. Typische Fehler werden häufig in Fehlerbibliotheken oder „Buglisten“ hart kodiert. Das System muß den Fehler dann nur bei der Analyse der Lösung erkennen.
- Es handelt sich weder um einen antizipierten Fehler noch um einen Fehler, den das System selbst erkennen kann. Es kann nicht einmal genau entscheiden ob ein oder mehrere Fehler vorliegen.
- Es tritt ein Fall ein, in dem das System weder entscheiden kann ob ein Fehler vorliegt oder nicht.

Der dritte und vierte Fall sind eine Art *worst case*, mit ihnen muß in komplexeren Problemlösedomänen immer gerechnet werden. Ein System muß beispielsweise in der Lage sein, auch im Falle von mehreren Fehlern Aussagen über den einzelnen Fehler und den gegebenenfalls bestehenden Zusammenhang zwischen den Fehlern machen zu können.

5 Das VC-Projektmodul in Osnabrück

An der Universität Osnabrück wird im Rahmen des Virtuellen-Campus-Projekts ein interaktives Prolog-Tutor-System entwickelt, das im Rahmen von Lehrveranstaltungen zum Logischen Programmieren eingesetzt wird. Im Umfeld des Institut für Semantische Informationsverarbeitung wurden bereits mehrere Lehr- und Lernsysteme entwickelt, wobei intelligente Fehlerbehandlung jeweils im Zentrum stand (z.B. Gust et al. (1993), Krüger (1995), Bongardt et al. (1997)). Bei der Entwicklung des VC-Prolog-Tutors kann somit auf eine mehrjährige Erfahrung in diesem Bereich zurückgegriffen werden.

5.1 Ein Überblick über das VC-System

Beim VC-Prolog-Tutor handelt es sich um ein Client-Server-System, durch das die lokale Arbeitsumgebung des Lernenden mit einem zentralen Server verbunden wird. Der Client ist als Java-Applet realisiert, der Lernende kann dadurch auf die Lernumgebung mit Java-fähigen Browsern zugreifen. Durch den Server wird jedem Benutzer, der über das Internet auf ihn zugreift, ein eigener Prozeß zur Verfügung gestellt. Er fungiert dabei als Schnittstelle zur jeweiligen Applikation. Darüberhinaus verwaltet er über eine Datenbank die Daten für den Lehr- und Übungsbetrieb. Der Zugang zu den verschiedenen

Bereichen ist über personen- und gruppenspezifische Zugriffsrechte reglementiert. Durch das Zusammenspiel von Client und Server kann der Lernende auf die entfernt liegenden Daten wie auf Teile des eigenen Dateisystems zugreifen.

Aufbau und Funktionalität lassen sich an einer Architekturskizze (siehe Abbildung 3) veranschaulichen. Für Studenten und Tutoren stellt das System eigene Arbeitsumgebungen zur Verfügung, die an die unterschiedlichen Aufgabenstellung angepaßt sind. Tutoren werden bei der Verwaltung des Übungsbetriebes, dem Erstellen von Aufgaben und Musterlösungen, sowie bei der Analyse der abgegebenen Übungsaufgaben unterstützt (vgl. in Abbildung 3 die Komponente „Lehrumgebung“). Studenten bietet das System eine tutorielle Komponente, die die Lernenden bei der Programmentwicklung im Rahmen von Übungsaufgaben unterstützt und auf Fehler aufmerksam macht. Neben diesem angeleiteten Programmieren besteht die Möglichkeit, das System als freie Programmierumgebung selbständig zu nutzen und auch dabei von dem in das System eingebundene Lehrmaterial Gebrauch zu machen (vgl. „Lernumgebung“ in Abbildung 3).

Diese Umgebungen verkapseln die eigentliche Funktionalität des Systems. Das System besitzt zwei unterschiedliche Wissensbereiche, lehrstoff- und lernspezifisches Wissen. Diese Bereiche besitzen wiederum Unterbereiche, auf denen die Prozesse arbeiten, die für die Präsentation von Wissen, der Auswahl und Generierung von Übungen sowie der Diagnose zuständig sind. Die Prozesse werden in der Regel durch Aktionen des Benutzers aktiv, sie können sich aber auch gegenseitig auslösen. Um die Funktionsweise an einem Beispiel zu illustrieren: ein Student hat eine Übungsaufgabe bearbeitet und legt diese dem System zur Korrektur vor. Dazu benötigt der Diagnoseprozeß Wissen über die Aufgabenstellung und die korrekte Lösung der Aufgabe. Dieses Wissen bezieht er aus den Wissensbasen über die Übungsspezifikationen und dem Anwendungsdomänenmodell innerhalb des lehrstoffspezifischen Wissens. Zur Korrekur der Aufgabe benötigt er ferner lernspezifisches Diagnosewissen. Eine auf den einzelnen Studenten abgestimmte Korrektur der Aufgabe erfordert Informationen über den Betreffenden. Dieses Wissen bezieht der Prozeß aus dem Benutzermodell und vermerkt seinerseits dort den Erfolg der gerade bearbeiteten Aufgabe. Das Resultat des Diagnoseprozeß wird dem Benutzer über den Präsentationsprozeß mitgeteilt.

Im folgenden sollen im wesentlichen zwei Bereiche betrachtet werden, die im Kontext von tutoriellen Systemen von größter Wichtigkeit sind: die Repräsentation des Domänenwissens sowie die Fehlerdiagnose und -behandlung.

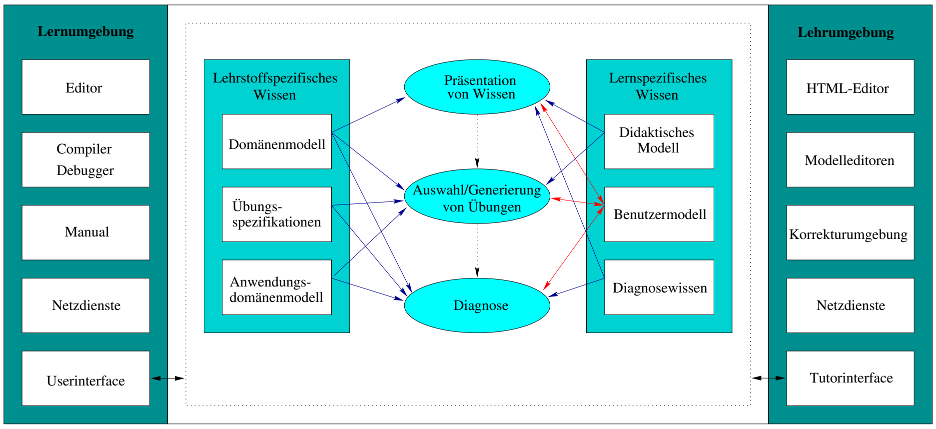

Abbildung 3: Der VC-Prolog-Tutor

Repräsentation des Domänenwissens In einem System, das Lernen über das Netz ermöglichen soll, muß Wissen in sehr viel stärkerem Grad explizit gemacht werden, als dies bei Präsenzveranstaltungen der Fall ist. In herkömmlichen Präsenzveranstaltungen können Vorkenntnisse verlangt und vorausgesetzt werden. Im Zweifelsfall kann ein routinierter Dozent anhand der Reaktion seines Auditoriums abschätzen, inwieweit das notwendige Vorwissen vorhanden ist, um das gegenwärtig zu behandelnde Thema einordnen und verstehen zu können. Sollte die Notwendigkeit bestehen, kann er – zur Not *ad hoc* durch eingeschobene Exkurse – das fehlende Hintergrundwissen vermitteln. Diese Flexibilität durch unmittelbares *feedback* ist in netzbasierten Lehr- und Lernumgebungen nicht gegeben. Hier kann nur mittelbar aus den Rückmeldungen des Systems – oder im schlimmsten Fall: erst aus den Übungsergebnissen – geschlossen werden, ob möglicherweise ein grundsätzliches Verstehensproblem zugrunde liegt. Aus diesem Grund muß das ganze relevante Wissen in einer Form bereitgestellt werden, die den Vernetzungsgrad der Domäne angemessen berücksichtigt und die Rezeption – also den Wissenserwerb – möglichst gut unterstützt.

Unserer Ansicht nach kann dies am besten durch eine umfassende Modellierung der involvierten Konzepte und Domänen in einer Art Ontologie erfolgen. Durch eine taxonomische Strukturierung soll es den Lernenden ermöglicht werden, die Struktur des Wissens in einem Sachbereich nachzuvollziehen (vgl. Edelmann (1996)). Die Granularität der Modellierung sollte so gewählt sein, daß sie zum einen den Erfordernissen des jeweiligen Kontextes genügt und zum anderen nicht durch unnötige Detailversessenheit die Lernenden verwirrt und den Blick auf das Ganze verstellt. Außerdem sollte es möglich sein, die Granularität später noch zu verfeinern, wenn sich etwa die

Notwendigkeit durch die Integration einer weiteren Domäne ergeben sollte, ohne die Konsistenz des Ganzen zu gefährden.

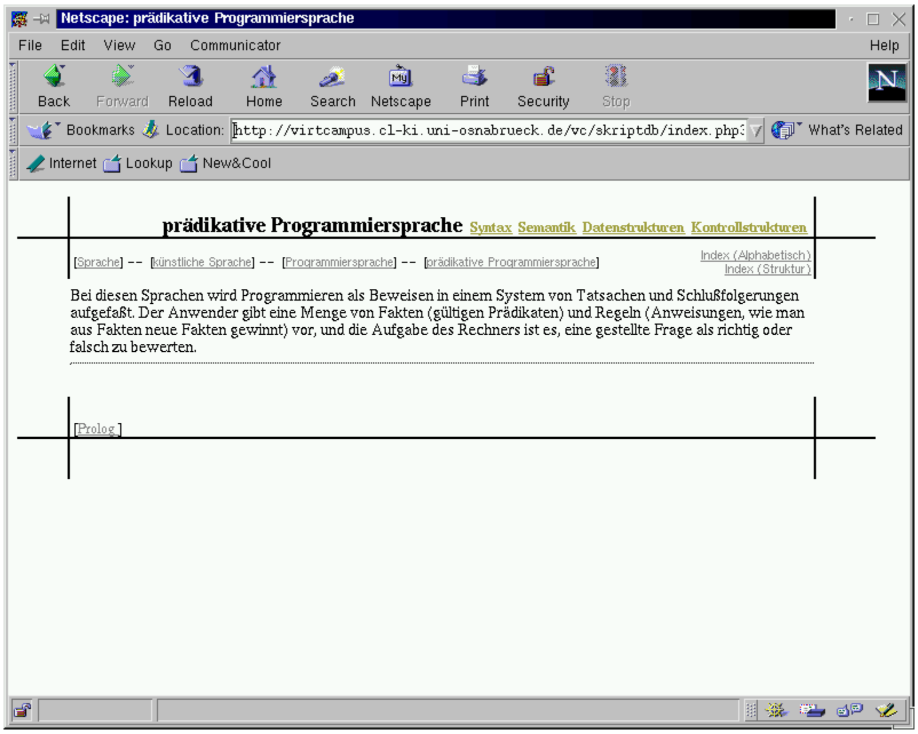

Abbildung 4: Beispiel einer Skriptseite. Eine Seite besteht aus der Vorstellung eines Konzepts. Konzepte, die im Sinn einer *part-of*-Relation zu dem Konzept gehören (hier: Syntax, Semantik, Daten- und Kontrollstrukturen) werden neben dem Konzeptnamen aufgeführt. Darunter wird die Stellung des Konzepts im Ableitungsbaum der *is-a*-Relationen dargestellt, ganz unten die Kinderknoten.

Formal läßt sich diese Struktur als gerichteter Graph mit Knoten und Kanten interpretieren. Die einzelnen Begriffe, wie z.B. „Programmiersprache“, stellen die Knoten innerhalb des Graphen dar. Durch die Kanten, also die Verbindungen zwischen Knoten, lassen sich Beziehungen zwischen den Begriffen darstellen. Die so modellierten Domänen können in Analogie zu Semantischen Netzen (zur Übersicht siehe Reichgelt (1991)) verstanden werden. Der hier verfolgte Ansatz läßt sich in Brachmans Typologie der Semantischen Netze dem „epistemological Level“ (Brachman, 1979) zuordnen. Abbildung 4 zeigt eine Seite des Skripts, die unseren Ansatz illustrieren soll. Das Skript ist eng mit dem VC-Prolog-Tutor verwoben und an dessen Anforderungen ausgerichtet, umfaßt aber keine Adaption der Präsentation an den Benutzer, wie beispielsweise adaptive Hyperbooks vgl. Fröhlich und Nejdl (1998), Fröhlich et al. (1998)).[4]

Fehlerdiagnose und -behandlung Die Fehlerbehandlung im VC-Prolog-Tutor geht auf ein Studentenprojekt zurück, das innerhalb eines Jahres eine intelligente Fehlerbehandlung für Prologprogramme entwarf und realisierte (Bönnen et al., 1999).

Die Fehlererkennung erfolgt in mehreren, aufeinander aufbauenden Modulen. Zunächst wird der Programmcode des Lernenden syntaktisch analysiert, dabei werden erkannte Syntaxfehler korrigiert. Im Falle von Ambiguitäten werden mehrere Korrekturalternativen erstellt, die mit entsprechenden Wahrscheinlichkeiten annotiert und an das Modul Fehlerbehandlung weitergeleitet werden. Das Resultat der Syntaxanalyse wird zur Überprüfung der semantischen Korrektheit an die Semantikanalyse weitergereicht. Die Semantikanalyse erfolgt in drei Schritten: Zunächst wird für jedes Goal, das in einer Prädikatsdefinition auftaucht, überprüft, ob das in ihm zu beweisende Prädikat auch tatsächlich definiert wurde. Dann erfolgt eine Analyse von Schleifenkonstrukten.[5] Da es keine sichere Methode zur Detektion von Endlosschleifen gibt, bedient sich die Semantikanalyse einer Heuristik, die eine Endlosschleife annimmt, wenn eine bestimmte Rekursionstiefe überschritten wurde. Im letzten Analyseschritt wird die Lösungsmenge der vom Lerner vorgelegten Aufgabe mit der Lösungsmenge der gestellten Aufgabe verglichen. Auch hier wird wieder eine Heuristik zugrunde gelegt: Es wird eine Auswahl von Anfragen und den dazu passenden Antworten eines Prologsystems mit den entsprechenden Daten einer abgegebenen Lösung verglichen. Die gefundenen Fehler werden an die Fehlerbehandlung weitergeleitet.

Als letztes Modul der Fehlererkennung erfolgt die Strukturanalyse. Diese basiert auf der Eigenheit von Prolog, daß die Struktur eines Prologprogrammes durch die grundsätzliche Herangehensweise an die Lösung gekennzeichnet ist. Wenn sich zwei Programme strukturell voneinander unterscheiden, unterscheiden sich auch die Lösungswege, deren Realisierung die Programme darstellen. Die Zuordnung einer Lösung zu einem Lösungsweg ermöglicht somit eine Rekonstruktion des Plans, den der Lernende bei der Bearbeitung der Aufgabe verfolgte. Existiert mindestens eine Musterlösung, die dem Programm bis auf einige Abweichungen strukturell gleicht, wird die Musterlösung

[4]Gegenwärtig wird an einer Integration des VC-Prolog-Skripts in ein Hyperbook gearbeitet.

[5]Wenn im folgenden von Schleifen in Prolog die Rede ist, ist natürlich die Emulation von Schleifen durch Rekursion gemeint. Schleifenkonstruktionen können dazu führen, daß ein Programm nicht terminiert, wenn beispielsweise bei der Abbruchbedingung einer Rekursion ein Fehler gemacht wurde.

mit den geringsten Abweichungen als Referenz für die Struktur des abgegebenen Programmes gewählt. Die Strukturanalyse wird durch eine Layoutanalyse unterstützt, die, unter Zugrundelegung von Programmierkonventionen, zur Disambiguierung möglicher Fehler beitragen kann. Die Fehlerbehandlung

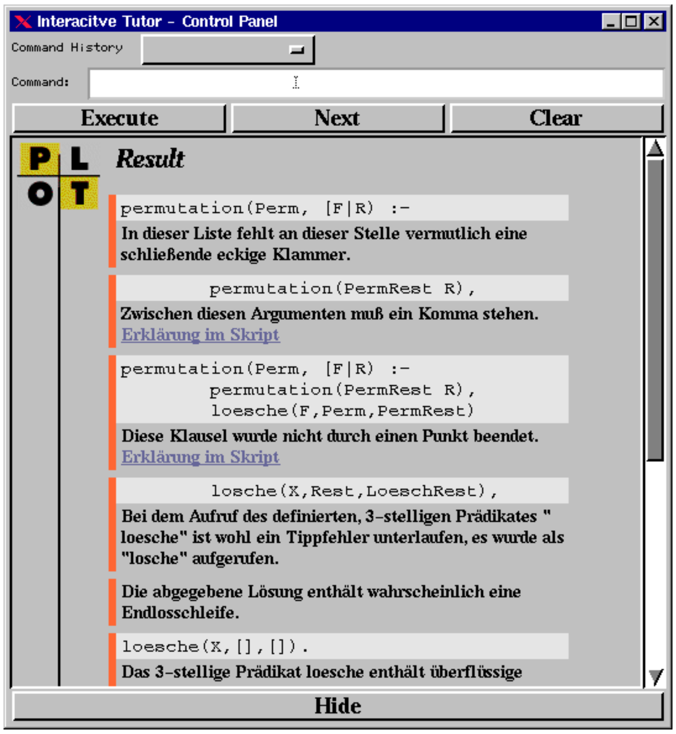

Abbildung 5: Beispiel einer Fehlererklärung, die sowohl syntaktische als auch semantische Fehler beinhaltet. Da mehrfach syntaktische Fehler auftraten, wird der Lernende auf die entsprechenden Seiten im Skript verwiesen.

interpretiert die von den einzelnen Analysemodulen gefundenen Fehler und präsentiert sie mit entsprechenden Erklärungen dem Benutzer (siehe Abbildung 5). Weiterhin werden während der Analyse des Programmcodes Informationen über die Struktur des Programmes generiert und an ein Modul weitergeleitet, das daraus den Ablauf des Programms graphisch veranschaulichen kann. Dies erleichtert dem Benutzer das Verständnis für Ablauf und Arbeitsweise des Prologbeweisers.

5.2 Ausblick

Das System wird im Sommersemester 1999 an der Universität Osnabrück im Rahmen einer Lehrveranstaltung zum logischen Programmieren eingesetzt und in der Praxis erprobt. Darüberhinaus steht das System derzeit allen Interessenten über die URL `http://virtcampus.cl-ki.uni-osnabrueck.de` zur Verfügung.

In der verbleibenden Laufzeit des Projekts wird vor allem die Benutzermodellierung und ihr Zusammenwirken mit den anderen Komponenten im Zentrum stehen. Da das bislang realisierte System als universeller Rahmen für internetbasierte Lehr- und Lernumgebungen mit integrierter Verwaltung eines Übungsbetriebes dienen kann, kann das System auch auf die Erfordernisse anderer Fachgebiete und Domänen übertragen werden.

Literatur

Adam, A. und Laurent, J.P. (1980): "Laura, A System to Debug Student Programs". *AI* 15: S. 75–122.

Anderson, J.R.; Boyle, C.F.; Corbett, A.T. und Lewis, M.W. (1990): "Cognitive Modeling and Intelligent Tutoring". In: *Artificial Intelligence and Learning Environments*, herausgegeben von Clancey, J. und Soloway, E., Cambridge, London: MIT-Press, S. 7 – 51. Reprinted from Artificial Intelligence: an International Journal, volume 42, number 1, 1990.

Bauer, M. (1996): "Acquisition of User Preferences for Plan Recognition". In: *Fifth International Conference on User Modelling*, herausgegeben von Chin, D. S. 105 – 112.

Bonar, J. und Solloway, E. (1985): "Programming Knowledge: A Major Source of Misconceptions in Novice Programmers". *Human-Computer Interaction* 1: S. 133–161.

Bongardt, G.; Dalinghaus, K.; Dittmann, H.; Huber, M.; Krumeich, A.; Peylo, C.; Rehm, G.; Rother, A. und Teiken, W. (1997): "Semf: Semantische Fehlerbehandlung". Technischer Bericht, Institut für Semantische Informationsverarbeitung.

Bönnen, C.; Gerstenhöfer, J.; Hartmann, N.; Holzwarth, A.; Lenschow, K. Kober N. und Meissner, B. (1999): "Plot: ProLog Online Tutor". Technischer Bericht, Institut für Semantische Informationsverarbeitung.

Brachman, R. (1979): "On the Epistemological Status of Semantic Networks". In: *Associative Networks: Representation and Use of Knowledge by Computers*, herausgegeben von Findler, N.V., New York: Academic Press, S. 3–50.

Brunner, J.S. (1966): *Towards a Theorie of Instruction.* New York: Norton.

Edelmann, W. (1996): *Lernpsychologie.* Weinheim: Psychologie Verlags Union, 5. Auflage.

Fröhlich, P. und Nejdl, W. (1998): "A Database-Oriented Approach to the Design of Educational Hyperbooks". In: *Proceedings of the workshop 'Intelligent Educational Systems on the World Wide Web'. 8th Conference of the AIED Society, Kobe, Japan, 18-22 August 1998.*

Fröhlich, P.; Nejdl, W. und Wolpers, M. (1998): "Kbs-hyperbook - An Open Hyperbook System for Education, 10th World Conference on Educational Multimedia and Hypermedia (EDMEDIA 98), 20.-25. June 98, Freiburg, Germany".

Gust, H.; Ludewig, P.; Krüger, A. und Kronenberg, F. (1993): "Cavol: Computer Assisted Vocabulara Learning". Technischer Bericht, Institut für Semantische Informationsverarbeitung.

Hong, J. (1997): "An Intelligent Tutoring and Assessing System for Prolog Programming". In: *Artificial Intelligence in Education*, herausgegeben von du Boulay, B. und Mizoguchi, R. Amsterdam, Berlin, Oxford etc.: IOS Press, S. 586–588.

Johnson, W.L. (1990): "Understanding and Debugging Novice Programs". In: *Artificial Intelligence and Learning Environments*, herausgegeben von Clancey, J. und Soloway, E., Cambridge, London: MIT-Press, S. 51 – 99. Reprinted from Artificial Intelligence: an International Journal, volume 42, number 1, 1990.

Krüger, A. (1995): *Interaktives Lernen räumlicher Präpositionen mit dem Sprachlernsystem PROMISE.* Diplomarbeit, Universität Osnabrück.

Papert, S. (1972): "Teaching children thinking". *Programmed Learning and Educational Technology* 9 (5): S. 245–255.

Papert, S. (1980): *Mindstorms: Children, Computers, and Powerful Ideas.* Harvest Studies in Cognitive Science. Brighton, Sussex: The Harvest Press.

Park, O.-C.; Perez, R.S. und Seidel, R.J. (1987): "Intelligent CAI: Old Wine in New Bottles, or a new Vintage". In: *Artificial Intelligence and Instruction: Applications and Methods*, herausgegeben von Kearsley, G., Reading, Massachussets: Addison- Wesley, S. 11 – 45.

REICHGELT, J. (1991): *Knowledge Representation: An AI Perspective.* Norwood, New Jersey: Ablex Publishing Corporation. Zur Uebersicht und zum reinschnuppern geeignet, wenig detailliert und eher etwas oberflaechlich -¿ bedingt zitierfaehig.

SCHANK, R.C. UND ABELSON, R.P. (1977): *Scripts, Plans, Goals and Understanding.* Hilsdale, New Jersey: Lawrence Erlbaum Associates.

SCHRÖDER, O. (1996): "Intelligente Tutorielle Systeme". In: *Wörterbuch der Kognitionswissenschaft*, herausgegeben von Strube, G.; Becker, B.; Freska, C.; Hahn, U.; Opwis, K. und Palm, G., Stuttgart: Klett – Cotta, S. 289–291.

SCHULMEISTER, R. (1997): *Grundlagen hypermedialer Lernsysteme.* München, Wien: Oldenbourg, 2. Auflage.

SEELS, B. (1989): "The Instructional Design Movement in Educational Technology". *Educational Researcher* 18: S. 11 – 15.

SKINNER, B.F. (1968): *The Technology of Teaching.* New York: Appleton-Century Crofts.

SPECHT, M. UND WEBER, G. (1997): "Kognitive Lernermodellierung". *Kognitionswissenschaft* 6 (4): S. 165–176.

WEBER, G. (1994): *Fallbasiertes Lernen und Analogien.* Weinheim: Psychologie Verlags Union.

WENGER, E. (1987): *Artificial Intelligence and Tutoring Systems.* Los Altos, California: Morgan Kaufmann.

WINKELS, R. UND BREUKER, J. (1992): "What's in a ITS? a Functional Decomposition". In: *New Directions for Intelligent Tutoring Systems*, herausgegeben von Costa, E., Berlin, Heidelberg, New York: Springer, S. 57 – 68.

WOOLF, B. P. (1992): "Ai in Education". In: *Encyclopedia of Artificial Intelligence*, herausgegeben von Shapiro, S., John Wiley & Sons, Inc., S. 434–444.

YANG, Q. (1997): *Intelligent Planning: a decomposition and abstraction based approach.* Berlin, Heidelberg, New York: Springer, artificial Intelligence Auflage.