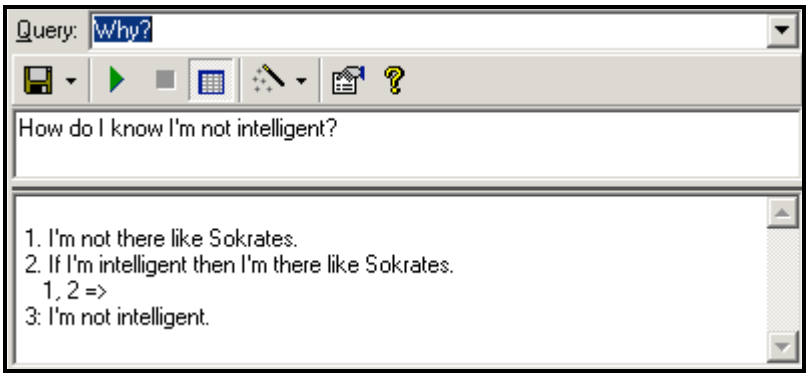
Natürlichsprachliche Datenbankschnittstellen: Sprachverstehen und Benutzerinteraktion

Lizenziatsarbeit

Christoph Biveroni



Eingereicht bei Prof. Dr. M. Hess
Institut für Computerlinguistik
Universität Zürich

Oktober 2002

Christoph Biveroni
Wiesenstr. 19
8307 Effretikon
Tel. 052 343 15 23
E-Mail ChristophBiveroni@access.unizh.ch

Inhaltsverzeichnis

Inhaltsverzeichnis **2**

Abbildungsverzeichnis **4**

1 Einleitung **5**

2 Datenbanken **5**
2.1 Das Datenbanksystem 6
2.2 Benutzerschnittstellen 6
2.3 Relationale Datenbanksysteme 7
2.3.1 Das Relationenmodell 7
2.3.2 Relationenmodell und logische Programmierung 8
2.3.3 Die Relationenalgebra 9
2.3.4 Die Datenmanipulationssprache SQL 10
2.4 Alternative Datenbanksysteme 11
2.4.1 Objektorientierte Datenbanksysteme 11
2.4.2 Deduktive Datenbanksysteme 12
2.4.3 Heterogene Datenbanksysteme 13
2.5 Trends im Datenbankbereich 13
2.5.1 Data Warehousing 13
2.5.2 Datenbanktechnologie im WWW 14

3 Natürlichsprachliche Schnittstellen zu Datenbanken **15**
3.1 Natürlichsprachliche Retrieval-Schnittstellen 15
3.2 Natürlichsprachliche Datenbankschnittstellen 15
3.3 Fortschritte der NLIDB-Forschung 17
3.3.1 Prototypen-Phase 17
3.3.2 Front-End-Phase 17
3.3.3 Benutzer-zentrierte Phase 18
3.4 Perspektiven der NLIDB-Forschung 19
3.4.1 Natural Language Processing 19
3.4.2 Implementierung und Portabilität 21
3.4.3 Mensch-Maschine-Kommunikation 21

4 Das Front-End-Paradigma **22**
4.1 Systemarchitektur und Verarbeitung des Eingabesatzes 22
4.1.1 Von der Eingabe zur Bedeutungsrepräsentation 24
4.1.2 Von der Bedeutungsrepräsentation zur Datenbankabfrage 26
4.2 Probleme der Analyse, Interpretation und Generierung 27
4.2.1 Verarbeitung von Datenbankwerten in der Anfrage 28
4.2.2 Verarbeitung von Diskursphänomenen 29
4.2.3 Pragmatikanalyse und kooperative Antworten 30
4.2.3.1 Präsuppositionen 31
4.2.3.2 Benutzerabsichten 31
4.2.3.3 Front-End-Systeme versus Dialogsysteme 33
4.2.4 Generierung des anwendungsspezifischen Formalismus 34
4.3 Vorteile der Front-End-Architektur 35
4.4 System X: Ein Front-End-System 36
4.4.1 Analyse und Interpretation 36
4.4.2 Portabilität 39

5 Faktoren für die mangelnde Akzeptanz **39**
5.1 Alternative Benutzerschnittstellen 40
5.2 Schwächen von NLIDB-Systemen 40
5.2.1 Sprachverstehen 40
5.2.2 Handhabung und Habitabilität 41

5.2.3 Kosteneffektivität ... 42

6 Neue Lösungsansätze ... 42
6.1 Dialogsysteme ... 43
6.1.1 Die Architektur eines typischen Dialogsystems ... 43
6.1.2 KDA: ein Front-End-System mit Dialogfähigkeiten ... 44
6.2 Multimodalität ... 46
6.3 DOA: Integration in DB-Architektur ... 48
6.3.1 VIENA Classroom CSCW ... 48
6.3.2 Das NLIDB ... 49
6.3.2.1 Lexikalische Komponente ... 52
6.3.2.2 Semantische Komponente ... 52
6.3.2.3 Unterhaltskomponente ... 53
6.3.3 Beurteilung des DOA ... 54
6.4 Moderne kommerzielle NLIDBs ... 55
6.4.1 English Query ... 55
6.4.1.1 Die Bestandteile von EQ ... 55
6.4.1.2 Phrasierungen ... 57
6.4.1.3 Realisierung einer EQ-Anwendung ... 59
6.4.1.3.1 Modell-Editor ... 59
6.4.1.3.2 Assistenten zur Erstellung eines Modells ... 60
6.4.1.3.3 Testumgebung ... 61
6.4.1.3.4 Anwendung für den Endbenutzer ... 62
6.4.1.4 Natürlichsprachliche Analyse und Generierung ... 65
6.4.1.4.1 Datenbankwerte in der Anfrage ... 65
6.4.1.4.2 PP-Anbindung ... 66
6.4.1.4.3 Skopusambiguitäten ... 67
6.4.1.4.4 Konjunktion und Disjunktion ... 67
6.4.1.4.5 Anaphorische Referenzen und Ellipsen ... 68
6.4.1.4.6 Präsuppositionen ... 69
6.4.1.4.7 Deixis ... 69
6.4.1.4.8 Fehlertoleranz ... 70
6.4.1.4.9 Meta-Wissen ... 70
6.4.1.5 Beurteilung von EQ ... 71
6.4.2 EasyAsk (English Wizard) ... 72

7 Einsatz von NLIDB-Systemen ... 74
7.1 Einsatzszenarien ... 75
7.2 Entwurf eines NLIDB ... 77

8 Appendix A: Bibliographie ... 80

9 Appendix B: Materialien zur EQ-Beispielanwendung ... 85
9.1 Diagramm der Datenbank ... 85
9.2 Beispielanfragen ... 87

Abbildungsverzeichnis

Abbildung 1 - Architektur eines prototypischen Front-End-Systems 23
Abbildung 2 - Architektur von System X 37
Abbildung 3 - Systemarchitektur eines Dialogsystems mit Unterstützung gesprochener Sprache 44
Abbildung 4 - Systemarchitektur von KDA 45
Abbildung 5 - Corrective Guidance Subsystem von KDA 46
Abbildung 6 - VIENA CLASSROOM inklusive Question Support Facility und NLI 49
Abbildung 7 - Systemarchitektur des VIENA CLASSROOM NLI 51
Abbildung 8 - Systemarchitektur von EQ im Verbund mit SQL-Server-DB 56
Abbildung 9 - Dialogfenster für die Spezifizierung einer Verb-Phrasierung 58
Abbildung 10 - Modell-Editor: Semantics-Ansicht 60
Abbildung 11 - Testumgebung: Antwort-Ansicht des Model-Test-Werkzeugs 61
Abbildung 12 - Testumgebung: Analyse-Ansicht des Model-Test-Werkzeugs 61
Abbildung 13 - Einsatzszenario: Web-basierter, natürlichsprachlicher Zugriff auf SQL Server 62
Abbildung 14 - ASP-basierte EQ-Web-Anwendung: Default-Modus 63
Abbildung 15 - ASP-basierte EQ-Web-Anwendung: Question-Builder-Modus 64
Abbildung 16 - Klärungs-Dialogfenster zur semantischen Bestimmung eines Namens 66
Abbildung 17 - Koordination von zwei Hauptsatz-Anfragen 68
Abbildung 18 - Klärungs-Dialogfenster Deixis 69
Abbildung 19 - Meta-Wissen über das semantische Modell 71
Abbildung 20 - Client-Schnittstelle der Web-basierten Client-Server-Lösung von ENGLISH WIZARD 74
Abbildung 21 - Architektur eines Daten-Integrationssystems 76
Abbildung 22 - Diagramm der SQL-Server-Datenbank 86

(Die Abbildung auf der Titelseite stellt einen Screenshot des Modell-Test-Werkzeuges von ENGLISH QUERY (EQ) dar. Die Fähigkeit von EQ, dynamisch Wissen zu assertieren und daraus einfache Schlüsse ziehen, ist im Zusammenhang mit der Wiedergewinnung von Daten aus einer Datenbank wohl eher als Spielerei zu betrachten.)

1 Einleitung

Natürlichsprachliche Benutzerschnittstellen zu Datenbanken („NLIDBs“) können auf eine verhältnismässig lange Tradition zurückblicken: Bereits Ende der 1960er-Jahre werden erste Versuche unternommen, die natürliche Sprache als Mittel zur Datenwiedergewinnung, damals noch aus proprietären Test-Datenbanken, nutzbar zu machen. Während der folgenden zwei Jahrzehnte entstehen komplexe Sprachverarbeitungssysteme, welche als Schnittstellen zu bestehenden relationalen „Real-World“-Datenbanken funktionieren und in einigen Fällen sogar kommerziell eingesetzt respektive als kommerzielles Produkt vertrieben werden.

Vor diesem Hintergrund ist die Tatsache, dass sich seit dem Beginn der 1990er-Jahre hinsichtlich der Forschungs- und Entwicklungstätigkeit eine eigentliche NLIDB-„Krise“ abzeichnet,[1] zunächst etwas irritierend. Ich möchte in der vorliegenden Arbeit denn auch zuerst der Frage nachgehen, welche Faktoren dafür verantwortlich sein könnten, dass sich NLIDB-Systeme bis heute nicht zur ernsthaften Alternative im Bereich der Datenbankschnittstellen entwickeln konnten und dass das akademische Interesse an NLIDB-Systemen in den 1990er Jahren deutlich zurückgegangen ist. Anhand der Untersuchung älterer und neuerer Systeme soll aufgezeigt werden, was NLIDBs zur „Blütezeit“ der NLIDB-Forschung in den 1980ern zu leisten imstande waren und wo sich Schwächen und Unzulänglichkeiten offenbaren.

Basierend auf dieser tendenziell „historischen“ Untersuchung soll anschliessend der Bezug zur Gegenwart hergestellt werden, indem diskutiert wird, unter welchen Umständen der Einsatz von NLIDBs sinnvoll und gerechtfertigt erscheint und wie entsprechende „moderne“ NLIDB-Systeme konzipiert sind bzw. sein müssten.

Betont werden dabei Aspekte der Datenwiedergewinnung sowie der Analyse der natürlichsprachlichen Anfrage, während die Generierung der Antwort weniger ausführlich besprochen und die Eingabe und Änderung von Daten nur gestreift wird. Angaben zu NLIDB-Systemen stammen in den meisten Fällen aus Forschungsberichten und Produkt-Dokumentationen, was nicht zuletzt auf den Umstand zurückzuführen ist, dass sich die Suche nach funktionsfähigen, testfähigen oder zumindest distributionsfähigen NLIDB-Systemen als schwierig erwies und praktisch[2] erfolglos blieb.

2 Datenbanken

Für die vorliegende Arbeit relevant ist die Frage, inwiefern sich die *strukturierte* Ablage von Daten in einer Datenbank von anderen, weniger systematischen und komplexen Ansätzen der Datenspeicherung unterscheidet. Diejenigen Datenbankmodelle, auf welche ich mich bei der folgenden Besprechung von NLIDB-Systemen beziehe, sollen kurz vorgestellt werden.

---

[1] Vgl. z.B. [Androutsopoulos et al. 95].

[2] Abgesehen vom uneingeschränkt verfügbaren ENGLISH QUERY hatte/habe ich Zugang zu einem hinsichtlich der Funktionalität stark eingeschränkten Prolog-Framework einer natürlichsprachlichen Schnittstelle zu temporalen Datenbanken (vgl. [Androutsopoulos 96]) sowie zu der zwar lauffähigen, aber sehr rudimentären NLP-Komponente NLBEAN, implementiert als Java Enterprise Bean (vgl. [NLBean]). Sowohl das Framework als auch die Java-Bean sind zwar „programmiertechnisch“ interessant, als Testsystem jedoch nicht zu gebrauchen.

2.1 Das Datenbanksystem

Die verschiedenen Anforderungen an eine „Datenbank“ werden von folgender Definition des Begriffs *Datenbankverwaltungssystem* (*database management system*, *DBMS*) zusammengefasst:

> [DBMS] sind Softwareprodukte für die dauerhafte, integre und anwendungsunabhängige Speicherung und Verwaltung sowie die flexible und bequeme Verwendung von grossen, mehrfachbenutzbaren integrierten Datenbasen. [Rechenberg und Pomberger 99: 876]

Der Begriff der *Datenbasis* bezeichnet „eine Menge von Daten, die aus der Sicht der Systembetreiber in irgendeiner Weise als zusammengehörig betrachtet werden“ [Rechenberg und Pomberger 99: 876]. Diese Datenmenge kann man im prädikatenlogischen Sinne als Domäne eines Modells des interessierenden Ausschnittes der „realen“ Welt betrachten; typischerweise stellt diese *Anwendungsdomäne* im Datenbankbereich „the best available approximation“ [Binot 91: 102] nicht etwa der gesamten „realen“ Welt, sondern vielmehr eines Teils davon, welcher durch innere Beziehungen zusammengehalten und für den jeweiligen Anwendungskontext relevant ist, dar.

Dauerhaft heisst, dass Daten für einen beliebig langen Zeitraum ohne Gefahr des Datenverlustes aufbewahrt werden können. Diese Anforderung verlangt nach effizienten Mechanismen zur Verwaltung des Sekundärspeichers, z.B. des Festplattenspeichers.

Integre Speicherung von Daten bedeutet, dass nach Möglichkeit nur Daten, die einen Sachverhalt aus der realen Welt korrekt beschreiben, gespeichert werden, und zwar in konsistenter, d.h. widerspruchsfreier Art und Weise. Zur Sicherstellung der *Integrität* im weiteren Sinne gehört „der Schutz der Datenbank vor unberechtigtem Zugriff und Verändern“. [Rechenberg und Pomberger 99: 876]

Anwendungsunabhängige Speicherung und Verwaltung von Daten in einer Datenbasis ist dann gewährleistet, wenn die Verwendung einer Datenbank „ohne Kenntnis des die jeweiligen Daten erzeugenden Programms und der darin enthaltenen Festlegungen zur Datenstrukturierung“ [Rechenberg und Pomberger 99: 876] möglich ist.

Häufig ist mit der Bezeichnung „Datenbank“ eigentlich ein *Datenbanksystem* gemeint, d.h. ein spezifisches *Informationssystem*, welches aus einem DBMS sowie der *Datenbank* im engeren Sinne besteht. Die Datenbank beinhaltet die Datenbasis, d.h. die Nutzdaten, alle zu deren Beschreibung erforderlichen Daten und weitere Hilfsdaten.

2.2 Benutzerschnittstellen

Die Benutzerschnittstelle des DBMS trägt entscheidend dazu bei, dass das Einfügen neuer Daten und das Verändern, Löschen und Aufsuchen vorhandener Daten „flexibel und bequem“ erfolgen kann. In der Praxis findet man diverse Typen von Benutzerschnittstellen, die sich einerseits in der zur Verfügung stehenden Eingabe-Modalität, andererseits in ihrer Ausrichtung auf bestimmte Benutzergruppen unterscheiden. [Elmasri und Navathe 94]

- *Formalsprachliche Schnittstellen*. Formale Datenbanksprachen wie z.B. SQL erlauben die Formulierung eindeutiger Anfragen, sind im Hinblick auf eine optimale Zusammenarbeit mit dem DBMS entwickelt, können prinzipiell bei verschiedenen Datenbanken und DBMS eingesetzt werden und sind entsprechend weit verbreitet. Ihre Verwendung setzt allerdings einen geschulten Benutzer voraus.
- *Menü-basierte Schnittstellen*. Options-Listen eröffnen dem Benutzer einen wohldefinierten Rahmen, um seine Anfrage zu formulieren. Ohne sich die spezifischen Befehle und die Syntax einer formalen Abfragesprache merken zu müssen, kann er Schritt für Schritt die interessierenden Daten auswählen und Beschränkungen setzen.
- *Formular-basierte Schnittstellen*. Formulare oder Eingabemasken repräsentieren eine gewöhnlich auf den unbedarften Anwender zugeschnittene Eingabe-Modalität, welche sich nur für unveränderliche Standard-

Transaktionen eignet. Der Benutzer formuliert eine Anfrage, indem er diejenigen Felder eines mit der betreffenden Relation korrespondierenden Formulars, deren Wert bereits bekannt ist, ausfüllt – das DBMS liefert in der Folge die passenden Daten der unspezifizierten Felder zurück. [Androutsopoulos et al. 95]
Ein flexible(re) formularbasierte Methode zur Befragung relationaler Datenbanken liegt im „query by example" mit der von IBM entwickelten Abfragesprache *QBE* (*Query By Example*) vor: Hier wird die Anfrage formuliert, indem Gitter, welche exakt die Struktur der einzelnen Relationen abbilden, einerseits mit Konstanten und Variablen gefüllt und andererseits, im Gegensatz zu weniger mächtigen Formular-basierten Methoden, beliebig kombiniert werden. Dieses u.U. Relationen-übergreifende Muster wird anschliessend auf die Datenbasis abgebildet.
- *Graphische Schnittstellen.* Typischerweise stellen graphische Schnittstellen das Datenbankschema in Diagramm-Form dar und ermöglichen dem Benutzer, durch Manipulieren des Schemas mit Hilfe eines *Zeigegerätes* (z.B. einer Maus) seine Anfrage zusammenzustellen. Sie sind häufig kombiniert mit Menüs.[3]
- *Natürlichsprachliche Schnittstellen*. Bei diesem Schnittstellen-Typ erfolgt der Zugriff auf Daten über Anfragen in einer mehr oder weniger eingeschränkten natürlichen Sprache. (Natürlichsprachliche Schnittstellen zu Datenbanken sind Gegenstand der vorliegenden Arbeit.)
- *Multimodale Schnittstellen*. [Bretan et Karlgren 93] u.a. vertreten die Ansicht, dass die Kombination verschiedener Eingabe-Modalitäten zu Schnittstellen führt, die in Handhabung und Zuverlässigkeit den unimodalen Schnittstellen überlegen sind. Multimodalität bedeutet, dass die einzelnen Modalitäten untereinander abgeglichen werden, so dass sie verschiedene Sichtweisen auf denselben logischen Sachverhalt (z.B. die logische Repräsentation einer Datenbankanfrage) anbieten.

Stellvertretend für Benutzer-spezifische Schnittstellen sei hier die Schnittstelle für die Gruppe der *Datenbankadministratoren* (*DBA*) erwähnt: Meistens ist es diesen Benutzern vorbehalten, z.B. Zugriffsrechte zu vergeben, System-Parameter einzustellen sowie die Struktur der Datenbasis zu verändern.

2.3 Relationale Datenbanksysteme

Relationale Datenbanksysteme repräsentieren das momentan bekannteste und am weitesten verbreitete Datenbank-Paradigma. In den folgenden Unterkapiteln soll einerseits das zugrunde liegende Datenmodell beschrieben und andererseits kurz diskutiert werden, welche Charakteristika relationaler Datenbanken, abgesehen von ihrer Verbreitung, die Entwicklung natürlichsprachlicher Schnittstellen vor allem zu SQL-Datenbanken begünstigt haben.

2.3.1 Das Relationenmodell

Das *Relationenmodell* ermöglicht es, Daten auf eine einfache und logische Art und Weise zu speichern und zu organisieren.[4] Eine Datenbasis besteht aus einer Sammlung von *Relationen*, deren Struktur man gewöhnlich als Tabelle modelliert: Jede Reihe enthält eine Menge von Daten, die eine *Entität* oder eine Beziehung zwischen Entitäten der zu repräsentierenden bzw. der *interessierenden Anwendungswelt* (*reale Welt*, *real world*) beschreiben, und die Namen der Kolonnen bezeichnen die Bedeutung der Daten in der jeweiligen Kolonne.
Die Terminologie[5] des Relationenmodells verwendet für die Tabellenreihe den Begriff *Tupel*, für den Kolonnennamen den Begriff *Attribut* und für den Datentyp, der beschreibt, welche Typen von atomaren

[3] Vgl. z.B. [Catarci et al. 95], [Murray et al. 98]. Von [Harris 98] als „Point&Click interface" bezeichnet.
[4] Das Relationenmodell wird durch [Codd 70] (und folgende Arbeiten) eingeführt.
[5] Vgl. [Elmasri and Navathe 94].

Werten innerhalb einer Kolonne erscheinen dürfen, den Begriff *Domäne*. Ein Tupel entspricht also einem Datensatz, dessen Interpretation bezüglich der Attribute und v.a. derer Domänen Informationen über ein Objekt oder über eine Beziehung zwischen Objekten der Anwendungsdomäne liefert.
Neben den Daten selbst, den *Primärdaten*, werden in einem Datenbanksystem „Daten über Daten", sog. *Metadaten* gespeichert, die es ermöglichen, die Primärdaten zu interpretieren, d.h. aus den Primärdaten Informationen zu gewinnen. Im Relationenmodell ist jeder Relation, unter diesem Gesichtspunkt oft *Relationeninstanz* genannt, genau ein *Relationenschema* zugeteilt, das die Struktur und die Semantik der Relationeninstanz beschreibt und aus dem theoretisch beliebig viele Relationeninstanzen abgeleitet werden können.
Ein Relationenschema besteht aus einem *Relationennamen R* und einer Liste von Attributen $A_1, A_2, ..., A_n$: $R(A_1, A_2, ..., A_n)$. Jedes Attribut A_i ist der Name einer *Rolle*, welche eine Domäne $D = dom(A_i)$ im Relationenschema einnimmt.
Das Relationenschema $R(A_1, A_2, ..., A_n)$ beschreibt eine Relationeninstanz r (auch $r(R)$ genannt), wobei die Anzahl Attribute im Relationenschema den *Grad* (*degree*) n der Relationeninstanz bestimmt. Die Relationeninstanz r besteht aus einer Menge von n-Tupeln: $r = \{t_1, t_2, ..., t_m\}$. Jedes n-Tupel t ist eine geordnete Liste von n Werten $t = <v_1, v_2, ..., v_n>$, wobei jeder Wert v_i $(1 \leq i \leq n)$ entweder ein Element von $dom(Ai)$ oder ein spezieller NULL-Wert[6] ist.
Eine Relationeninstanz beinhaltet zu einem bestimmten Zeitpunkt, dem aktuellen *Relationenzustand*, von allen möglichen Kombinationen nur diejenigen gültigen Tupel, die einen bestimmten Zustand der Anwendungswelt repräsentieren.
Sobald sich der Zustand der interessierenden Welt ändert, wird idealerweise auch die Relation in einen anderen Zustand überführt. Das Relationenschema allerdings wird höchstens dann verändert, wenn sich an der Struktur der Tupel etwas ändert, z.B wenn ein neues Attribut eingefügt wird.

2.3.2 Relationenmodell und logische Programmierung

Ähnlich der Art und Weise, wie Datensätze in einer relationalen Datenbank als geordnete Listen in Relationen organisiert sind, funktioniert die Ablage von Fakten (und Regeln) eines Prolog-Programmes als Klauseln eines Prädikats in der Prolog-Datenbank. Während in einer Relation jede Kolonne hinsichtlich der Semantik, des Datentyps und des Wertebereichs verwandte Werte enthält, definiert ein Prolog-Prädikat eine feste Anzahl von Argumenten, wobei an identischen Argumentspositionen verschiedener Klauseln des Prädikats verwandte Werte stehen.
Eine Relation r vom Grad n lässt sich auf eine Menge von Klauseln des logischen Prädikats r/n abbilden, indem jedem relationalen Tupel t aus r eine Klausel, deren Prädikatsfunktor dem Relationennamen von r und deren Prädikatsargumente den Attributswerten von t entsprechen, zugewiesen wird. [Marcus 86]
Der wesentliche Unterschied zwischen einem relationalen Tupel und der korrespondierenden Prolog-Klausel besteht darin, dass bei der letzteren die Bedeutung der einzelnen Argumente alleine (implizit) durch ihre Argumentsposition, bei ersterem hingegen (explizit) durch die jeweiligen Attribute spezifiziert ist. Möchte man in Prolog neben den einzelnen Tupeln auch das Relationenschema einer Relation abbilden, so lässt sich dies z.B. durch Definition eines entsprechenden Prädikats *table/1*, das die Funktor-Argument-

[6] Der NULL-Wert ist ungleich dem mathematischen „0" und dem Leerzeichen (blank). Er wird folgendermassen interpretiert: Entweder kann das betreffende Attribut nicht auf das Tupel angewendet werden, oder der Attribut-Wert für das Tupel ist unbekannt, oder aber der Attribut-Wert ist bekannt, aber noch nicht in die Datenbasis aufgenommen worden. Vgl. [Elmasri and Navathe 94].

Struktur des Tupel-Prädikats festlegt, und durch Definition eines Prädikats *type/1*, das Datentypen und Werterestriktionen[7] für einzelne Argumente festlegt, bewerkstelligen. [Marcus 86]

star_wars		
id	**title**	**rating**
1	The Phantom Menace	2
2	The Clone Wars	3
3	A New Hope	5
4	The Empire Strikes Back	6
5	Return Of The Jedi	3

Die entsprechenden Daten in Prolog-Klauselform:

```
table(star_wars(id,title,rating)).

type(id(int,notnull)).
type(title(atom,notnull)).
type(rating(int,notnull)).

star_wars(1,'The Phantom Menace',2).
star_wars(2,'The Clone Wars',3).
star_wars(3,'A New Hope',5).
star_wars(4,'The Empire Strikes Back',6).
star_wars(5,'Return Of The Jedi',3).
```

Professionelle Prolog-Umgebungen wie z.B. MASTER ProLog[8] stellen Schnittstellen zu verbreiteten relationalen DBMS zur Verfügung, welche relationale Tupel direkt auf Prolog-Klauseln abbilden.[9] So kann eine Prolog-Anwendung relativ einfach an externe relationale Datenbasen angebunden werden, was einerseits der Prolog-Anwendung den effizienten Umgang mit grossen Datenmengen überhaupt erst ermöglicht und andererseits die Erweiterung relationaler Datenbanksysteme um deduktive Module zur Ableitung neuer Daten aus bereits bestehenden vereinfacht.

2.3.3 Die Relationenalgebra

Die *Relationenalgebra* ist eine formale Sprache, die der prozeduralen Spezifizierung von Anfragen im Zusammenhang mit relationalen Datenbanken dient. Sie beschreibt die für eine Manipulierung ganzer Relationen benötigten Operationen. [Elmasri and Navathe 94]

Man unterscheidet sieben Standardoperationen, welche gewöhnlich in zwei Gruppen eingeteilt werden: Als Mengenoperationen (gemäss der mathematischen Mengentheorie) gelten die *Vereinigung*, der *Durchschnitt* und die *Differenz* von Mengen sowie das *kartesische Produkt*; als speziell für relationale Datenbanken entwickelte Operationen gelten die *Projektion* (*project*), die *Restriktion* (*select*) und der *Verbund* (*join*).

Die *join-Operation* wird wichtig, sobald eine relationale Datenbank mehr als eine einzige Relation umfasst. Sie dient dazu, verbundene Tupel aus zwei Relationen in einem Tupel zusammenzufassen, und ermöglicht dadurch die Verarbeitung von Beziehungen zwischen Relationen.

---

[7] Mit „notnull“ könnte man z.B. aussagen, dass der Wert an dieser Argumentsposition definiert sein muss, d.h., dass das Argument nicht „leer“ bzw. mit dem NULL-Wert gefüllt sein darf.

[8] Früher als „BIM ProLog“ bekannt, wird das „commercial high performance Prolog development and run-time system“ nun von IT Masters vertrieben. Gemäss Feature-Liste werden die RDBS Sybase und Oracle unterstützt; erhältlich ist die Umgebung für UNIX und Windows NT. Vgl. [MASTER ProLog].

[9] [Lucas 88] z.B. beschreibt eine transparente Schnittstelle, die beim Zugriff auf eine relationale Datenbank die Datensätze als Prolog-Fakten zur Verfügung stellt. In [Androutsopoulos 92] werden Vor- und Nachteile einer derartigen RDBS-Schnittstelle diskutiert.

Bei der Anwendung der Standardoperationen auf eine oder mehrere gespeicherte Relationen, sog. *Basisrelationen*, entstehen neue, nicht „physisch“ vorhandene Relationen, sog. *virtuelle Relationen* oder *Sichten* (*views*). Anders ausgedrückt, spezifiziert eine Sicht diejenigen Basisrelationen, Operationen und weiteren Sichten, die zur Generierung der gewünschten virtuellen Relation notwendig sind. [Rechenberg und Pomberger 99]

2.3.4 Die Datenmanipulationssprache SQL

Eine formale *Datenmanipulationssprache* (*data manipulation language*, *DML*) wie *SQL*[10] erlaubt es, über das DBMS auf Daten einer relationalen Datenbank zuzugreifen, ohne deren physische Speicherstruktur und Organisation zu kennen. Allerdings benötigt der Datenbankbenutzer Wissen über das Relationenmodell und die Relationenalgebra sowie Kenntnisse im Umgang mit dem Relationenschema, welches das logische Modell der Daten, d.h. der für die Abfrage zur Verfügung stehenden Entitäten und Beziehungen, enthält. Standardisiertes SQL ist hinsichtlich seiner Mächtigkeit relativ eingeschränkt, was sich u.a. folgendermassen manifestiert:

- Der Sprachumfang von SQL ist für die Formulierung allgemeiner Algorithmen nicht ausreichend – SQL enthält z.B. weder Konstrukte für Iteration und Rekursion noch Mechanismen zur Definition von Modulen. Die Lösung für das Problem besteht darin, SQL mit einer allgemeinen Programmiersprache zu kombinieren; sog. *embedded SQL* kann z.B. in einige gängige Programmiersprachen (z.B. C) eingebettet werden. [Rechenberg und Pomberger 99]
- Im Gegensatz zur Prädikatenlogik enthält SQL keinen Allquantor, so dass Allquantifikation (für den Datenbank-Benutzer relativ umständlich und komplex) durch doppelte Negation ausgedrückt werden muss.

Einer SQL-Anfrage an eine relationale Datenbank liegt das folgende Muster zugrunde:

```
SELECT  <welche Attribute?>
FROM    <aus welcher Relation?>
WHERE   <unter welchen Bedingungen?>
```

Man spricht bei obenstehender Anfrage von einer SQL-*Anweisung*, während die *Schlüsselwörter* SELECT, FROM und WHERE je eine entsprechende *Klausel* einleiten: die *SELECT-Klausel*, welche spezifiziert, von welchen Attributen das Anfrageresultat die Werte enthalten soll, die *FROM-Klausel*, welche die Relationen bezeichnet, die zum Ergebnis beitragen oder zu dessen Ermittlung benötigt werden, und die *WHERE-Klausel*, welche die Bedingungen spezifiziert, die Tupel erfüllen müssen, um ins Anfrageresultat aufgenommen zu werden.

SQL unterstützt zusätzlich zu den Standardoperationen der Relationenalgebra die sog. *Aggregatfunktionen*, welche die Berechnung neuer Werte aus zahlartigen Datenbankwerten (SUM für die Summenberechnung, AVG für die Berechnung des Durchschnittswertes, MAX für die Ermittlung des Maximums, MIN für die Ermittlung des Minimums) oder das Zählen bestimmter Tupel (COUNT) durch das DBMS ermöglichen.

Das Resultat einer SQL-Anfrage ist eine (virtuelle) Ergebnisrelation, welche die Attributwerte derjenigen Tupel, welche die WHERE-Bedingungen erfüllen, enthält. Attribute und korrespondierende Werte können dabei aus verschiedenen Basis-Relationen stammen. Falls keine passenden Tupel gefunden werden, gibt das DBMS den NULL-Wert zurück.

[10] Die Sprachbezeichnung bezieht sich meistens auf den Standard *SQL2* bzw. *SQL-92*, der in [Melton and Simon 93] beschrieben ist.

2.4 Alternative Datenbanksysteme

Datenbanksysteme erfüllen in immer zahlreicheren Anwendungsbereichen die Aufgabe der effizienten und sicheren Datenspeicherung und Datenverwaltung. Neue, komplexe Anwendungsbereiche wie CAD, geographische Informationsverarbeitung, Bildverarbeitung und Multimedia, KI und andere verwenden dabei Informationen, die übliche, auf dem Relationenmodell basierende Datenbanksysteme kaum adäquat wiedergeben können. [Elmasri and Navathe 94]

Während in herkömmlichen DBS die Gegenstände der interessierenden Umwelt durch (relativ) wenige, unstrukturierte Eigenschaften (z.B. Attribute mit atomaren Werten) beschrieben werden, arbeiten neue Anwendungen mit umfangreichen, komplex strukturierten Informationen. Den grossen Mengen an uniformen Daten in RDBS stehen komplexe Datenobjekte gegenüber, die sich in ihrer Struktur stark voneinander unterscheiden. Die letzteren verlangen zudem nach flexiblen, komplexen Verarbeitungsmechanismen (z.B. für Datenretrieval, Integritätssicherung usw.) – im Gegensatz zu den vorgegebenen, standardisierten und generischen Operationen relationaler Datenbanken.

Die für die vorliegende Arbeit relevanten „neuen" Typen von Datenbanksystemen basieren entweder auf dem relationalen Datenbankparadigma oder sind zumindest als Lösungsansätze für Probleme, die sich beim relationalen Modell stellen, zu verstehen. Ebenfalls verbreitete Datenbankmodelle wie das Netzwerkmodell oder das hierarchische Modell scheinen im Bereich der natürlichsprachlichen Schnittstellen eine untergeordnete Rolle zu spielen, was sich wohl einerseits mit der Dominanz des relationalen Modells und andererseits mit der erwähnten engen Beziehung zwischen dem relationalen Modells und den Konzepten der in der Computerlinguistik verwendeten logischen Programmierung erklären lässt.

Einen Spezialfall stellt die Kategorie der heterogenen Datenbanksysteme dar, da sie nicht wie in den anderen diskutierten Fällen auf ein spezifisches Datenmodell verweist, sondern eine IT-Infrastruktur beschreibt, die quasi als Meta-Datenbanksystem von den darunter liegenden Datenbanksysteme abstrahiert.

2.4.1 Objektorientierte Datenbanksysteme

Objektorientierte Datenbanksysteme (OODBS) bieten dem Datenbank-Designer die Möglichkeit, sowohl die Struktur komplexer Objekte als auch die Operationen, die auf diese Objekte angewendet werden, zu definieren.

Im Gegensatz zum mengenorientierten relationalen Datenbanksystem besteht die Datenbasis eines OODBS aus einer Sammlung von Datenbankobjekten, welche je ein reales Objekt der interessierenden Welt als Gesamteinheit[11] repräsentieren: physische Gegenstände, Konzepte, Ideen usw. Um die direkte Verbindung zwischen Real-Objekt und Datenbankobjekt aufrechtzuerhalten, vergibt die OODB eindeutige *Objekt-Identifikatoren* (*OID*, *object identifier*). So wird die Integrität der Objekte sichergestellt und ein leichte Adressierbarkeit gewährleistet. Komplexe Datenbankobjekte enthalten ihrerseits (Unter-)Objekte; entsprechende Operatoren ermöglichen den Umgang mit komplex strukturierten, hierarchisch aufgebauten Objekten. Beziehungen zwischen Objekten werden häufig durch Verweise auf Bezugsobjekte, d.h. durch Platzieren von deren OID im Objekt selbst, ausgedrückt. [Elmasri and Navathe 94]

Im OODBS finden u.a. folgende Konzepte der *objektorientierten Programmierung*[12] Anwendung :

- *Kapselung*: Die interne Datenstruktur des Objektes ist von aussen nicht sichtbar; gegen aussen spezifiziert sind alle Operationen, die auf ein Objekt angewendet werden können.

[11] Im Gegensatz zum Relationenmodell, wo Informationen über ein Real-Objekt unter Umständen auf verschiedene Relationen und Tupel verteilt sind, wird das Real-Objekt bzw. dessen Semantik in der OODB durch genau ein Datenbankobjekt repräsentiert.

[12] Vgl. z.B. [Rechenberg und Pomberger 99].

- *Vererbung*: Eine Anforderung an OODBS besteht darin, dass das System die Definition neuer Objekttypen basierend auf vordefinierten Typen erlauben soll. Mit Hilfe des Vererbungsmechanismus, der die Definition eines neuen, abgeleiteten Typs als Deklaration der Abweichung vom bestehenden Basistyp erlaubt, lässt sich eine derartige *Typenhierarchie* einfach, systematisch und vor allem unter Wiederverwendung der bereits bestehenden Typendefinitionen erstellen. [Elmasri and Navathe 94]
- Weitere wichtige Konzepte wie *Polymorphie* sollen hier nicht näher besprochen werden; für eine detaillierte Behandlung sei auf [Elmasri and Navathe 94] verwiesen.

2.4.2 Deduktive Datenbanksysteme

Während objektorientierte Datenbanksysteme die natürliche Modellierung von Real-Objekten betonen, steht bei *deduktiven Datenbanksystemen* (DDBS) die Repräsentation von Beziehungen der realen Welt in Regelform sowie das Ableiten neuen Wissens aus bestehenden Daten unter Verwendung dieser Regeln im Vordergrund. Die formalen Grundlagen deduktiver DBS basieren einerseits auf dem Relationenmodell und andererseits auf der logische Programmierung, wobei letztere den Begriff der „logic database“, häufig synonym zu „deductive database“ verwendet, geprägt hat. [Elmasri and Navathe 94: 730] Deduktive Datenbanken unterscheiden sich unter anderem dadurch von *Expertensystemen* (*wissensbasierten Systemen*) im traditionellen Sinne, dass sie übliche DBMS-Funktionen wie effiziente Verwaltung des Sekundärspeichers und schema-basierte Konsistenzsicherung zur Verfügung stellen.[13]

Gemäss Kapitel **2.3.2** kann man eine relationale Datenbank als logisches Programm betrachten, dessen Fakten weder Variablen noch Funktionen enthalten. Anfragen können entsprechend als „prädikatenlogische Formeln mit mengentheoretischen Ergänzungen“ [Rechenberg und Pomberger 99: 580] behandelt werden. Eine deduktive Datenbank stellt insofern die Erweiterung einer relationalen Datenbank dar, als sie erstens komplexe Terme[14] und zweitens (implikative) Regeln enthalten kann. Ein Inferenz- bzw. Deduktions-Mechanismus kann aus bestehenden Fakten, sog. *Ground Axioms*, neue Fakten ableiten, indem die Regeln, sog. *Deductive Axioms*, interpretiert werden. [Elmasri and Navathe 94: 736]
Fakten sind vergleichbar mit relationalen Tupeln, wobei die Bedeutung eines Attributwerts nur durch seine (Arguments-)Position innerhalb des Tupels und nicht durch einen Attributnamen festgelegt ist. Regeln lassen sich mit relationalen Sichten vergleichen: Sie spezifizieren virtuelle, d.h. physisch nicht vorhandene Mengen von Fakten, die jedoch unter Anwendung von Inferenzmechanismen aus bestehenden Daten hergeleitet werden können. Der Hauptunterschied zwischen Regeln deduktiver DBS und relationalen Sichten liegt darin, dass Regeln *Rekursion* beinhalten und dadurch Mengen von Fakten repräsentieren können, die sich mit virtuellen Relationen nicht darstellen liessen. [Elmasri and Navathe 94]
Anfragen werden in DDBS als *Ziele* (*goals*) behandelt, die es zu beweisen gilt; analog zu den rekursiven Regeldefinitionen lassen sich auch rekursive Anfragen stellen. Da die Standardoperationen der Relationenalgebra direkt auf Regeln der deduktiven Datenbanksprache abgebildet werden können,[15] ist die Spezifizierung einer relationalen Standardanfrage problemlos. [Winiwarter 94]

---

[13] [Elmasri and Navathe 94: 729] spricht vom erweiterten („enhanced“) DBMS respektive vom erweiterten Expertensystem: Ein deduktives DBS ist, einfach formuliert, entweder als DBMS mit Expertensystem-Funktionalität oder als Expertensystem mit Sekundär-Speicher-Management konzipiert.
[14] Allerdings nicht in der deduktiven Datenbanksprache Datalog.
[15] Für Datalog-Beispiele vgl. [Elmasri and Navathe 94: 743-744].

In der deklarativen Datenbanksprache *Datalog*, einer Untermenge von Prolog, dessen operationale Semantik nicht festgelegt ist,[16] werden die Fakten und Regeln als Horn-Klauseln repräsentiert:

```
supervise(franklin, john).  % Faktum

superior(X, Y) :-           % Regel (rekursiv)
        supervise(X, Z),
        superior(Z, Y).
```

Während Datalog primär zu Forschungszwecken eingesetzt wird, finden deduktive Datenbanksprachen wie z.B. *LDL* (*Logical Data Language*), welche hinsichtlich der Unterstützung typischer DBMS-Funktionen wie Datenmodellierung, Datenspeicherung und Anfrageoptimierung wesentlich ausgereifter sind, auch im kommerziellen Umfeld Verwendung. [Elmasri and Navathe 94]

2.4.3 Heterogene Datenbanksysteme

Ein *heterogenes Datenbanksystem* vereinigt eigenständige Datenbanksysteme zu einem *Multi-Datenbanksystem* mit einheitlicher Schnittstelle und ermöglicht die Kooperation zwischen den *Komponentensystemen*, die unter Umständen auf unterschiedlichen Daten(bank)modellen basieren. Externe Anwendungen profitieren vom vereinheitlichten Zugriff auf Daten verschiedenster Provenienz, ohne dass dabei Kenntnis der Struktur von Komponentensystemen notwendig wäre. [Rechenberg und Pomberger 99]

2.5 Trends im Datenbankbereich

Die folgenden Unterkapitel enthalten kurze Exkurse in zwei IT-Anwendungsbereiche, die zunehmend an Bedeutung gewinnen, intensiven Gebrauch von Datenbanktechnologie machen und massgeblich von ausgereiften Benutzerschnittstellen profitieren (könnten).

2.5.1 Data Warehousing

Hinter dem Begriff *Data Warehousing* steckt die Absicht, implizite Informationen aus bestehenden operativen Datenbanken zu erschliessen. Im Gegensatz zum traditionellen Ansatz zur Integration von Informationen aus heterogenen Datenquellen, dem *Query-Driven Approach*, werden Daten beim Data Warehousing nicht auf Anfrage gesammelt und kombiniert, sondern im Voraus in ein *Data Warehouse* kopiert und für direkte Befragung und Analyse zur Verfügung gestellt. Neben der erhöhten Performance bietet dies den Vorteil, dass lokale Vorgänge bei den Datenquellen nicht beeinträchtigt werden.

Data-Warehousing-Technologie wird vor allem für Datenanalyse und Entscheidungsunterstützung unter Berücksichtigung der zeitlichen/historischen Dimension eingesetzt. Unter den Stichworten *OLAP* (*On-Line Analytical Processing*) und *Data Mining* fasst man jene Applikationen zusammen, welche auf die im Data-Warehouse gespeicherten Informationen zugreifen.

Verglichen mit einer („operativen“) Standard-Datenbank enthält das Data Warehouse weitaus grössere Datenmengen (im Gigabyte- bis Terabyte-Bereich), die sich aufgrund temporaler Attribute in einen zeitlichen Ablauf einordnen lassen. [Hammer 98]

---

[16] In anderen Worten: dessen Abarbeitungsstrategie im Gegensatz zu Prolog nicht festgelegt ist. Die Optimierung der Anfrage-Verarbeitung, die effiziente Kombinierung von Top-Down- (backward chaining) und Bottom-Up-Verarbeitung (forward chaining) nimmt entsprechend einen wichtigen Stellenwert in der Entwicklung eines DDBS ein.

Die ideale Benutzerschnittstelle zu einem Data Warehouse kann sowohl von einem Analytiker als auch einem Entscheidungsträger bzw. Manager effektiv und effizient eingesetzt werden. Während wiederkehrende Informationsbedürfnisse mit vordefinierten und standardisierten Abfragen oder Berichten befriedigt werden können, erfordert das dynamische Geschäftsumfeld häufig flexiblere, möglichst frei konfigurierbare Abfragemechanismen. Dazu kommt, dass die Benutzerschnittstelle aufgrund der zu erwartenden grossen Datenmengen sowie der strukturellen Komplexität des Data Warehouse auch aufwändige Abfragen ermöglichen muss, wobei idealerweise zumindest ein Teil der Komplexität vor dem Benutzer versteckt wird.

2.5.2 Datenbanktechnologie im WWW

Im *World Wide Web* (*WWW*) spielt Datenbanktechnologie in zweierlei Hinsicht eine Rolle: Erstens bilden eigentliche Datenbanken, z.B. relationale, objekt-orientierte oder deduktive Datenbanken, die Basistechnologie für die Speicherung derjenigen Informationen, welche im Internet „angezeigt" werden. Zweitens erweisen sich die bestens erforschten und bewährten Datenbanktechnologien zum Umgang mit sehr grossen Datenmengen auch als geeignet für bestimmte Aufgaben des Informationsmanagements im WWW, d.h. für den Umgang mit Datenbank-ähnlichen Strukturen bzw. mit Sammlungen strukturierter Daten, wobei nicht selten auch in diesem Fall im Hintergrund Datenbanken ihren Dienst verrichten.
[Florescu et al. 98] unterscheidet drei Kategorien von Aufgaben des Informationsmanagements im WWW, bei denen Datenbanken oder zumindest verwandte Technologien eine Rolle spielen:
- *Modellierung und Befragung des WWW*. In diese Kategorie gehört z.B. die Inhalts- oder Linkstruktur-basierte Suche nach Inhalten bzw. Seiten im WWW mit Hilfe einer *Suchmaschine*, wobei das WWW als gerichteter Graph betrachtet wird, aus dem einzelne Knoten „wiedergewonnen" werden.
- *Informationsextraktion und Informationsintegration*. Seiten des WWW können in vielen Fällen als Benutzerschnittstellen zu einer (meistens heterogenen) Datenbank betrachtet werden, so z.B. die unter www.imdb.com abrufbare Startseite des grossen Filme-Portals. Der Benutzer kann Anfragen stellen, indem er z.B. ein Anfrage-Formular ausfüllt und abschickt. Ein sog. *Daten-Integrationssystem* (*data integration system*) integriert Daten aus verschiedenen Datenbanken, aus strukturierten Dateien und u.U. aus anderen Web-Seiten, um anhand dieser Datenquellen die Anfrage des Benutzers beantworten zu können. Der Unterschied zu einem heterogenen DBS besteht vor allem in der hohen Anzahl der Datenquellen, im Mangel an Metadaten zu den Datenquellen und im höheren Autonomie-Grad der einzelnen Datenquellen.
- *Konstruktion und Restrukturierung einer Website*. Komplexere Websites kann man sich als Schnittstellen[17] zu Daten in (potentiell) heterogenen Datenbanken vorstellen. Der Bau einer solchen Site umfasst also Auswahl der darzustellenden Daten, Schaffen einer Zugangsmöglichkeit zu den Daten, Entwurf der Site-Struktur sowie Entwurf der graphischen Präsentation der Site.

Sowohl bei der Befragung des WWW als auch bei der Web-basierten Informationsextraktion aus integrierten Datenquellen trägt die Qualität der Benutzerschnittstelle wesentlich zur Benutzerakzeptanz bei. Soll der ungeübte Benutzer die Informationsangebote regelmässig nutzen, so wird man davon absehen müssen, von ihm allzu komplexe[18] formalsprachliche Anweisungen zu verlangen.
Spezielle Aufmerksamkeit verdient auch der Umstand, dass dank der Datenquellen-Integration eine einheitliche Benutzerschnittstelle zwar möglich (und wünschenswert) ist, diese Universalität jedoch bei der Auswahl bzw. beim Entwurf des Kommunikationsmittels zwingend berücksichtigt werden muss.

---

[17] Im Sinne einer Schnittstelle „gegen innen", nicht „gegen aussen" für den Benutzer.
[18] Präzise Anfragen an Suchmaschinen, z.B. mit verschachtelten logischen Operatoren, übersteigen vermutlich die Kompetenz der meisten Benutzer.

3 Natürlichsprachliche Schnittstellen zu Datenbanken

In den folgenden Unterkapiteln soll erstens der Begriff der natürlichsprachlichen Datenbankschnittstelle eingeführt und zweitens anhand der Erläuterung verschiedener damit verbundener Aspekte und Konzepte ein erster Eindruck der interdisziplinären Relevanz des Themas vermittelt werden.

3.1 Natürlichsprachliche Retrieval-Schnittstellen

Der Begriff der *natürlichsprachlichen Schnittstelle* (*natural language interface*, *NLI*) bezeichnet eine Benutzerschnittstelle, welche den Benutzer dazu befähigt, mittels natürlicher Sprache Prozesse zu steuern und auf Daten zuzugreifen. Das Konzept der „natürlichen Sprache" ist hier primär im Sinne einer Abgrenzung gegenüber „künstlichen", d.h. formal eindeutig definierten Sprachen zu verstehen. In den meisten Fällen meint man mit „natürlicher Sprache" eine Untermenge der unbeschränkten natürlichen Sprache.

Mit dem Unterbegriff *natürlichsprachliche Retrieval-Schnittstelle* bezieht man sich auf diejenigen NLI, welche als Benutzerschnittstellen zu Informationssystemen dem einfachen, natürlichsprachlichen Zugriff auf Datenbestände dienen. Auch wenn die eindeutige Zuordnung realer Systeme selten möglich ist, so lassen sich doch drei Typen von natürlichsprachlichen Retrieval-Schnittstellen unterscheiden:[19]

- *Schnittstellen zu Information-Retrieval-Systemen*. Beim Zugriff auf umfassende unstrukturierte Datenbestände in IR-Systemen stellt die Abbildung der Anfrage auf die Textdokumente die primäre Herausforderung dar. Insofern resultiert der beträchtliche Entwicklungsaufwand nicht nur aus der Verarbeitung der natürlichsprachlichen Anfrage, sondern vor allem aus dem „Retrieval" selbst.[20]
- *Schnittstellen zu Expertensystemen* bzw. *wissensbasierten Systemen*. Sog. „natural language (understanding) systems" oder „question answering systems" kombinieren natürlichsprachliche Verarbeitung, Inferenzfunktionalität und Wissensbasis zu einem integrierten System. Die natürlichsprachliche Retrieval-Schnittstelle *ist* in diesem Sinne das Expertensystem.
- *Schnittstellen zu Datenbanken*. Datenbanksysteme stellen im Gegensatz zu Expertensystemen ausgereifte Funktionen der Zugriffssteuerung und Datenmanipulation zur Verfügung und profitieren gegenüber IR-Systemen von der strukturierten Ablage der Daten. Dank dieser soliden Basis muss sich die natürlichsprachliche Komponente im Prinzip nicht um die eigentliche Ausführung einer Anfrage kümmern, sondern kann sich auf die Analyse der Eingabe und die Übersetzung in eine Datenbank-gerechte Repräsentation konzentrieren.

3.2 Natürlichsprachliche Datenbankschnittstellen

> It is fair to say that databases will play a critical role in almost all areas where computers are used, including business, engineering, medicine, law, education, and library science, to name a few. [Elmasri and Navathe 94: 1]

Mit der zunehmenden Verbreitung kommerzieller Datenbanksysteme steigt das Bedürfnis nach einer einheitlichen und ohne Vorkenntnisse bedienbaren Retrieval-Schnittstelle. Letzteres impliziert, dass (1) der Benutzer ohne jegliche Kenntnis einer formalen Abfragesprache seine Informationsbedürfnisse formulieren

[19] Vgl. [Winiwarter 94].
[20] Für einen Überblick über aktuelle Tendenzen im Bereich des Information Retrieval bzw. der Volltextsuche vgl. z.B. [Carstensen et al. 01: 425-441].

kann, dass (2) der Benutzer keine Kenntnis der Datenbank-Struktur benötigt und dass (3) der Benutzer trotz (1) und (2) seine Informationsbedürfnisse einfach, präzise und effizient formulieren kann.

Menü- und Formular-basierte sowie graphische Datenbank-Benutzerschnittstellen zu relationalen Datenbanken (vgl. Kapitel **2.2**) stellen Versuche dar, das vom Benutzer aufzubringende Wissen über formalsprachliche Syntax und Semantik im Vergleich zum direkten SQL-Retrieval zu minimieren. Allerdings muss der Benutzer auch in diesem Fall die logische Datenbankstruktur und zumindest die Grundlagen der SQL-Semantik kennen, um z.B. die Wirkungsweise eingebauter Funktionen für Datums- und Zeitverarbeitung zu verstehen und um verschachtelte Boolesche Auswahlen formulieren zu können. Dazu kommt, dass graphische Schnittstellen selten verschachtelte Anfragen[21] unterstützen, was u.U. die manuelle Nachbearbeitung der Anfrage erfordert, und dass sie explizite Auswahl und Verbindung aller benötigten Relationen verlangen, was effektiv Kenntnisse der Datenbankstruktur voraussetzt. [Harris 98]

Ursprünglich als Ersatz für „künstliche“ Retrieval-Formalismen ist die „natürliche“ Sprache sog. *natürlichsprachlicher Datenbankschnittstellen* (*natural language interface to databases*, *NLIDB*)[22] gedacht. Gerade im Zusammenspiel mit Datenbankanwendungen scheint sie tatsächlich ein effektives und mächtiges Kommunikationsmittel darzustellen, das auch gegenüber den GUI-basierten Alternativen einige prinzipielle Vorteile bietet:
(1) Universalität: Der Benutzer arbeitet mit einem universellen Kommunikationsmittel, das prinzipiell für verschiedene Datenbankstrukturen, Wissensdomänen und Anwendungsbereiche eingesetzt werden kann.
(2) Natürlichkeit: Der ungeschulte Benutzer, der u.U. sogar mit Formulareingaben Mühe bekundet, kann auf ein natürliches, vertrautes Kommunikationsmittel zurückgreifen. Abgesehen davon, dass die „künstliche“ Barriere wegfällt, wird sowohl der Schulungsaufwand als auch die Einarbeitungszeit wesentlich reduziert.
(3) Flexibilität: Derselbe Sachverhalt kann in natürlicher Sprache verschieden dargestellt werden; entsprechend kann der Benutzer seine Bedürfnisse auf verschiedene Art und Weise formulieren.
(4) Mächtigkeit: Für bestimmte Selektionen und Operationen ist die natürliche Sprache besonders geeignet: So lassen sich Negation („Which department has no programmers?“) oder Allquantifikation („Which company supplies every department?“) nur bedeutend umständlicher mit Hilfe von formalen Abfragesprachen oder gar Formular-basierten oder graphischen Benutzerschnittstellen formulieren.[23]
(5) Effektivität: Natürlichsprachliche Anfragen sind in vielen Fällen wesentlich einfacher als deren formalsprachliche Entsprechungen. „Who bought high margin products but not extended service plans?“ entspricht z.B. der folgenden SQL-Anfrage:[24]

```
SELECT [CUSTOMERS].[LAST_NAME] as [Last Name],
[CUSTOMERS].[FIRST_NAME] as [First Name], [CUSTOMERS].[COMPANY_NAME]
as [Company Name]
FROM [CUSTOMERS]
WHERE exists(SELECT [CUSTOMERS].[LAST_NAME],
[CUSTOMERS].[FIRST_NAME], [CUSTOMERS].[COMPANY_NAME], [Product Name]
FROM [ORDERS], [LINE ITEMS], [PRODUCTS] WHERE
([Product_Margin]='High') and [CUSTOMERS].[CUST ID] = [ORDERS].[CUST
ID] and [ORDERS].[ORDER_ID] = [LINE ITEMS].[ORDER_ID] and
[PRODUCTS].[Product ID] = [LINE ITEMS].[PRODUCT_ID]) and
not exists(SELECT [CUSTOMERS].[LAST_NAME], [CUSTOMERS].[FIRST_NAME],
```

[21] Bei einer verschachtelten Anfrage enthält die WHERE-Klausel weitere SELECT-Anweisungen.
[22] [Binot 91] verwendet den Begriff der „dialog interfaces“, was insofern etwas irreführend ist, als nicht jedes NLIDB in der Lage ist, einen effektiven Dialog zu führen.
[23] Vgl. [Androutsopoulos et al. 95].
[24] Vgl. [Harris 98].

```
[CUSTOMERS].[COMPANY_NAME], [Product Name] FROM [ORDERS], [LINE
ITEMS], [PRODUCTS] WHERE ([Product Name]='Extended Service') and
[CUSTOMERS].[CUST ID] = [ORDERS].[CUST ID] and [ORDERS].[ORDER_ID] =
[LINE ITEMS].[ORDER_ID] and [PRODUCTS].[Product ID] = [LINE
ITEMS].[PRODUCT_ID])
```

(6) Diskursfähigkeit: Natürliche Sprache ist diskursfähig, d.h. sie enthält Mechanismen, um satzübergreifend – z.B. mit kurzen, syntaktisch unvollständigen und semantisch unterspezifizierten Sätzen – effizient und kohärent zu kommunizieren. Der Diskurskontext wird von formalen Abfragesprachen und Formular-basierten/graphischen Schnittstellen gewöhnlich nicht verarbeitet. [Androutsopoulos et al. 95]

Ein NLIDB ermöglicht den Einsatz natürlicher Sprache als effektives Kommunikationsmittel zwischen Benutzer und Datenbanksystem. Als Schnittstelle stellt es diejenigen Mechanismen zur Verfügung, die für eine Übersetzung natürlichsprachlicher Ausdrücke in anwendungsspezifische, für das DBMS „verständliche“ Anweisungen benötigt werden.
Pointiert ausgedrückt, wird der Aufwand, welcher beim Lernen einer formalen Abfragesprache anfällt, durch den Aufwand ersetzt, welcher beim „Trainieren“ der Schnittstelle, so dass sie natürlichsprachliche Sätze „versteht“, anfällt. Ist die Schnittstelle einmal konfiguriert, so kann auch der Nicht-Experte das Datenbanksystem effizient nutzen.

3.3 Fortschritte der NLIDB-Forschung

Der folgende historische Überblick über das Gebiet der natürlichsprachlichen Datenbankschnittstellen soll das breite Spektrum von NLIDB-Systemen in einen zeitlichen Kontext stellen und so die Basis für eine einfache NLIDB-Typologie bilden. (Angaben zur Funktionalität und bibliographische Hinweise sind in vielen Fällen [Androutsopoulos et al. 95], [Wu et al. 99] und/oder [Manaris 98] entnommen.)

3.3.1 Prototypen-Phase

In den späten 1960ern und frühen 1970er-Jahren werden vor allem NLIDB-Prototypen entwickelt, die auf spezifische Datenbanken zugeschnitten und praktisch nicht portierbar sind. Der bekannteste Vertreter ist wohl LUNAR[25] von BBN, das eine natürlichsprachliche Benutzerschnittstelle zu einer Datenbank mit chemischen Analysen von Mondgestein zur Verfügung stellt.
Während der 1970er-Jahre versucht man, vermehrt dem Anspruch der Benutzerfreundlichkeit zu genügen: RENDEZVOUS [Codd 74] z.B. hilft durch ein Dialogmodul dem Benutzer, seine Anfrage zu formulieren, während LADDER [Hendrix et al. 78] mit grossen Datenbanken verwendet und als Schnittstelle zu verschiedenen DBMS konfiguriert werden kann. Weitere NLIDB-Systeme der 1970er-Jahre sind z.B. ROBOT [Harris 77], LIFER [Hendrix et al. 78], PLANES [Waltz 78] und PHILIQUA1 [Scha 77].
CHAT-80 [Warren and Pereira 82] bildet in den frühen 1980er-Jahren Ausgangspunkt für diverse andere Projekte, so z.B. MASQUE. Vollständig in Prolog implementiert, übersetzt CHAT-80 englische Anfragen in Prolog-Ausdrücke, die in der Folge als Anfragen an die Prolog-Datenbank dienen.

3.3.2 Front-End-Phase

Mitte der 1980er-Jahre sind NLIDB-Systeme zu einem beliebten Forschungsgebiet geworden, wobei das Hauptaugenmerk meistens auf dem Kriterium der Portabilität liegt. Während die frühen NLIDB-Systeme

[25] Vgl. [Woods et al. 72], [Woods 73].

auf einzelne, spezifische Datenbanken und deren Anwendungsdomänen fixiert sind, fordert man nun linguistische Front-End-Systeme, die mit vernünftigem Kostenaufwand an neue Datenbanken bzw. Anwendungsdomänen angepasst werden können. Beispiele für portierbare NLIDB-Systeme sind IRUS [Bates et al. 86], TEAM [Grosz et al. 87], CO-OP [Kaplan 84] und KID [Ishikawa et al. 87].
Das Laborsystem MASQUE („Modular Answering System for QUeries in English“)[26] stellt einen Versuch des Department of Artificial Intelligence der University of Edinburgh dar, CHAT-80 in ein portableres und effizienteres System zu überführen. MASQUE ist in SICStus Prolog implementiert, wobei das System in seiner ursprünglichen Form auf eine proprietäre Prolog-Datenbank (MASQUE/PRO) und erst in einer überarbeiteten Version (MASQUE/SQL)[27] auf relationale Datenbanken wie z.B. Ingres zugreift. Eine MASQUE/SQL-ähnliche NLIDB-Architektur beschreibt Kapitel **4.1**.
IRUS, (wie LUNAR) von BBN entwickelt, ist ein wissensbasiertes System, deren prozedurale Komponenten unabhängig und deren Wissens- bzw. Daten-Komponenten abhängig von Datenbankstruktur und Anwendungsdomäne sind. Die Wissensbasis lässt sich mit Hilfe eines spezifischen Moduls anpassen und erweitern.
Das Menü-basierte Werkzeug zur Wissenserwerbung von TEAM (Transportable English database Access Medium) soll die System-Konfiguration durch einen Datenbank-Administrator, der weder Kenntnis vom inneren Systemaufbau noch eine Ahnung von Syntaxverarbeitung oder Wissensrepräsentation besitzt, ermöglichen.
CO-OP unterstützt den Benutzer mit kooperativen Antworten und bezieht das Domänenwissen primär aus Datenbankstruktur und Datenbankinhalt.
ASK [Thompson and Thompson 85] erlaubt es dem Benutzer, während der Interaktion mit dem System neue Wörter und Konzepte in die Wissensbasis einzufügen. Das System stellt sowohl eine eigene interne Datenbank als auch Schnittstellen zu diversen externen Datenbanken (und anderen Applikationen wie E-Mail-Programmen) zur Verfügung. Natürlichsprachliche Anfragen ermöglichen die Interaktion mit sämtlichen angebundenen Anwendungen, wobei das System die passenden formalen Anfragen generiert.
Das System JANUS[28], entwickelt von BBN, bietet ähnliche Möglichkeiten, mehrere angebundene Subsysteme anzusprechen (z.B. Datenbanken, Expertensysteme, Graphikmodule). Die Anfrage kann von verschiedenen Modulen verarbeitet werden, ohne dass der Benutzer die Heterogenität des Gesamtsystems bemerkt.
Daneben erscheinen Mitte der 80er-Jahre u.a. Systeme wie DATALOG[29], EUFID [Templeton and Burger 83], LDC [Ballard et al. 84], TQA[30] und TELI [Ballard and Stumberger 86].

3.3.3 Benutzer-zentrierte Phase

Die erste Hälfte der 1990er-Jahre prägt eine Reihe kommerziell erhältlicher NLIDB-Systemen, nicht selten Abkömmlinge von Labor-Systemen der Prototypen- und Front-End-Phase:
- INTELLECT von Trinzic. Das System basiert auf den mit ROBOT gemachten Erfahrungen und dient dem Benutzer als Schnittstelle zu (mehreren) darunter liegenden Datenbanken oder, je nach Konfigurierung, zur Experten-System-Konsole KBMS. INTELLECT wird nicht mehr unterstützt, hat jedoch in ENGLISH WIZARD und EASYASK (Kapitel **6.4.2**) Nachfolger gefunden.
- BBN's PARLANCE, das u.a. auf Erfahrungen mit IRUS basiert. PARLANCE wird nicht mehr unterstützt.[31]

---

[26] Vgl. [Auxerre 86], [Auxerre and Inder 86].
[27] Vgl. [Androutsopoulos 92], [Androutsopoulos et al. 93].
[28] Vgl. [Resnik 89], [Weischedel 89].
[29] Vgl. [Hafner 84], [Hafner and Godden 85]. Zwischen dem NLIDB DATALOG und der deduktiven Datenbanksprache Datalog besteht kein direkter Zusammenhang.
[30] Vgl. [Damerau 81], [Damerau 85].

- IBM's LANGUAGEACCESS [Sanamrad and Bretan 92], ein NLI zu relationalen Datenbanken, welches Englisch und Deutsch unterstützt [Androutsopoulos 92] und bis Oktober 1992 kommerziell erhältlich ist.
- Q&A von Symantec. Gemäss [Dahl et al. 00] kann Q&A nur mit einer proprietären „Flat-Database" verwendet werden. Q&A wird nicht mehr unterstützt.[32]
- LOQUI von BIM. BIM entwickelt auf der Basis der im Esprit-Projekt LOKI [Binot et al. 88] bis Ende der 1980er-Jahre geschaffenen NLIDB-Technologie das kommerzielle Produkt LOQUI. Das NLIDB unterstützt die Sprachen Englisch, Französisch, Holländisch, Deutsch und läuft auf der Sun-Plattform (SPARCstation). Implementiert in BIM Prolog, kommuniziert LOQUI nicht über SQL, sondern über die proprietären Schnittstellen der Programmier-Plattform BIM Prolog mit relationalen Standard-DBMS.[33] (Gemäss persönlicher Angaben von David Sedlock, einem der Autoren von LOQUI, ist das System seit geraumer Zeit nicht mehr erhältlich.)
- NATURAL LANGUAGE von Natural Language Inc.

Allerdings können diese Systeme, zumal sie meistens schnell wieder vom Markt verschwunden sind, nicht darüber hinweg täuschen, dass die erhoffte Akzeptanz ausgeblieben ist und dass NLIDB-Systeme schon gar nicht als „standard option" bei DBMS bezeichnet werden können. [Androutsopoulos et al. 95: 3]

Der Begriff der „Benutzer-zentrierten" Phase [Manaris 98] deutet auf einen eigentlichen Paradigmenwechsel hin: Während der wissenschaftliche Fokus bis anhin primär auf der Entwicklung portabler Systemarchitekturen sowie adäquate(re)r Strategien der natürlichsprachlichen Analyse und Generierung gelegen hat, rückt nun zusehends der Aspekt des Benutzers und seines Verhaltens bei der Mensch-Maschine-Kommunikation ins Blickfeld. Dies resultiert unter anderem in Versuchen, verschiedene Typen von Datenbank-Benutzerschnittstellen zu kombinieren, um so deren individuellen Nutzen zu erhöhen.

3.4 Perspektiven der NLIDB-Forschung

Wie bereits angedeutet, findet die Forschungsdiskussion über NLIDB-Systeme auf verschiedenen Ebenen statt. Die folgenden Kapitel sollen unterschiedliche Perspektiven der Forschung aufzeigen, zentrale Begriffe und Beurteilungskriterien einführen sowie allgemeine Beobachtungen bezüglich NLIDB-Systeme zum Ausdruck bringen.

3.4.1 Natural Language Processing

Aus computerlinguistischer Perspektive sind primär die Fähigkeiten des NLIDB-Systems im Bereich des *Natural Language Processing* (*NLP*) interessant. So stellt sich die Frage nach der linguistischen und konzeptuellen Abdeckung, d.h. nach derjenigen Untermenge der natürlichen Sprache, welche das System korrekt analysieren und u.U. generieren kann. Damit verbunden ist auch die Frage nach den beim NLP eingesetzten Algorithmen und Formalismen – wegen des weiten Fokus der vorliegenden Arbeit wird dieser Aspekt allerdings nur am Rande behandelt werden.

Das Ziel der Analyse besteht darin, den natürlichsprachlichen Eingabesatz in eine oder mehrere anwendungsspezifische, vom DBMS evaluierbare Anweisungen, welche die gewünschten Datenzugriffs- oder Datenmanipulationsoperationen spezifizieren, zu übersetzen. Natürlichsprachliche Generierung kommt vor allem bei Antwort- und Feedback-Mechanismen zum Einsatz.

---

[31] Die 1996 von BBN und Amerscan gegründete Parlance Corporation entwickelt mittlerweile Telefondienste mit Spracherkennungs-Technologie. Link: [Parlance Corp.].

[32] Link: [Q&A FAQ].

[33] Vgl. [LIM 92].

Das Spektrum der Komplexität des NLP bei NLIDB-Systemen reicht von einfachen Pattern-Matching-Ansätzen [Androutsopoulos et al. 95] über elaboriertes *Sprachverstehen* (*Natural Language Understanding*, *NLU*) bis hin zur Diskurs- und Pragmatikanalyse. Man unterscheidet analog zu den drei Phasen des historischen Überblicks gewöhnlich drei Generationen von NLIDB, welche je durch spezifische Merkmale der Systemarchitektur sowie durch typische Verfahren zur Analyse und Generierung charakterisiert sind:

- *Systeme zur direkten Übersetzung der Anfrage*. Frühe NLIDB-Systeme wie z.B. LUNAR übersetzen die natürlichsprachliche Anfrage unter Verwendung meist proprietärer syntaktischer oder semantischer Grammatiken direkt in einen anwendungsspezifischen Abfrage-Formalismus. Sie greifen typischerweise auf integrierte, proprietäre Datenbanken zu, deren Datenbanksprache die Abbildung der syntaktischen/semantischen Struktur auf die Datenbankabfrage möglichst vereinfachen soll.
- *Front-End-Systeme*. Front-End-Systeme wie z.B. MASQUE/SQL (vgl. Kapitel **4**) übersetzen die natürlichsprachliche Anfrage zunächst in eine domänenunabhängige, meistens logische Zwischenrepräsentation und generieren daraus den anwendungsspezifischen Abfrage-Formalismus. Sie verwenden im Vergleich zu den Systemen der ersten Generation eine zusätzliche Wissensbasis, welche das domänenspezifische (linguistische und konzeptuelle) Wissen enthält und als *Domänenmodell* bezeichnet wird. Viele Front-End-Systeme sind als Schnittstellen zu relationalen SQL-Datenbanken, d.h. zu „Real-World"-Datenbanken konzipiert. Meistens funktionieren sie als Frage-Antwort-Systeme, wenn auch ausgereiftere Front-End-Systeme bereits über ein einfaches Diskursmodell im Sinne eines Diskursgedächtnisses verfügen.
- *Dialogsysteme*. Zusätzlich zum Domänenmodell enthalten NLIDB-Systeme der dritten Generation sowohl ein komplexes *Diskursmodell* als auch ein *Benutzermodell*. Die zentrale *Dialogsteuerung* übernimmt die Funktion einer Schaltzentrale zwischen den Komponenten des Sprachverstehens und den Komponenten der Sprachgenerierung und löst je nach Analyseergebnis eine passende Systemreaktion aus (vgl. Kapitel **6.1**). Dialogsysteme zeichnen sich häufig durch Unterstützung gesprochener Sprache in Eingabe und Ausgabe aus (*speech processing*) und finden entsprechend z.B. bei Telefon-basierten Auskunftssystemen Verwendung.

Da NLIDB-Systeme generell nur eine Untermenge der natürlichen Sprache verarbeiten können, trachtet man danach, dem Benutzer bereits bei der Formulierung der Anfrage Einschränkungen zu auferlegen. Diese sollen der Komplexität der natürlichsprachlichen Eingabe systemspezifische Grenzen setzen und dadurch eine möglichst effiziente Verarbeitung möglichst wohlgeformter Ausdrücke gewährleisten. So verwendet z.B. LOQUI sog. *kontrollierte* Sprache, wobei der Benutzer zwar vollständige, natürlichsprachliche Sätze formulieren kann, diese jedoch gemäss der proprietären und relativ limitierten Grammatik syntaktisch wohlgeformt sein müssen.[34] PRE akzeptiert nur Anfragen, die nach dem folgenden syntaktischen Muster geformt sind [Androutsopoulos et al. 95]:

what *conjoined noun phrases* *nested relative clauses* *conjoined relative clauses*

Menü-basierte natürlichsprachliche Schnittstellen wie NLMENU schränken den Benutzer noch stärker ein, indem sie sowohl erlaubte syntaktische Konstrukte als auch die zur Verfügung stehenden Konzepte in einer Auswahlliste vorgeben.[35]

[34] Einem kommerziellen Produkt angemessen, enthalten die „User Guidelines" zu LOQUI neben diversen Beispielen zur linguistischen Abdeckung des Systems die vollständige BNF-Grammatik von „English for Querying", der verwendeten kontrollierten Sprache. Vgl. [LOQUI 91].
[35] Vgl. [Androutsopoulos et al. 95], [Hall et al. 96].

3.4.2 Implementierung und Portabilität

Aus einer „Software Engineering"-orientierten Perspektive lassen sich folgende allgemeine Aussagen zu Architektur und Implementierung von NLIDB-Systemen machen:

- Viele bedeutende NLIDB sowohl der Prototypen- als auch der Front-End-Phase, so z.B. CHAT-80 und seine Abkömmlinge, sind (häufig sogar exklusiv) in Prolog implementiert. Dies hängt einerseits mit dessen Eignung für NLP- bzw. generell deduktive Aufgaben [Binot 91] und andererseits mit der bereits erwähnten (Kapitel **2.3.2**) engen Beziehung zwischen der Horn-Klausel-Logik und dem Relationenmodell zusammen.
- Spätestens seit der Front-End-Phase gilt *Portabilität* als wichtiges Qualitätskriterium für NLIDB-Systeme. Je „portabler" ein NLIDB ist, desto einfacher kann es in verschiedene Hardware- und Softwareumgebungen versetzt bzw. für verschiedene Plattformen, DBMS, Datenbanken und vor allem Anwendungsdomänen konfiguriert werden, und desto breiter ist folglich das mögliche Einsatzspektrum.

Portabilität stellt insofern einen wichtigen Faktor für die Kosteneffektivität eines NLIDB-Systems dar, als sowohl Entwicklung wie auch Domänenkonfigurierung teuer sind und als ein allgemein einsetzbares System, welches uneingeschränkte natürliche Sprache verarbeitet und nicht konfiguriert werden muss, an der Komplexität des erforderlichen Sprachverstehens scheitert und vorläufig „Zukunftsmusik" bleibt. Zur Minimierung des Konfigurierungs- und Portierungsaufwandes kommen häufig dedizierte Konfigurierungsprogramme sowie halb- und vollautomatische Verfahren der Wissensakquirierung zum Einsatz; die sog. *Front-End-Architektur* (Kapitel **4**) hat sich unter dem Aspekt der Portabilität als besonders geeignet erwiesen.

3.4.3 Mensch-Maschine-Kommunikation

Der Begriff der *Mensch-Maschine-Kommunikation* (*MMK*, englisch *human-computer interaction*, *HCI*) bezeichnet im Allgemeinen „die interaktive Bedienung des Computers durch den Benutzer und alles, was mit der Entwicklung interaktiver Programme zu tun hat", und im Spezifischen ein interdisziplinäres Fach der Informatik, das Technik und Ergonomie von Mensch-Maschine-Schnittstellen untersucht. [Rechenberg und Pomberger 99: 791]

Vor allem seit den späten 1980er-Jahren hat die Perspektive der MMK der computerlinguistischen Forschung im Bereich der natürlichsprachlichen Schnittstellen wertvolle Impulse vermitteln können. Sie ist ausschlaggebend für den Übergang von der älteren, tendenziell System-orientierten Betrachtungsweise zur neueren, eher Benutzer-orientierten Betrachtungsweise. Betont die erstere technische Aspekte wie Systemportabilität und NLP, so steht bei letzterer die dem Benutzer angemessene Interaktion mit dem System im Zentrum. Die Maxime lautet, Systeme zu entwickeln, welche die physischen, perzeptiven und kognitiven Fähigkeiten des Benutzers ergänzen und falls möglich verbessen. [Manaris 98]

Gewöhnlich werden im Zusammenhang mit der Anwendung von NLIDB-Systemen einige der folgenden Benutzerrollen erwähnt:

- Der *Programmierer*. Der mit der Rolle des Programmierers assoziierte Anwendungsfall besteht vor allem in der Konfigurierung und/oder Portierung einzelner Module und Komponenten.
- Der *Wissensingenieur* (*knowledge engineer*) ist primär für die Konfigurierung des Domänenmodells zuständig. Im Gegensatz zum Programmierer verfügt er nicht über Programmierkenntnisse, wohl aber über Kenntnis der linguistischen Konzepte, Kenntnis der Wissensrepräsentations-Techniken sowie (grundlegende) Kenntnis der Datenbanktheorie.
- Der *Datenbank-Administrator* (*DBA*) verwaltet die zugrunde liegende(n) Datenbank(en) und ist z.B. für die Vergabe von Benutzerrechten zuständig. Er besitzt primär fundierte Kenntnisse der Datenbanktheorie. Die Entwickler von TEAM skizzieren ein einleuchtendes, wenn auch nicht dem typischen Anwendungsfall

entsprechendes Szenario, wonach ein NLIDB vom DBA an eine bereits im Einsatz stehende Datenbank angebunden wird. In diesem Fall ist der DBA für die Domänenkonfigurierung zuständig, was hohe Ansprüche an die dazu benötigten Werkzeuge stellt. [Wu et al. 96]
- Der *Endbenutzer* (*end user*) „benutzt" die Datenbank und damit die Schnittstelle im eigentlichen Sinne, indem er z.B. Daten holt oder Daten manipuliert. Der *Gelegenheitsbenutzer* (*casual end user*), z.B. ein Manager, greift gelegentlich auf die Datenbank zu und benötigt u.U. jedesmal unterschiedliche Informationen. Da er zwar fachlich versiert ist, jedoch über keinerlei Programmierkenntnisse verfügt, ist er angewiesen auf eine hochentwickelte Benutzerschnittstelle. Der *naive Benutzer* (*naive end user*), der „klassische" Beamte, führt dagegen standardisierte Abfrage- und Änderungstransaktionen, die sorgfältig auf den jeweiligen Anwendungsfall abgestimmt sind, regelmässig aus. Der *intellektuelle Benutzer* (*sophisticated end user*), z.B. ein Wissenschaftler oder ein Geschäftsanalytiker, macht sich auch mit komplexen Funktionen und Abfragemechanismen vertraut. Er bevorzugt Situations-abhängig den effektivsten und effizientesten der verfügbaren Schnittstellen-Typen.

Das Gelingen der Kommunikation mit dem Benutzer hängt wesentlich von der Fähigkeit des Systems ab, Benutzer-gerechte Antworten zu generieren. Die Frage, ob eine natürlichsprachliche Schnittstelle dem Benutzer generell natürlichsprachliche Antworten in Satzform präsentieren sollte, kann z.B. getrost mit „nein" beantwortet werden – gerade in folgender Beispiel-Interaktion mit LOQUI [LOQUI 91: 45] scheint die Tabelle das einzig adäquate Formatierungsmittel zu sein:

```
Loqui> who leads projects

project        leader
-------        ------
BIM_Loqui      Jean Louis Binot
MMI2
BIM_Probe      Isabelle de Zegher
HOSCOM
IRHIS
[...]          [...]
```

4 Das Front-End-Paradigma

Das sog. *Front-End-Paradigma* stellt den meistbeachteten und am häufigsten realisierten NLIDB-Ansatz dar. Diese Dominanz widerspiegelt sich auch in der langen Liste kommerziell erhältlicher Front-End-Systeme, die von „älteren" Systemen wie INTELLECT, LOQUI, ENGLISH WIZARD und PARLANCE bis hin zu Systemen der späten 1990er-Jahren, so z.B. EASYASK und ENGLISH QUERY, reicht.
In diesem Kapitel soll das „klassische" Front-End-Paradigma der 1980er-Jahre sowie dessen konkrete Umsetzung beschrieben und diskutiert werden. Dabei stehen Überlegungen zur Portabilität, zur linguistischen Abdeckung und zur Benutzerfreundlichkeit im Zentrum.

4.1 Systemarchitektur und Verarbeitung des Eingabesatzes

Aus einer Prozess-orientierten Perspektive basiert die prototypische Front-End-Systemarchitektur auf drei Hauptbestandteilen, welche für je einen der folgenden drei Verarbeitungsschritte zuständig sind:
- *Linguistisches Front-End*: Analyse der natürlichsprachlichen Eingabe und Übersetzung in eine domänenunabhängige Bedeutungsrepräsentation, formuliert als Ausdruck einer *Bedeutungs-repräsentationssprache* (*meaning representation language*, *MRL*).

- *Datenbank-Back-End*: Übersetzung des MRL-Ausdrucks in einen anwendungsspezifischen Formalismus, d.h. in einen Ausdruck der DML.
- *Antwort-Komponente*, hier vereinfacht als *Ausgabe-Formatierer*-Modul dargestellt: Generierung der Antwort.

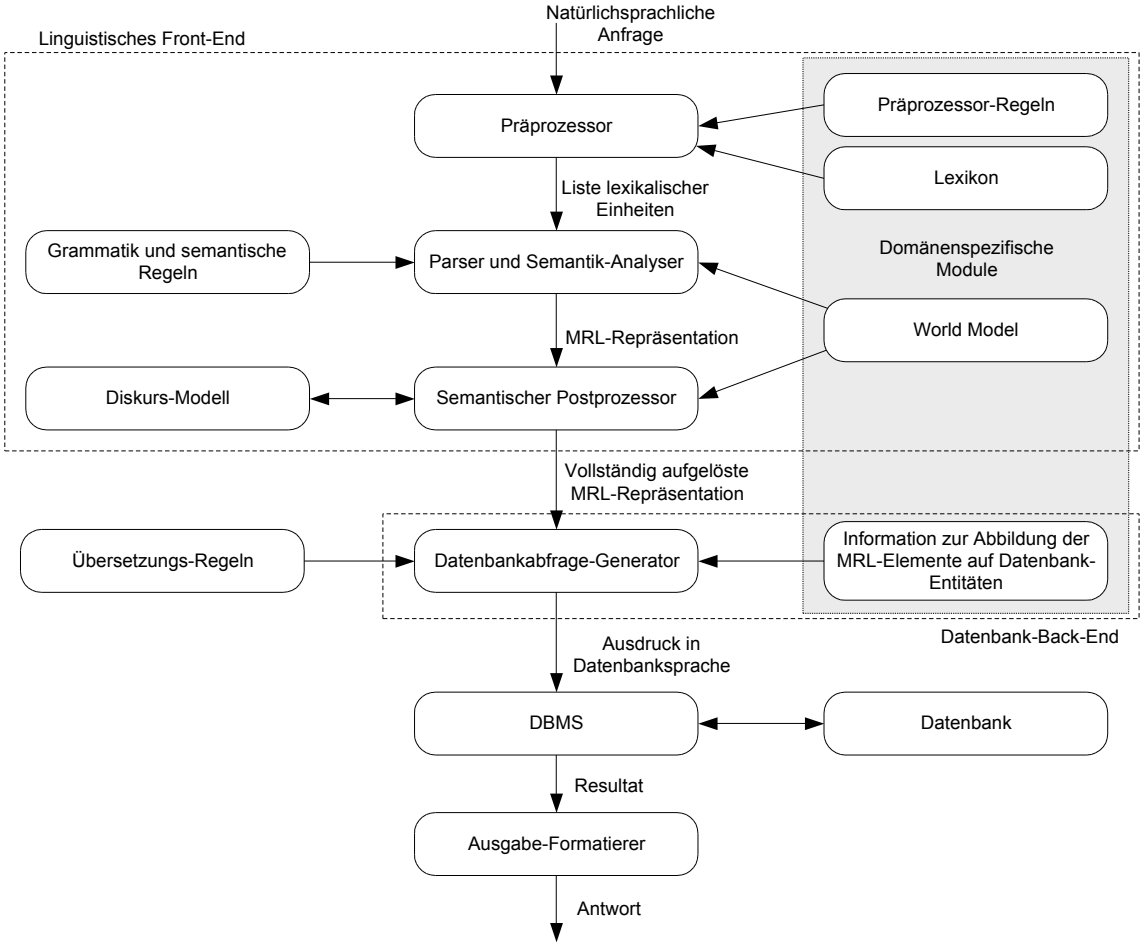


Abbildung 1 - Architektur eines prototypischen Front-End-Systems[36]

Abstrahiert man von den „physischen“ Modulen und Komponenten, so besteht ein Front-End-System im Prinzip aus einer Analyseeinheit, einer Interpretationseinheit sowie einer domänenspezifischen Wissensbasis.

Die Wissensbasis, häufig als *Domänenmodell* bezeichnet, stellt ein explizites Modell der Anwendungsdomäne respektive der Diskursdomäne dar. Einerseits definiert sie die in der Zwischenrepräsentation verwendeten Konzepte (*World Model*) und andererseits stellt sie die für eine Interpretation notwendigen Informationen zur Verfügung (*Information zur Abbildung der MRL-Elemente auf Datenbank-Entitäten*). Die Analyseeinheit, die im Wesentlichen aus *Präprozessor*, *Parser*, *Semantik-Analyser* und *semantischem Postprozessor* besteht, verwendet das Domänenmodell zur Generierung einer domänenunabhängigen, logischen Zwischenrepräsentation, der *MRL-Repräsentation*. Dieser Formalismus wird anschliessend von

[36] Eine sehr ähnliche Architektur besitzt MASQUE/SQL. Vgl. [Androutsopoulos and Ritchie 00].

der Interpretationseinheit, in diesem Fall dem *Datenbankabfrage-Generator*, domänenspezifisch, d.h. anhand des Domänenmodells interpretiert.

4.1.1 Von der Eingabe zur Bedeutungsrepräsentation

Das *Präprozessor-Modul* bereitet die natürlichsprachliche Eingabezeichenkette für die syntaktische und semantische Analyse auf. Dieser Prozess beinhaltet das Tokenizing, die morphologische Analyse sowie die Bestimmung der syntaktischen und semantischen Eigenschaften der Wörter mit Hilfe des Lexikons. Präprozessor-Regeln werden u.a. für die Erkennung von Eigennamen bzw. generell von Datenbankwerten in der Anfrage anhand derer äusseren Form verwendet. Da als Anfragen meistens nur einzelne Phrasen oder Sätze zugelassen sind, fällt die Segmentierung in Sätze und die Analyse der Interpunktion weg.
Die lexikalisch eindeutig bestimmten Wortformen sowie die verschiedenen Interpretationen lexikalisch ambiger Wortformen werden gewöhnlich in Listenform an die syntaktisch-semantische Analyseeinheit weitergereicht.

Das *Lexikon* besteht gerade bei ausgereifteren Systemen meistens aus einem domänenunabhängigen Teil, d.h. aus fest eingebauten Funktionswörtern wie Artikel, Interrogativpronomen, Hilfsverben usw.[37], und aus einem domänenspezifischen Teil, welcher für die jeweilige Anwendungsdomäne konfiguriert werden muss und primär Inhaltswörter enthält. Zwar gilt, dass umso weniger Konfigurationsaufwand anfällt, je höher der Anteil des domänenunabhängigen Lexikons ist, es gilt jedoch ebenso, dass der Aufwand des Erstellens eines universell einsetzbaren „domänenunabhängigen“ Lexikons den Aufwand der Domänenportierung angesichts der typischerweise begrenzten Anwendungsdomänen bei Datenbanken bei weitem übersteigt.

Der *Parser* berechnet aus der Wort-Liste des Präprozessors und einer Menge von Syntaxregeln eine Syntaxstruktur der Eingabedaten. Interessanterweise verwendet gerade das kommerziell erhältliche System LOQUI einen einfachen Top-Down-Parser, der nur korrekte Eingaben akzeptiert und keinerlei partielles Parsing ermöglicht.
Die Syntaxregeln bilden im Idealfall eine domänenunabhängige Grammatik, welche für beliebige Anwendungsdomänen „wiederverwendet“ werden kann und sich dadurch wesentlich von den domänenspezifischen syntaktischen und semantischen Grammatiken[38] einfacherer NLIDB-Systeme, welche die Anfrage direkt in einen anwendungsspezifischen Formalismus übersetzen, abhebt. So verwendet z.B. MASQUE/SQL [Androutsopoulos et al. 93] eine domänenunabhängige Extrapositions-Grammatik, und LOQUIs ebenfalls domänenunabhängiger Grammatik liegt ein GPSG-ähnlicher Formalismus zugrunde.[39]
Der *Semantik-Analyser* generiert aus der syntaktischen Struktur eine formalsprachliche Bedeutungsrepräsentation, welche unter Umständen noch unterspezifiziert ist: So sind z.B. anaphorische Referenzen nicht aufgelöst und Wirkungsbereiche (scopes) logischer Quantoren nicht explizit.
Systeme wie z.B. JANUS oder LOQUI übersetzen die Syntaxstruktur Regel für Regel in eine Bedeutungsrepräsentation, gemäss der Tradition der kompositionalen Montague-Semantik.[40]

Sowohl der Semantik-Analyser als auch der semantische Postprozessor greifen auf ein explizites semantisches Modell der interessierenden Welt, das *World Model*, zu, welches semantische Entitäten und

---

[37] Z.B. Wörter, die auf Aggregatfunktionen, Vergleichsprädikate (z.B. greater_than) oder metalogische Prädikate (z.B. \+: „Non-satisfiability“) abgebildet werden. Vgl. z.B. [Androutsopoulos 92] (MASQUE).
[38] Vgl. [Androutsopoulos et al. 95].
[39] Vgl. [LOQUI 91], [Androutsopoulos et al. 95].
[40] Vgl. [Binot 91], [Androutsopoulos et al. 95].

Beziehungen zwischen semantischen Entitäten spezifiziert. Bei MASQUE/SQL besteht das World Model z.B. aus einer Typenhierarchie und einer Menge von Selektionsrestriktionen:
Die *Typenhierarchie* („is-a hierarchy") bildet die in der Datenbank modellierten Entitäten als Menge hierarchischer Strukturen von Typen, Supertypen und Subtypen ab. Selektionsrestriktionen legen fest, welche Typen von Entitäten die einzelnen Argumente eines MRL-Prädikats enthalten dürfen, wobei Supertypen ihre Subtypen subsumieren. Typenhierarchie und Selektionsrestriktionen werden zur Überprüfung der Konsistenz der Benutzeranfragen, zur syntaktischen Disambiguierung und zur Beschränkung der Menge möglicher Referenzobjekte von anaphorischen Verweisen eingesetzt:
- *Erkennen konzeptueller Fehler*: Z.B. wäre die Anfrage „Does 5 border Italy?" inkonsistent, weil (1) bei dessen logischer Zwischenrepräsentation *borders(5, italy)* das erste Argument vom Typ number_type und nicht vom Typ country_type ist, wie die Selektionsrestriktionen für das Prädikat *borders/2*, ausgedrückt im Prädikat *borders(country_type, country_type)*, vorschreiben, und (2) der Typ number_type kein Subtyp vom Typ country_type ist. [Androutsopoulos et al. 95]
- *Syntaktische Disambiguierung*: Die Anfrage „List all employees in a company with a driving licence" ist insofern syntaktisch ambig, als sich die PP „with a driving licence" entweder auf die NP „all employees" oder auf die NP „a company" beziehen kann.[41] Die letztere Lesart wird kein Resultat bzw. im Idealfall eine Fehlermeldung generieren, da die Datenbank im Zusammenhang mit Firmen keine Informationen zu Führerscheinen beinhaltet – der typische Fall einer konzeptuell fehlerhaften Anfrage. Aufgrund des in der Typenhierarchie und den Selektionsrestriktionen kodierten Welt- bzw. Domänenwissens kann das NLIDB diese Lesart jedoch herausfiltern und die erste Lesart als die intendierte bestimmen.
- *Auswahl möglicher Referenzobjekte*: Eine Selektionsrestriktion könnte deklarieren, dass die Datenbank Lohnangaben nur zu Angestellten, nicht aber zu Kunden macht, was den Suchraum möglicher Bezugsobjekte des Pronomens „his" in „What is his salary?" auf die Menge aller im relevanten Diskurskontext bereits erwähnten (männlichen) Angestellten begrenzt. [Androutsopoulos and Ritchie 00].

Der *semantische Postprozessor* berechnet die vollständig aufgelöste Bedeutungsrepräsentation. Je nach Komplexität des Systems lassen sich dabei verschiedene Phasen unterscheiden: Bestimmung der expliziten logischen Wirkungsbereiche, Auflösung von anaphorischen Referenzen, Ellipsen und deiktischen Ausdrücken anhand des Diskurskontexts und Bestimmung von Sprechakten. [Binot 91]
- Bei der semantischen Analyse werden natürlichsprachliche Artikel gewöhnlich auf logische Quantoren und natürlichsprachliche Konjunktionen wie „und" bzw. „oder" auf logische Konnektoren abgebildet. So enthält das Resultat der syntaktischen und semantischen Analyse, die Bedeutungsrepräsentation in MRL-Form, zwar bereits Quantoren und Konnektoren, diese sind jedoch noch nicht aufgelöst, d.h., deren Skopus ist noch nicht explizit. Ein dediziertes Modul für die eindeutige Skopus-Zuweisung stellt z.B. der „Scoper" von MASQUE/SQL [Androutsopoulos 92] dar.
- Für die Auflösung von anaphorischen Referenzen wird ein einfaches *Diskursmodell* verwendet; gewöhnlich speichert es die erwähnten Entitäten zusammen mit Informationen über deren Eigenschaften sowie deren Auftauchen im Diskurs in einer Liste. [Androutsopoulos and Ritchie 00]

Die vollständig aufgelöste Bedeutungsrepräsentation wird auch als „logische Zwischenrepräsentation" (*intermediate logical form, ILF*) [Binot 91: 63-64] bezeichnet und stellt gewöhnlich einen Ausdruck der Prädikatenlogik erster Stufe oder der Typenlogik dar, erweitert durch spezifische logische Sprachelemente

[41] Oder aber, der Vollständigkeit halber, auf die VP „list <XY>".

höherer Ordnung, wie sie bei der natürlichsprachlichen Datenbankabfrage benötigt werden: z.B. Aggregatfunktionen wie COUNT und AVERAGE oder generalisierte Quantoren[42].

Die MRL von MASQUE/SQL, die sog. LQL („Logical Query Language"), stellt eine Untermenge der Horn-Klausel-Logik dar. Die Anfrage „What is the capital of each country bordering Greece?" würde z.B. in folgende LQL-Repräsentation überführt [Androutsopoulos and Ritchie 00: 6]:

```
answer([Capital, Country]) :-
      is_country(Country),
      borders(Country, greece),
      capital_of(Country, Capital).
```

4.1.2 Von der Bedeutungsrepräsentation zur Datenbankabfrage

Der *Datenbankabfrage-Generator* übersetzt die domänenunabhängige (logische) Bedeutungsrepräsentation in einen anwendungsspezifischen DML-Formalismus, welchen das DBMS „versteht". Er verwendet dabei domänenspezifische Informationen, welche die Basis-Elemente der Bedeutungsrepräsentationssprache (z.B. Konstanten und Prädikatsfunktoren logischer Ausdrücke) auf entsprechende Datenbank-Konstrukte (z.B. Felder und Relationen) abbilden, und domänenunabhängige Übersetzungsregeln, welche die Transformation von MRL-Ausdrücken in ausführbare DML-Formeln spezifizieren.

Der Datenbankabfrage-Generator von MASQUE/SQL z.B. generiert aus der Anfrage in LQL-Form eine passende SQL-Anfrage, die zur Gewinnung der Antwort-Daten aus der relationalen Datenbasis verwendet wird. [Androutsopoulos 92] Dieser Prozess lässt sich in drei Schritte zerlegen, die u.U. wiederholt werden:
(1) Zuerst werden die Konstanten und Prädikatsfunktoren des LQL-Ausdrucks mit Datenbankobjekten assoziiert. In Bezug auf die Klausel „borders(Country, greece)" des obenstehenden LQL-Formalismus (Kapitel **4.1.1**) heisst dies, dass der Prädikatsfunktor „borders" auf eine Relation „BORDERS_INFO" (mit den Attributen „borders1" und „borders2", welche je Länderwerte spezifizieren) und die Konstante „greece" auf einen Datenbankwert „Greece" abgebildet wird.
(2) In einem zweiten Schritt wird unter Anwendung nicht-rekursiver Basis-Regeln jede Klausel der LQL-Repräsentation in einen entsprechenden Ausdruck der DML übersetzt: z.B. „borders(Country, greece)" in

```
SELECT *
FROM   BORDERS_INFO
WHERE  borders2 = ‚Greece'
```

(3) Unter Verwendung rekursiver Übersetzungsregeln und der während (2) produzierten Teilanfragen wird aus der gesamten LQL-Repräsentation die vollständige SQL-Anfrage generiert.

Das DBMS führt die eigentliche Datenbankabfrage aus und reicht deren Resultat an die Antwort-Komponente weiter. Gerade in einem „Real-World"-Szenario sind z.B. Anfrage-Optimierung, Konsistenzsicherung und Vergabe von Zugriffsrechten wichtige Zusatzaufgaben, welche das DBMS am effektivsten und effizientesten erledigt. So generiert z.B. MASQUE/SQL aus jeder Benutzeranfrage höchstens eine SQL-Anfrage, um das DBMS bei deren Ausführung nicht einzuschränken. [Androutsopoulos et al. 93]

[42] Zur Semantik derjenigen natürlichsprachlichen Quantoren, welche nicht auf Quantoren der Prädikatenlogik zurückgeführt werden können, der sog. generalisierten Quantoren, vgl. [Carstensen et al. 01: 268-272].

Ein anderer Retrieval-Ansatz wird bei LOQUI verfolgt: Die proprietären Datenbank-Schnittstellen der Prolog-Umgebung BIM ProLog ermöglichen den Zugriff auf relationale Datenbanken direkt aus dem Prolog-Programm, was die Übersetzung der Zwischenrepräsentation in die anwendungsspezifische DML, in diesem Fall ebenfalls in Prolog, wesentlich erleichtert. [LOQUI 91] Dazu kommt, dass relativ einfach zusätzliche deduktive Module integriert werden können, welche während der Analyse der natürlichsprachlichen Anfrage auf die Datenbank zugreifen oder während der Ausführung der Anfrage z.B. partielle Anfragen evaluieren.

Die *Antwort-Komponente* generiert aus den vom DBMS gelieferten Daten eine passende Antwort, z.B. in Form einer natürlichsprachlichen Phrase, in Tabellenform oder sogar als Datengraphik. Je nach Präsentation der Antwort verwendet die Ausgabe-Komponente die Informationen und Ressourcen des linguistischen Front-Ends zur *Textgenerierung*.
Idealerweise wird beim Fehlschlag der Anfrage eine passende Fehlermeldung generiert, und kooperative Antworten reagieren angemessen auf falsche Benutzerannahmen und indirekte Sprechakte.

4.2 Probleme der Analyse, Interpretation und Generierung

Zahlreiche Probleme bei der Analyse des Eingabesatzes, bei der Interpretation der Bedeutungsrepräsentation sowie bei der Antwortgenerierung sind Ausdruck der Komplexität der Front-End-typischen Aufgabenstellung des *Sprachverstehens* (*Natural Language Understanding*, *NLU*) und bilden eigentliche Prüfsteine für die vermeintlich flexiblen und effizient nutzbaren NLIDB-Systeme.
Zum Beispiel sollte ein NLIDB eigentlich in der Lage sein, syntaktische Ambiguitäten, wie sie bei PP-Anbindungen oder Relativsatzanbindungen vorkommen, zu erkennen und sinnvoll zu behandeln. Aus der Benutzerperspektive ist die Präsentation mehrerer Antworten auf entsprechende alternative Lesarten einer Anfrage schliesslich inakzeptabel.
Eine „pragmatische“ Variante, das Problem anzugehen, besteht darin, den Benutzer in den Disambiguierungsprozess mit einzubeziehen. Aus einer Liste eindeutiger Paraphrasen muss der Benutzer die beabsichtigte Lesart auswählen; die Schnittstelle kann ihn dabei unterstützen, indem sie nur diejenigen Lesarten präsentiert, die mit einer gewissen heuristischen oder statistischen Wahrscheinlichkeit der intendierten Lesart entsprechen. Im Sinne einer „universellen“ Disambiguierungsstrategie können so auch lexikalische und semantische Ambiguitäten sowie Unvollständigkeiten bei anaphorischen Referenzen oder elliptischen Konstruktionen beseitigt werden. Allerdings darf bezweifelt werden, dass diese Strategie zu Effizienz-steigernden „Real-World“-Systemen führt.[43]
Die Abbildung der Konjunktion „und/and“ auf einen logischen Konnektor stellt im Bereich der NLIDB ein weiteres Problem dar. In der Beispielanfrage „List all applicants who live in California and Arizona“ kann anhand der syntaktisch-semantischen Analyse alleine nicht entschieden werden, ob der konjunktiven oder der disjunktiven Lesart von „and“ der Vorzug zu geben ist. Eine einfache Abbildung von natürlichsprachlicher Konjunktion auf logische Konjunktion schlägt in diesem Fall fehl: In einer die reale Welt widerspiegelnden Anwendungsdomäne kann eine Person nicht gleichzeitig in verschiedenen Staaten wohnen. [Androutsopoulos et al. 95] Zur Lösung ähnlicher Fälle bietet es sich an, das erforderliche Weltwissen im World Model zur Verfügung zu stellen. Dieses würde z.B. spezifizieren, dass einer Person nur ein Wohnort zugewiesen werden kann, was die konjunktive Lesart im Beispiel oben ausschliesst.
Allerdings lassen sich längst nicht alle Probleme derart einfach beseitigen – generell gilt, dass der Erfolg des Sprachverstehensprozesses durch die im Zusammenhang mit natürlichsprachlichen „Real-World“-Ausdrücken zu erwartenden Schwierigkeiten gefährdet ist:

[43] Zur Behandlung syntaktischer Ambiguitäten bei NLIDBs vgl. z.B. [Androutsopoulos and Ritchie 00].

- Durch *Inkorrektheiten* wie Rechtschreibefehler, ungrammatikalische Satzkonstruktionen und semantisch „unsinnige“ Sätze. („Show the employees which **earns** more than 50k.“)
- Durch *Unvollständigkeiten*, wie sie bei anaphorischen Verweisen und Ellipsen typisch sind. („Show me the products. – **Their suppliers**?“)
- Durch *Ambiguitäten* lexikalischer (z.B. Homonymie, Homographie), syntaktischer (z.B. Anbindungs-Mehrdeutigkeiten) und semantischer Art (z.B. Skopusambiguitäten).
- Durch *Vagheiten* im Sinne pragmatischer Unbestimmtheit,[44] z.B. bei relativen Ausdrücken ohne Bezugspunkt („Does employee XY earn **a lot**?“).

Was die Antwortgenerierung anbelangt, so erweist sich eine direkte Anzeige der vom DBMS gelieferten Tupel bzw. Tabellen in einigen Fällen als problematisch:
- Die Tupel enthalten u.U. codierte Informationen wie IDs, die in Klartext übersetzt werden sollten.
- Falls keine Tupel angezeigt werden, kann der Benutzer nicht entscheiden, ob der Misserfolg seiner Anfrage durch Fehlschlagen der linguistischen Analyse oder durch Fehlen entsprechender Informationen in der Datenbank verursacht worden ist.
- Falls keine Tupel angezeigt werden und die Anfrage falsche (implizite) Benutzerannahmen enthält, sollte der Benutzer über letztere aufgeklärt werden.
- Generell sollten die (impliziten) Benutzerabsichten berücksichtigt werden: U.U. liegt ein indirekter Sprechakt vor, dessen wortwörtliche Analyse und Interpretation nicht zum gewünschten Resultat führt.

In den folgenden Unterkapiteln sollen einige der im Zusammenhang mit NLIDB-Systemen besonders relevanten und/oder problematischen Aufgabenstellungen aus den Bereichen Analyse, Interpretation und Generierung zur Sprache kommen: die Verarbeitung von Datenbankwerten in der Anfrage, die Verarbeitung von Diskursphänomenen, die Abbildung der Bedeutungsrepräsentation auf die DML und die Generierung kooperativer Antworten.

4.2.1 Verarbeitung von Datenbankwerten in der Anfrage

Bei der Verarbeitung des Eingabesatzes stellt sich das Problem der Erkennung und der lexikalisch-semantischen Bestimmung von Datenbankwerten, z.B. von Eigennamen. Diese stellen im Hinblick auf die finale Datenbank-Abfrage wichtige Elemente der semantischen Bedeutungsrepräsentation dar, spezifizieren sie doch diejenigen Eigenschaften, die ein Tupel aufweisen muss, um in die Ergebnisrelation aufgenommen zu werden.
Das Problem der Datenbankwerte in der Anfrage besteht aus drei Teilproblemen:
(1) Wie erkennt das System, dass ein natürlichsprachlicher Ausdruck auf einen Datenbankwert verweist?
(2) Welche semantische Repräsentation weist das System dem (erkannten) Datenbankwert zu?
(3) Wie unterscheidet das System zwischen neuen bzw. nicht in der Datenbank enthaltenen Werten und falsch geschriebenen aktuellen Datenbankwerten?

Aufgrund der hohen Datenaufkommen bei „Real-World“-Datenbanken ist die Aufnahme aller Datenbankwerte (wie z.B. Eigennamen) ins (domänenspezifische) Lexikon bzw. ins Domänenmodell nicht empfehlenswert. Einerseits leidet die Effizienz bei der Ausführung der Anwendung unter der (potentiellen) Grösse des Lexikons, und andererseits besteht zwischen den Informationen in der Datenbank und denjenigen im Lexikon eine hohe Redundanz, was wiederum erfordert, dass das Lexikon bei jeder

[44] Vgl. [Bussmann 90].

Änderung der Datenbasis angepasst wird. Die Konsequenz ist, dass Wörter, welche Datenbankwerte repräsentieren, vom NLIDB beim Nachschlagen im Lexikon (lexical lookup) nicht erkannt werden.
Damit der Analyseprozess nicht gefährdet ist, muss das System folglich über einen alternativen Mechanismus zur Erkennung von Datenbankwerten bzw. über einen Mechanismus zur Bestimmung unbekannter Wörter verfügen. Ein verbreiteter Lösungsansatz basiert auf Pattern-Matching-Regeln und typographischen Konventionen. So könnte z.B. eine Regel englische Wörter mit grossem Anfangsbuchstaben auf einen Basis-Lexikoneintrag vom Typ Eigennamen abbilden. Ein Nachteil dieses Ansatzes besteht darin, dass der Basis-Lexikoneintrag semantisch unterspezifiziert ist und sich insofern nicht für die Auflösung anaphorischer Referenzen eignet. Zudem ist der Ansatz bei Anwendungsdomänen mit typographisch heterogener oder inkonsistenter Namensgebung praktisch nicht realisierbar.
Der „gewöhnliche" Ansatz zur Bestimmung unbekannter Wörter besteht darin, diese prinzipiell und ohne weitere Analyseschritte als Datenbankwerte zu behandeln. [Winiwarter 94] Allerdings besteht so keine Möglichkeit, zwischen neuen, (noch) nicht in der Datenbank enthaltenen Werten (z.B. bei Hinzufüge- oder Aktualisierungsoperationen) einerseits und falsch geschriebenen aktuellen Datenbankwerten andererseits zu unterscheiden, was eine Rechtschreibkorrektur verunmöglicht.
Die z.B. bei INTELLECT realisierte alternative Methode sieht einen zusätzlichen Suchvorgang in der Datenbank während der natürlichsprachlichen Analyse vor. Allerdings scheinen die (potentiell) immensen Suchräume und die Unzulänglichkeiten der relationalen Datenbanktechnologie massive Effizienzprobleme zu verursachen. [Winiwarter 94]

4.2.2 Verarbeitung von Diskursphänomenen

Die natürliche Sprache stellt Mechanismen zur Verfügung, die eine sehr effiziente Dialogführung ermöglichen. Die maschinelle Verarbeitung von satzübergreifenden Phänomenen wie anaphorischen Verweisen oder elliptischen Konstruktionen erfordert allerdings Methoden der *Diskursanalyse* und der *Diskursrepräsentation*,[45] welche, im Gegensatz zu den Satz-orientierten Methoden der Semantikanalyse, den sprachlichen Diskurskontext mit berücksichtigen.

Kohäsionsmittel wie anaphorische Verweise oder elliptische Satzkonstruktionen generieren einen Zusammenhang zwischen mehreren Sätzen, indem sie bei wiederholt auftauchenden Elementen Substitutionen vornehmen. So wird z.B. der Name eines wiederholt referenzierten Objektes durch ein Pronomen oder, im Fall einer Satzellipse, eine rekurrente Phrase durch eine Leerstelle, durch „nichts" ersetzt. Bei einer Interaktion mit natürlichsprachlichen Schnittstellen stellen solche Mechanismen einen wesentlichen Effizienzfaktor dar, erlauben sie doch in Analogie zur Mensch-Mensch-Kommunikation die Verwendung von Rückbezügen und kurzen *Folgefragen* (*follow-up questions*).
Anaphorische Verweise und Ellipsen verursachen bei der Analyse der natürlichsprachlichen Anfrage insofern Probleme, als ihnen mangels expliziter Referenz beim Satz-orientierten, syntaktisch-semantischen Parsing keine semantische Repräsentation zugewiesen werden kann. Hier kommen die bereits erwähnten Postprozessor-Module (Kapitel **4.1.1**) zum Einsatz, welche neben expliziten Skopuszuweisungen auch Referenzobjekte bestimmen bzw. Substitutionen rückgängig machen. Potentiell einfachere (für anaphorische Referenzen) oder potentiell komplexere *Diskursmodelle* (für elliptische Konstruktionen) stellen das benötigte Diskurswissen bzw. das Wissen über den Diskurskontext zur Verfügung.[46]

[45] Vgl. z.B. die Besprechung der Diskursrepräsentationstheorie in [Carstensen et al. 01: 272-284].
[46] Vgl. [Androutsopoulos and Ritchie 00].

Während die Laborsysteme MASQUE/SQL und SYSTEM X (Kapitel **4.4**) Diskursanalyse als wünschenswerte und mit geringem Aufwand integrierbare Option behandeln, verarbeitet das kommerziell erhältliche LOQUI immerhin anaphorische Verweise, die sich auf Nominalphrasen in einem vorangehenden Satz beziehen. Unterstützt wird die Verwendung von Personal- und Demonstrativpronomen, die Verwendung von anaphorischen (definiten) Nominalphrasen und die Verwendung von Possessivpronomen, wobei sowohl Numerus als auch Genus von Verweis und Referenzobjekt übereinstimmen müssen. [LOQUI 91]
Auf Wunsch zeigt LOQUI zusätzlich zum Anfrage-Resultat das zu einem anaphorischen Ausdruck ermittelte Referenzobjekt an, was eine Überprüfung des semantischen Interpretationsprozesses ermöglicht [LOQUI 91: 11]:

```
Loqui> jlb works on mmi2

Yes.

Loqui> does he lead it

person = [JLB]
project = [MMI2]
Yes.
```

Hinsichtlich der Effizienz der Anfrageformulierung besonders interessant sind sog. *partitive Konstruktionen*, d.h. die Möglichkeit, mittels Nominalphrase eine grobe Resultatmenge vorzuselektieren und anschliessend in der eigentlichen Anfrage auf diese Selektion zu verweisen [LOQUI 91: 51]:

```
Loqui> the people who work on MMI2

OK.

Loqui> Which of them lead a project?

David Sedlock, Fabienne Balfroid, Jean Louis Binot
project        leader
-------        ------
BIM_Loqui      Jean Louis Binot
MMI2
MAE_IEXP       Fabienne Balfroid
NLPAD          David Sedlock

Loqui> One of them reports to MM.

Yes.
Jean Louis Binot

Loqui> Which projects does JLB lead?

BIM_Loqui, MMI2

Loqui> FB works on how many of them?

1
MMI2
```

(Man beachte in obenstehendem Beispiel auch die Tabellenformatierung sowie die Überbeantwortung.)

4.2.3 Pragmatikanalyse und kooperative Antworten

Die *Pragmatik* betrachtet sprachliche Äusserungen nicht isoliert, sondern eingebettet in einen situativen, aussersprachlichen Kontext. Sie verwendet Begriffe u.a. der *Sprechakttheorie*, um Sprache unter einem handlungsbezogenen Gesichtspunkt zu beschreiben, und untersucht Phänomene wie Deixis,

Präsuppositionen, Implikatur, Sprechakte und Konversationsstruktur. [Linke et al. 94] Das Ziel der pragmatischen Analyse einer natürlichsprachlichen Anfrage besteht entsprechend darin, die Bedeutungsrepräsentation mit zusätzlichen, (im linguistischen Sinne) pragmatischen Angaben etwa zum situativen *Diskurskontext* und zu den *Benutzerabsichten* anzureichern. Diese erweiterte Bedeutungsrepräsentation bildet die Voraussetzung für eine der Benutzeranfrage angemessene Systemreaktion und insofern die Ausgangslage für die Generierung *kooperativer Antworten*.
Ein NLIDB wird dann als *kooperativ antwortendes System* bezeichnet, wenn es Antworten generiert, welche neben den direkten Resultaten des Daten-Retrievals Situations-gebundene, Benutzer-gerechte Informationen enthalten [Wu et al. 96], d.h.
- Informationen, welche im Falle eines Fehlschlags der Anfrage für das Verständnis der Ursache nützlich sind,
- Informationen, welche Missverständnisse bei der Auslegung der Anfrage-Resultate durch den Benutzer verhindern sollen,
- Informationen, welche falsche Benutzerannahmen bezüglich des Datenbank-Modells korrigieren sollen.

4.2.3.1 Präsuppositionen

Anhand des folgenden Beispiels einer Interaktion mit LOQUI [LOQUI 91: 49] soll der Zweck kooperativer Antworten, d.h. der durch die zusätzlichen Informationen entstehende Nutzen erläutert werden:

```
Loqui> does Ronald Reagan lead a project

There is no such thing as Ronald Reagan
```

Die obenstehende Anfrage kann als Ausdruck einer *falschen Benutzerannahme* (*user's misconception*) bezüglich der Anwendungsdomäne, dass es darin (überhaupt) eine Person (oder Sache) namens „Ronald Reagan" gibt, verstanden werden. Man spricht in diesem Fall von einer *falschen (Existenz-)Präsupposition* in der Anfrage. Ein schlichtes „No" als Antwort, abgeleitet vom NULL-Wert, den das DBMS zurückliefert, würde den Benutzer darüber im Ungewissen lassen, ob (a) „Ronald Reagan" kein Projekt leitet oder ob (b) gar kein „Ronald Reagan" existiert und deshalb keine Informationen zur dessen Funktion vorhanden sind. Da der Eigenname in diesem Fall gar kein Denotat besitzt und insofern eine konzeptuelle Fehlannahme des Benutzers (Variante (b)) vorliegt, reagiert das System mit einer angemessenen Fehlermeldung, welche die falsche Existenzpräsupposition aufdeckt bzw. die Benutzerannahme korrigiert.
Gemäss [LOQUI 91: 49] erkennt LOQUI neben fehlenden Eigennamen-Denotaten auch leere Allquantifikationen wie im folgenden Beispiel:

```
Loqui> does every person that works on 20 projects report to jlb

There is no such person
```

Eine relativ einfache Methode zur Erkennung falscher Präsuppositionen besteht darin, existenzquantifizierte Bestandteile der Bedeutungsrepräsentation separat in Teilanfragen zu übersetzen und diese vor dem Ausführen der vollständigen Anfrage zu evaluieren. Sobald eine falsche Existenzpräsupposition erkannt wird, bricht das NLIDB die Evaluierung ab und generiert eine entsprechende Fehlermeldung. [Binot 91]

4.2.3.2 Benutzerabsichten

Neben der Behandlung von Präsuppositionen stellt auch die Bestimmung der *Benutzerabsichten* eine Teilaufgabe der pragmatischen Analyse dar. Konzeptuell der Sprechakttheorie verpflichtet, versuchen

entsprechende Ansätze, einerseits zu ermitteln, was der Benutzer durch seine sprachliche „Handlung“ erreichen möchte, und andererseits dem Benutzer mittels kooperativer Antworten zu helfen, sein Ziel zu erreichen.
[LOQUI 91: 48-49] führt unter dem Stichwort „Elaborating“ vor, wie LOQUI unter „bestimmten“, nicht näher spezifizierten Umständen das in der Mensch-Mensch-Kommunikation verbreitete *Überbeantworten* (*over-answering*) nachahmt:

```
Loqui> does JLB lead more than 1 project

Yes.
BIM_Loqui, MMI2
```

LOQUI „vermutet“, dass der Benutzer neben dem wortwörtlich erfragten Sachverhalt wissen möchte, um welche Projekte es sich denn handelt. Insofern „erahnt“ das System den nächsten Schritt des Benutzer und nimmt dessen Beantwortung vorweg. Entsprechende kooperative Antworten können unter Anwendung relativ einfacher, domänenunabhängiger Strategien generiert werden: Im Falle einer affirmativen Antwort auf eine Ja/Nein-Frage ergänzt das NLIDB das Ergebnis „Yes“ mit der Antwort auf die intern automatisch generierte, korrespondierende WH-Frage, in diesem Beispiel etwa „which project(s) does JLB lead?“. [Androutsopoulos et al. 95]

Um eine weitere Form des kooperativen Antwortens handelt es sich bei der Interpretation einer sog. *skalaren Implikatur*[47], einer Art von Umdeutungsverfahren, die auf einer bestimmten Werte-Skala basiert und danach trachtet, die Qualität der Antwort zu erhöhen, indem semantische Entitäten der Bedeutungsrepräsentation durch geeignete höherwertige substituiert werden. So könnte semantisches Wissen, wie es z.B. in der Typenhierarchie repräsentiert ist, dazu verwendet werden, um semantisch möglichst präzise Antworten auf tendenziell vage Fragen zu geben:

```
> Is Smedley a student?

Yes. A graduate student.
```

Auch dies ist eine Form von Überbeantworten: Da der semantische Typ „graduate student“ spezifischer als dessen Super-Typ „student“ und deshalb höher in der Referenz-Skala eingestuft ist, ersetzt das System den in der Anfrage enthaltenen Typ durch dessen Sub-Typ.

Komplexere, mit domänenspezifischen Regeln operierende Mechanismen sollen z.B. unsinnige „kooperative“ Antworten wie im folgenden Beispiel der Befragung eines Flugreservationssystems verhindern [Androutsopoulos et al. 95: 35]:

```
> Do American Airlines have a night flight to Dallas?

No, but they have one to Miami.
```

In diesem Fall sollte das System erkennen können, dass der Benutzer primär beabsichtigt, nach Dallas zu fliegen, und dass die Wahl der Fluggesellschaft nur von untergeordneter Priorität ist. Basierend auf kontextabhängigem Regelwissen, das Beziehungen zwischen den (möglichen) Äusserungen eines Benutzers und den damit verbundenen Zielen deklariert, kann ein Expertensystem-ähnliches Inferenzmodul

[47] Begriff und Beispiel stammt aus [Wu et al. 96: 10]. „Implikaturen“ sind, vereinfacht ausgedrückt, Resultate von (nicht-logischen) Schlussverfahren zur Deutung nicht explizit ausgedrückter, mitgemeinter Inhalte. So werden z.B. die meisten Präsuppositionen zu den „konventionellen Implikaturen“ gezählt. Vgl. [Linke et al. 94: 195-198].

von der Bedeutungsrepräsentation auf die mit der Anfrage verbundenen Benutzerabsichten schliessen und die gewünschten Informationen aus der Datenbank liefern. [Androutsopoulos et al. 95]
Diese Regeln sind Teil der expliziten Vorstellung, die das System vom Benutzer hat, und konstituieren ein Modell des Benutzers, das sog. *Benutzermodell*. Der Begriff der *Benutzermodellierung* fasst all diejenigen Methoden zusammen, welche „interaktive Software-Systeme in die Lage versetzen, ihr Verhalten an ihren jeweiligen Benutzer anzupassen mit Hilfe der Erstellung und Ausnutzung eines Benutzermodells, das die Eigenschaften des Benutzers beinhaltet“ [Carstensen et al. 01: 316]. Benutzermodelle irgendwelcher Art kommen folglich immer dann zum Einsatz, wenn es darum geht, Inhalte und Interaktionsstil auf den Benutzer abzustimmen.

4.2.3.3 Front-End-Systeme versus Dialogsysteme

Vergleicht man die Analyse-Fähigkeiten von MASQUE/SQL mit denjenigen von LOQUI (oder auch MASQUE/PRO), so sind gerade im Bereich der Pragmatikanalyse die Vorteile von LOQUI augenfällig. Dies hängt mit den spezifischen Eigenheiten von dessen Systemarchitektur zusammen: Während das „klassische“ Modell des Front-End-Systems zu einer relationalen Datenbank nach der Generierung der SQL-Form die Kontrolle über die Anfrage verliert und vom DBMS typischerweise nur das wenig aussagekräftige, direkte Anfrage-Resultat in Form einer relationalen Tabelle erhält, hat LOQUI bzw. dessen Prolog-Interpreter aufgrund der integrierten Prolog-RDB-Schnittstelle jederzeit die Kontrolle über das Beweisverfahren und kann so mit bedeutend weniger Aufwand auch komplexere Fehlerbehandlungsmechanismen realisieren. Ein Front-End-System wie MASQUE/SQL müsste, um etwaige Fehler lokalisieren und interpretieren zu können, in der Lage sein, bei einer leeren Resultatmenge Teile der Anfrage separat zu übersetzen und auszuführen, bis die Fehlerursache gefunden ist. (Gemäss [Androutsopoulos et al. 93] generiert MASQUE/SQL im Gegensatz zum (älteren) MASQUE/PRO keinerlei kooperative Antworten.)
Allerdings beschränkt sich selbst bei LOQUI die „Pragmatikanalyse“ auf die Verarbeitung einer eng begrenzten Auswahl sprachlicher Phänomene und, im Gegensatz zu eigentlichen Dialogsystemen, auf die Verarbeitung einzelner isolierter Diskurseinheiten.
Damit ein NLIDB auch in der Lage ist, den Benutzer auf *inkonsistente Annahmen* (*inconsistent beliefs*) in einer Serie von Anfragen hinzuweisen, benötigt es ausgereiftere Methoden der Diskursmodellierung als die bei typischen Front-End-Systemen üblichen: So ist z.B. eine komplexe *Diskurshistorie*, d.h. ein kontinuierlich angepasstes und erweitertes semantisches Modell des laufenden Diskurses, unabdingbar.
Inkonsistente Benutzerannahmen liegen z.B. folgender Sequenz von Anfragen an eine akademische Datenbank zugrunde (und werden in diesem Fall vom System KDA erkannt) [Wu et al. 96: 59-60]:

```
Q1> Who specializes in ‚database’ in the department of computer
science?
R1> (person-name) W.Huang   U.Li   [...]

Q2> Show me their title.
R2> (   person-name  title of professional post)
        W.Huang      assoc-prof
        U.Li         assis-prof
        [...]        [...]


Q3> How many projects are supported by the department of computer
science?
R3> (CS-89441, CS-89461, [...], CS-89491)

Q4> Among the faculty members listed in R1, who joined CS-89491?
R4> Null. This query is inconsistent with the query Q1.
```

```
In Q1, you specified that ‚faculty members specialize in ‚database.'
In Q4, you specified that ‚faculty members joined ‚CS-89491'.
These specifications are inconsistent, because:
- the research area of ‚CS-89491' is ‚VLSI design'
- the specialty of a faculty member must be identical with the
research area of the project that the faculty member joined.
```

Aufgrund der Inkonsistenz der Benutzerannahmen, welche in *Q1* und *Q4* zum Ausdruck kommen, wird als Resultat auf Anfrage *Q4* der NULL-Wert zurückgegeben. Der Benutzer wird durch die kooperative Antwort über die Ursache für den Fehlschlag der Anfrage, d.h. über die Widersprüchlichkeit der aus seinen Anfragen ableitbaren Präsuppositionen aufgeklärt.

4.2.4 Generierung des anwendungsspezifischen Formalismus

Idealerweise verbirgt ein NLIDB-System die Komplexität der DBMS-spezifischen, formalen Abfragesprache vor dem Benutzer, ohne dabei die Funktionalität im Bereich der Datenwiedergewinnung und Datenmanipulation einzuschränken. Abgesehen von der adäquaten Übersetzung der natürlichsprachlichen Eingabe in eine Zwischenrepräsentation ist demnach eine möglichst exakte und konsistente Abbildung der Zwischenrepräsentation auf den anwendungsspezifischen Formalismus unabdingbar.

Gerade der Front-End-typische Ansatz der Trennung des linguistischen Front-Ends vom Datenbanksystem erweist sich in dieser Hinsicht als problematisch. Das Front-End-Paradigma beruht unter anderem auf der Annahme, dass jedem Konstrukt der Bedeutungsrepräsentationssprache ein Datenbank-Konstrukt, z.B. eine Relation, zugewiesen werden kann. Dabei spielt es im Prinzip keine Rolle, ob das Datenbank-Konstrukt in der gewünschten Form direkt in der Datenbank gespeichert ist oder aus der Datenbank berechnet werden muss, indem z.B. Beziehungen zwischen Relationen ausgewertet bzw. join-Operationen angewandt werden.[48] Dass diese Annahme nicht in jedem Fall zutrifft, illustriert das folgende Beispiel, welches dem sog. *„Doctor on board"-Problem* den Namen gegeben hat. [Androutsopoulos and Ritchie 00]

ships_info		
ship	**crew**	**doctor**
Vincent	420	true
Sparrow	14	false
…	…	…

Zwischen der (linguistisch motivierten) Bedeutungsrepräsentation der Anfrage „Is there a doctor on (board of) the Vincent?", wie sie etwa MASQUE/SQL generieren würde, (1) und einer Datenbank, welche nur die eine Relation *ships_info* enthält (siehe oben), besteht eine konzeptuelle Diskrepanz.

```
(1)     answer([]):-
              is_doctor(D),
              on_board(D, vincent).
```

[48] Häufig wird eine in der natürlichsprachlichen Anfrage ausgedrückte Beziehung zwischen Objekten der Welt in der relationalen Datenbank nicht durch eine Basis-Relation, sondern nur durch eine „virtuelle", d.h. aus den Basis-Relationen ableitbare Relation repräsentiert. „Ableitbar" heisst, dass man die implizite Relation bilden kann, indem man eine Sequenz von join-Operationen über zwei oder mehr Basis-Relationen ausführt. (Siehe Kapitel **2.3.3**) Das NLIDB sollte in der Lage sein, solchen natürlichsprachlichen Ausdrücken die korrespondierenden Join-Pfade zuzuweisen – dies erfordert allerdings, dass die Join-Pfade für jeden zu erwartenden Ausdruck entweder manuell spezifiziert (z.B. bei TEAM und TQA) oder (halb-)automatisch aus der Datenbankstruktur generiert werden. [Wu et al. 96] behandelt das Thema im Zusammenhang mit SYSTEM X relativ ausführlich.

Deren Ursache liegt in den inkonsistenten semantischen Modellen: Weder der Prädikatsfunktor *is_doctor* noch der Prädikatsfunktor *on_board* kann auf korrespondierende Datenbankobjekte abgebildet werden, da weder eine *doctor*-Relation noch eine Basisrelation, welche die Beziehung „to be on board" zwischen den Datenbankobjekten *doctor* und *ship* modelliert, existiert. (Letztere kann nicht einmal (als virtuelle Relation) berechnet werden, da die Datenbankobjekte *doctor* und *ship* eben nicht existieren.)
Weil die lingustische Analyse des Eingabesatzes ausschliesslich auf dem semantischen Modell des linguistischen Front-Ends, dem World Model, basiert, müssen für die anschliessende Generierung des anwendungsspezifischen Formalismus u.U. konzeptuelle Diskrepanzen zwischen der MRL-Anfrage und dem semantischen Modell der Datenbank überwunden werden. In diesem Fall besteht die Lösung des Problems darin, den problematischen MRL-Ausdruck in einen äquivalenten MRL-Ausdruck umzuwandeln, der die ursprüngliche Diskrepanz überbrückt. Die folgende Bedeutungsrepräsentation (2) enthält den Prädikatsfunktor *doctor_on_board*, der auf eine Restriktion der Relation *ships_info* ohne die *crew*-Kolonne abgebildet werden kann.

```
(2)     answer([]):-
              doctor_on_board(D, vincent).
```

Allerdings kann das Front-End nicht direkt eine linguistisch nicht motivierte Bedeutungsrepräsentation wie (2) generieren. Dazu wird eine zusätzliche Wissensbasis in Form von expliziten, domänenspezifischen Regeln benötigt, welche eine Assoziation zwischen dem World Model des NLIDB und dem (im Fall einer SQL-Datenbank impliziten) semantischen Modell der Datenbank herstellt. In der Illustration zur Front-End-Architektur (Kapitel **4.1**) ist diese Wissensbasis als *Informationen zur Abbildung der MRL-Elemente auf Datenbank-Entitäten* im Datenbank-Back-End integriert und steht dem Datenbankabfrage-Generator zusätzlich zu den generellen Übersetzungsregeln zur Verfügung.

4.3 Vorteile der Front-End-Architektur

Die Front-End-Architektur besitzt gegenüber dem Ansatz der direkten Übersetzung vor allem zwei prinzipielle Vorteile:
(1) *Modularität*: Da das NLIDB-System in mehrere Module mit definierten Schnittstellen unterteilt ist, lassen sich Module bei einer Portierung relativ einfach austauschen und zusätzliche Module relativ bequem einfügen. Denkbar sind z.B. deduktive Module, die aus bestehenden Daten neue Fakten berechnen und so dem Datenbanksystem Funktionen eines typischen Expertensystems „beibringen".
(2) Portabilität:
- Die Verwendung einer domänenunabhängigen Zwischenrepräsentation erleichtert die Portierung der linguistischen Analyseeinheit in eine neue DBMS-Umgebung bzw. generell in eine neue Anwendungsumgebung, da im Idealfall nur das anwendungsspezifische Übersetzungsmodul (Datenbank-Back-End) angepasst werden muss. So lässt sich das System im Prinzip als linguistisches Front-End zu verschiedenen Programmen (Expertensystem, Betriebssystem, usw.) einsetzen.
- Die Trennung von domänenspezifischem Wissen (Domänenmodell!) und domänenunabhängigem Wissen (z.B. Syntaxregeln) erleichtert die Konfigurierung für eine neue Anwendungsdomäne, z.B. beim Wechsel der Datenbank, da nur die domänenspezifischen Module angepasst werden müssen. Beim prototypischen Front-End-System (Kapitel **4.1**) sind dies die *Präprozessor-Regeln*, das *Lexikon*, das *World Model* sowie die *Information zur Abbildung der MRL-Elemente auf Datenbank-Entitäten*, wobei sich der Aufwand in Bezug auf die Lexikon-Konfiguration bei der Unterscheidung zwischen einem domänenunabhängigen und einem domänenspezifischen Wörterbuch weiter reduziert.

4.4 System X: Ein Front-End-System

Bei SYSTEM X[49] handelt es sich um ein an der Simon Fraser University entwickeltes Front-End-Laborsystem der mittleren bis späten 1980er-Jahren, das u.a. von den Erfahrungen mit TQA profitiert. Zu Beginn der 1990er-Jahre wird SYSTEM X abgelöst von SYSTEMX, dessen Fokus auf der „Real-World"-Tauglichkeit, d.h. auf dem Einsatz als „industrial application" liegt.[50]

Das hier besprochene „ältere" SYSTEM X unterscheidet sich insofern vom prototypischen Front-End-System (Kapitel **4.1**), als es für die Analyse der natürlichsprachlichen Anfrage nicht ein World Model mit vordefinierten Wissensstrukturen, sondern direkt ein semantisches Modell der Datenbankstruktur verwendet. Die prinzipiellen Vorteile bestehen darin, dass keine zusätzliche Abbildung der Wissensstrukturen auf die Datenbankstruktur notwendig ist, dass damit das Problem inkonsistenter semantischer Modelle entschärft wird und dass der Prozess der linguistischen Analyse und anwendungsspezifischen Interpretation „zielgerichteter", d.h. sich an der Zielrepräsentation orientierend, und damit effizienter verläuft.
SYSTEM X betreibt weder Diskurs- noch Pragmatikanalyse und kann aufgrund der limitierten Grammatik keine natürlichsprachlichen Konjunktionen verarbeiten.

4.4.1 Analyse und Interpretation

Die beiden Subsysteme *TEMPLATE* und *MORPHOS* dienen der Reduzierung des Lexikonumfangs. TEMPLATE erkennt Wörter in der Anfrage, welche Datenbankwerte repräsentieren, z.B. Eigennamen, codierte Namen und Zahlen, aufgrund ihrer typographischen Form und weist ihnen unabhängig vom syntaktischen Lexikon lexikalische Eigenschaften zu.
MORPHOS, ein regelbasiertes Morphologie-Analyse-Werkzeug, lemmatisiert die Wortformen des Eingabesatzes. Falls kein Lexikon vorhanden ist oder das Lemma nicht im syntaktischen Lexikon gefunden werden kann, generiert MORPHOS einen neuen Lexikoneintrag – entweder automatisch oder, bei ambigen Flexionssuffixen, gemäss der Benutzerantwort auf die Rückfrage des Systems. Entsprechend wird das syntaktische Lexikon (bzw. dessen domänenspezifischer Teil) während des Gebrauchs ständig erweitert.

Beim Parsing kommt eine Top-down-breadth-first-Strategie ohne Backtracking zum Einsatz. Dank eines Syntaxstruktur-Zwischenspeichers muss jede Konstituente nur einmal syntaktisch analysiert werden. Das Resultat des Parsing-Prozesses, die Menge der syntaktischen Lesarten des Eingabesatzes, kann vom Semantik-Interpreter zur syntaktischen Disambiguierung auf falsche Lesarten untersucht werden.

[49] Informationen zu SYSTEM X stammen, falls nichts anderes angegeben ist, aus [Wu et al. 96].
[50] Zu SYSTEMX „ohne Leerschlag" vgl. [Cercone et al. 93].

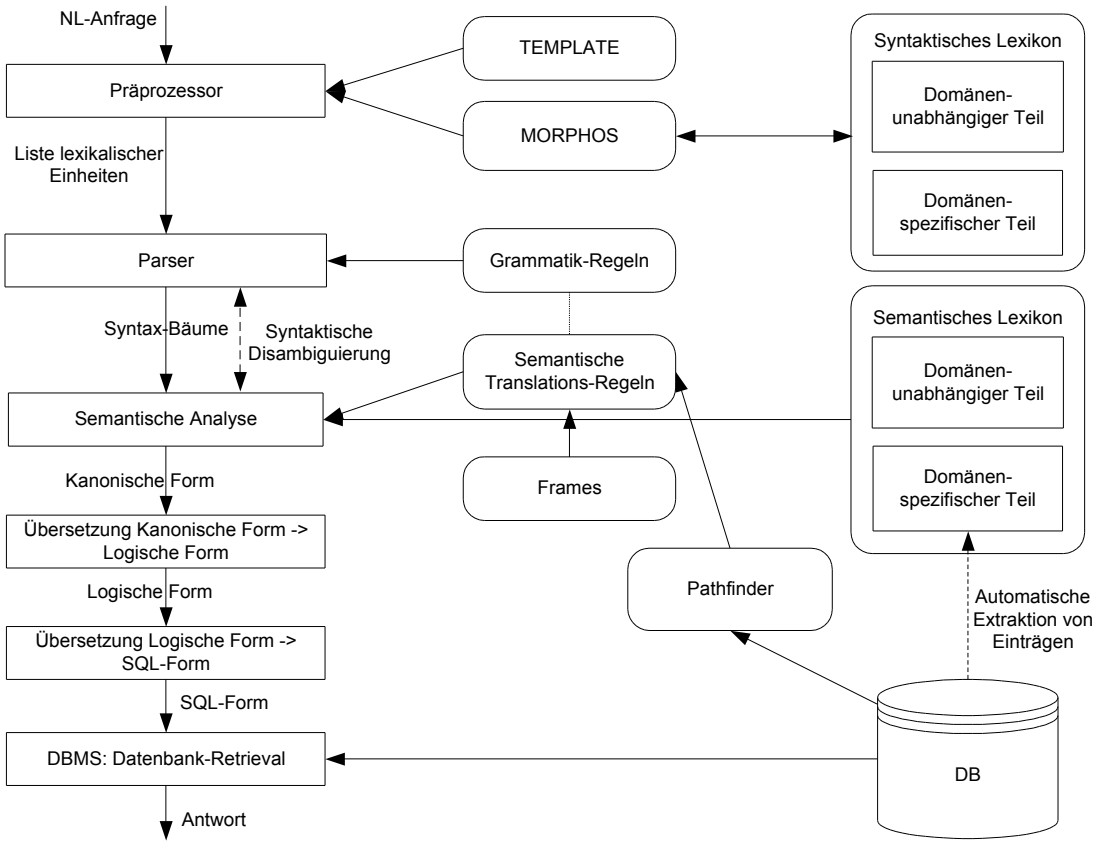


Abbildung 2 - Architektur von System X[51]

Die Aufgabe der semantischen Analyse besteht darin, aus einem Syntax-Baum die korrespondierende *kanonische Form*, die sog. *Canonical Query Representation* zu generieren. Diese entspricht im Prinzip einer vollständig aufgelösten MRL-Repräsentation (Kapitel **4.1**), die allerdings bereits interpretiert ist, d.h. bereits die assoziierten Datenbankobjekte, deren Beziehungen und darauf angewandte Prädikate (z.B. „greaterthan“) und Aggregatfunktionen spezifiziert.

Da die Datenbankstruktur die Basis des semantischen Modells bildet, ist die semantische Repräsentation, d.h. die „Bedeutung“ eines Ausdrucks, dessen Funktion in der Datenbank:

- Nomen und Adjektive werden gewöhnlich mit Datenbankwerten, die durch entsprechende Attribute definiert werden, assoziiert. Die semantische Repräsentation des Nomens „course“ in der NP „math course“ ist z.B. (in einer universitären Datenbank) ein Datenbankwert, der durch den Namen des Attributs „course_name“ spezifiziert ist, und diejenige des Adjektivs „math“ ein Datenbankwert, der durch den Namen entweder des Attributs „major“ (Hauptfach) oder des Attributs „department“ (universitärer Fachbereich) spezifiziert ist.
- Verben werden, falls möglich, mit Relationen, welche Beziehungen zwischen Datenbankobjekten ausdrücken, assoziiert. So ist z.B. die semantische Repräsentation eines Verbs „sich einschreiben / to enroll“ eine Relation namens „ENROLL“, welche eine Beziehung zwischen den Datenbankobjekten „STUDENT“ und „CLASS“, ihrerseits Relationen, spezifiziert.

[51] Vgl. [Wu et al. 96: 19-48].

- Die Bedeutung eines Ausdrucks, der aus mehreren Teilausdrücken besteht, ist die Beziehung zwischen den mit den Teilausdrücken assoziierten Datenbankobjekten, ausgedrückt in Form des *Join-Pfades* (*join path*) zwischen den Datenbankobjekten. Join-Pfade sind Sequenzen von join-Operationen über zwei oder mehr Basis-Relationen.
Das semantische Lexikon enthält zu den Objekten und Beziehungen der Anwendungsdomäne die entsprechenden semantischen Repräsentationen.

Vereinfacht dargestellt, führen die folgenden Verarbeitungsschritte vom Syntax-Baum zur kanonischen Form:
(1) Den Terminalsymbolen des Syntax-Baumes werden entsprechende Informationen aus dem semantischen Lexikon, d.h. die semantischen Repräsentationen angehängt. Daraus resultiert der semantisch ergänzte Syntax-Baum.
(2) Unter Anwendung spezifischer *semantischer Translations-Regeln*, sog. *Transformation Rules*, wird aus dem semantisch ergänzten Syntax-Baum eine „underlying form" generiert, die sowohl syntaktische als auch semantische Information enthält. Die Verarbeitungsrichtung ist in diesem Fall bottom-up.
(3) Andere semantische Translations-Regeln sind für die Übersetzung der „underlying form" in die *Canonical Query Representation* zuständig.

Schritt (2) basiert auf folgenden Mechanismen:
(2.1) Mit jeder syntaktischen Regel ist eine Menge von semantischen Translations-Regeln assoziiert. Aus dem Syntax-Baum, der die angewandte Grammatik-Regel enthält, wird die korrespondierende Menge semantischer Translations-Regeln abgeleitet. Die bei Schritt (2) verwendeten *Transformation Rules* sind Pattern-Matcher, welche Struktur und semantischen Inhalt der Konstituenten spezifizieren; sie sind unterteilt in domänenunabhängige und domänenspezifische Regeln, wobei letztere erweitert werden können.
(2.2) Zur Disambiguierung semantisch ambiger NPs, d.h. NPs, welche mit verschiedenen Datenbankobjekten assoziiert werden können, verwendet die semantische Analyse sog. *Frames*. Frames sind denjenigen Relationen zugewiesen, welche Beziehungen zwischen Datenbankobjekten spezifizieren (z.B. „ENROLL", vgl. oben), und beschreiben die propositionalen Rollen der in der Relation enthaltenen Datenbankobjekte in der durch die Relation spezifizierten Beziehung im Sinne von *Selektionsrestriktionen*: z.B. „students take courses from professors". Im Falle einer ambigen NP wird die mit dem Verb des Satzes assoziierte Relation geholt und der entsprechende Frame zur Disambiguierung verwendet.
(2.3) Nachdem alle NPs semantisch eindeutig bestimmt sind, wird der *Pathfinder* aufgerufen, um die Nichtterminalsymbole des Syntax-Baumes semantisch zu analysieren bzw. um die semantische Struktur des gesamten Satzes zu konstruieren.
Der Pathfinder verwendet ein Modell der Anwendungsdomäne, um automatisch die meisten Join-Pfade, welche bei der semantischen Interpretation benötigt werden, zu generieren. Das Modell der Anwendungsdomäne wird vom System direkt aus dem Datenbank-Schema unter „minimaler" Assistenz von jemandem, der mit der Datenbank vertraut ist, gewonnen. Wenn eine Menge von zwei oder mehr semantisch bestimmten NPs respektive eine Menge von zwei oder mehr Attributnamen gegeben ist, liefert der Pathfinder (aufgrund der Bedeutung der NPs) einen „best guess" des „richtigen" Join-Pfades.

Aus der kanonischen Form generiert SYSTEM X zuerst die sog. *logische Form* (*Logical Form*), d.h. einen Ausdruck des *Domänen-Relationenkalküls*[52], und erst dann die SQL-Form. Der zusätzliche Abbildungsschritt ist insofern unproblematisch, als SQL unter anderem auf Aspekten des Relationenkalküls basiert.

4.4.2 Portabilität

Die folgenden Eigenschaften, Mechanismen und Module von SYSTEM X dienen (u.a.) dazu, die Menge des vom Benutzer beizusteuernden Domänenwissens zu minimieren bzw. die Domänen-Portierung zu vereinfachen:

- Sowohl das syntaktische als auch das semantische Lexikon bestehen aus einem domänenunabhängigen und einem domänenspezifischen Teil. Bei der Domänen-Portierung muss entsprechend nur jeweils der domänenspezifische Teil konfiguriert werden.
- Dank MORPHOS wird der domänenspezifische Teil des syntaktischen Lexikons während des Gebrauchs (halb-)automatisch aufgebaut bzw. erweitert. Prinzipiell müssen nur die Nennformen der regelmässig flektierten sowie alle Wortformen der unregelmässig flektierten Wörter manuell eingefügt werden.
- Der domänenunabhängige Teil des semantischen Lexikons spezifiziert z.B. Quantoren und Aggregatfunktionen und kann prinzipiell ohne Anpassungen portiert werden. Mit relativ hohem Aufwand ist allerdings die Konfigurierung des domänenspezifischen Teils verbunden: Obwohl ein grosser Teil des (domänenspezifischen) semantischen Lexikons automatisch aus dem Datenbank-Schema extrahiert wird, bleibt die Definierung von domänenspezifischen Synonymen sowie von Verben, die nicht mit einer Relation assoziiert werden können, die Aufgabe des Benutzers, vermutlich eines Wissensingenieurs.

Keine Unterstützung bietet SYSTEM X hingegen bei der Spezifizierung von domänenspezifischen Syntaxregeln, domänenspezifischen Semantikregeln und Frames. Allerdings sind z.B. die Frames nur bei Basis-Relationen zu generieren, welche Beziehungen zwischen Objekten anderer Relationen repräsentieren, und diese dürften in den meisten (Datenbank-)Fällen nicht allzu zahlreich sein.

5 Faktoren für die mangelnde Akzeptanz

Während die vorangehenden Kapitel Einblicke in die Themenbereiche Datenbanken und natürlichsprachliche Schnittstellen im Allgemeinen sowie Front-End-NLIDB im Speziellen vermitteln, soll hier die Beschäftigung mit der einen Leitfrage der Arbeit, nämlich diejenige nach den Faktoren, welche für den deutlichen Rückgang des Interessens an NLIDB verantwortlich sein könnten, im Zentrum stehen.
Vermutet werden entsprechende Faktoren einerseits im gesamten IT-Umfeld und andererseits bei den NLIDB-Systemen selbst, welche den hohen Erwartungen (vgl. Kapitel **3.2**) nicht (oder zumindest nicht in jeglicher Hinsicht) gerecht werden.

[52] Das *Relationenkalkül* ist eine formale Sprache, die auf dem Prädikatenkalkül (erster Stufe) basiert. Die expressive Mächtigkeit des Relationenkalküls ist identisch mit derjenigen der *Relationenalgebra*; im Gegensatz zur prozeduralen Relationenalgebra zählt das Relationenkalkül allerdings zu den deklarativen Sprachen. Es existieren zwei verbreitete Varianten: das *Tupel-Relationenkalkül* und das *Domänen-Relationenkalkül*. Vgl. [Elmasri and Navathe 94].

5.1 Alternative Benutzerschnittstellen

Noch zu Beginn der 1980er-Jahre verläuft die Interaktion zwischen Mensch und Computer in den meisten Anwendungsbereichen über Kommandozeilen-Schnittstellen. Seit Mitte der 1980er-Jahre jedoch entwickelt sich das GUI-Paradigma, das durch Verwendung intuitiver Bild-Metaphorik und durch Unterstützung direkter Objekt-Manipulation gerade dem Endbenutzer Vorteile verschaffen soll, nicht zuletzt dank immer leistungsfähigerer Hardware zur ernsthaften Alternative. [Manaris 98]

Im Bereich der Datenbankschnittstellen manifestiert sich dieser Umstand in der zunehmenden Verbreitung von Formular-basierten und graphischen Schnittstellen, die gegenüber natürlichsprachlichen Eingabezeilen-Schnittstellen mehr Bedienungskomfort versprechen, prinzipiell einfacher zu konfigurieren (und zu entwickeln!) sind und, besonders kritisch, dem Benutzer die verfügbare Funktionalität anzeigen. Diese Konkurrenz „im eigenen Lager“ wird denn auch einer der Gründe dafür sein, dass das Interesse an NLIDB-Systemen noch vor dem eigentlichen „Durchbruch“ wieder nachgelassen hat.

5.2 Schwächen von NLIDB-Systemen

In den folgenden Unterkapiteln soll diskutiert werden, inwiefern bzw. wo die verbreiteten Front-End-Systeme Mängel aufweisen, die als (massgebliche) Faktoren für die „NLIDB-Misere“ seit Anfang der 1990er-Jahren in Frage kommen. Kapitel **5.2.1** greift das Thema aus der computerlinguistischen Perspektive auf und fasst zusammen, welche Probleme bei der Verarbeitung natürlicher Sprache sich im Anwendungsbereich der NLIDB als besonders gravierend erweisen. Kapitel **5.2.2** vertritt die Benutzer-Perspektive und weist auf Schwachpunkte in der Bedienbarkeit der NLIDB hin. Kapitel **5.2.3** schliesslich betrachtet das typische Front-End-System aus einer quasi-ökonomischen Perspektive und versucht, Aufwand und Nutzen bei Entwicklung und Betrieb abzuwägen.

5.2.1 Sprachverstehen

Die Front-End-typische Zwischenrepräsentation basiert auf einer vollständigen Analyse natürlichsprachlicher Äusserungen, die gerade im Fall von „Real-World“-Äusserungen aufgrund unausgereifter NLP-Methoden häufig nicht möglich ist.

Zusätzliche Probleme generiert die Front-End-typische Trennung zwischen Sprachverstehens-Komponente und Datenbank-Back-End, und dies gerade im Fall der NLI zu „Real-World“-SQL-Datenbanken. Die Analyse der natürlichsprachlichen Eingabe (Datenbankwerte und Rechtschreibkorrektur), die Abbildung der Zwischenrepräsentation auf einen anwendungsspezifischen Formalismus (inkonsistente semantische Modelle) sowie die Generierung kooperativer Antworten (Diskursfähigkeit als prinzipieller Vorteil!) wird häufig dadurch erschwert oder gar verhindert, dass keine effiziente Methode existiert, während der Analyse des Eingabesatzes auf die Datenbankwerte, auf das semantische Modell der Datenbank oder auf die Abfrage-Steuerung des DBMS zuzugreifen. Der Ansatz von SYSTEM X scheint, zumindest was das Problem der inkonsistenten semantischen Modelle angeht, zukunftsweisend zu sein.

Viele Front-End-Systeme vernachlässigen schliesslich Diskursanalyse und Pragmatikanalyse:

- Wie bereits in Kapitel **4.2.2** erwähnt, stellt die natürliche Sprache Mittel zur Verfügung, um effizient zu kommunizieren. Diesen prinzipiellen Vorteil natürlichsprachlicher Abfragesprachen gilt es allerdings zu relativieren: Während das zeitweise kommerziell erhältliche NLIDB LOQUI wenn auch keine Ellipsen, so zumindest anaphorische Referenzen (unter gewissen Voraussetzungen) verarbeitet,[53] so operieren Systeme wie MASQUE/SQL und SYSTEM X gänzlich ohne Unterstützung satzübergreifender Konstruktionen. Dies

[53] Ähnlich wie ASK, vgl. [Androutsopoulos et al. 95].

mag u.a. daran liegen, dass allgemeine Methoden der Diskursrepräsentation erst seit Anfang der 1980er-Jahre Verbreitung finden.[54]

- NLIDB-Systeme sind häufig als reine Datenwiedergewinnungs-Instrumente konzipiert, was ihren Einsatzbereich beträchtlich einschränkt. So lässt sich zwar meistens das linguistische und konzeptuelle Wissen der Schnittstelle anpassen, nicht aber der Datenbestand der zugrunde liegenden Datenbank manipulieren. Dies könnte darauf zurückzuführen sein, dass natürlichsprachliche Insert- und Update-Aktionen aufgrund von semantischen Ambiguitäten[55] häufig zu unerwünschten Nebeneffekten führen, wobei diese ohne Kenntnis der Datenbankstruktur nicht nachvollziehbar und ohne Auflösung der Benutzerabsichten kaum zu vermeiden sind. [Androutsopoulos et al. 95]

Die fehlende (linguistisch fundierte) Verarbeitung von Sprechakten äussert sich entsprechend darin, dass viele NLIDB-Systeme Aussagen nicht als Anweisungen zur Datenmanipulation, sondern als implizite Anfragen, d.h. als zu prüfende Thesen interpretieren – was Update- und Insert-Aktionen nur unter Verwendung expliziter formaler Sprachkonstrukte erlauben würde.

5.2.2 Handhabung und Habitabilität

Interaktion mit NLIDB-Systemen aus den 1980ern bzw. frühen 1990ern funktioniert primär über die Eingabezeile. Daraus resultieren Probleme bei der Benutzerführung, welche NLIDB-Systeme im direkten Vergleich mit graphischen Schnittstellen ungeachtet der dahinter steckenden Funktionalität diskreditieren.

Ein häufiges Problem besteht darin, dass die linguistischen Fähigkeiten und das Domänenwissen des NLIDB-Systems für den Benutzer nicht offensichtlich sind. Man spricht in diesem Zusammenhang von der *Habitabilität* eines Systems, einem Bewertungskriterium, welches die Fähigkeit der Benutzer, bei der Formulierung der Anfrage innerhalb der vom System gesetzten Grenzen zu bleiben, beurteilt. [Dahl et al. 00]

Während der Sprachumfang formaler Abfragesprachen (bzw. die Mächtigkeit Formular-basierter oder graphischer Schnittstellen) typischerweise bekannt (oder sogar sichtbar) ist, kann das Formulieren einer Anfrage, welche gemäss den syntaktischen Regeln wohlgeformt ist und keine unbekannten Wörter enthält, gerade bei Systemen mit kontrollierter Sprache, d.h. mit relativ freien Ausdrucksmöglichkeiten, eine mühsame Angelegenheit sein. LOQUI z.B. verarbeitet die Objekt-Koordination in „the budget and the duration of the projects that ef leads?“ problemlos, während Subjekt-Koordination wie in „what do ds and jlb lead?“ nicht unterstützt ist. [LOQUI 91: 17-18] Der Erfolg der ersten Anfrage könnte den Benutzer z.B. zur falschen Annahme verleiten, dass das System Koordination bzw. Konjunktionen generell „versteht“. Noch schwieriger ist die Situation bei MASQUE/SQL: Während „What are the capitals of the countries bordering the Baltic and bordering Sweden?“ problemlos ist, kann das System dieselbe Anfrage ohne Wiederholung der kompletten Partizip-Verbalphrase („What are the capitals of the countries bordering the Baltic and Sweden?“) nicht beantworten. [Androutsopoulos and Ritchie 00: 7]

Habitabilitäts-fördernd würden sich z.B. informative, von der Sprachverstehens-Komponente generierte Fehlermeldungen auswirken. Allerdings sind die entsprechenden Antwort- und Fehlerbehandlungs-mechanismen häufig unausgereift, so dass der Benutzer beim Fehlschlagen einer Anfrage beispielsweise nicht weiss, ob das System den konzeptuellen (Wissens-)Gehalt oder die syntaktische Konstruktion nicht verstanden hat.

---

[54] Z.B. in Form der *Diskursrepräsentationstheorie*, vgl. [Carstensen et al. 01].

[55] Ein Ausdruck der Bedeutungsrepräsentation ist dann semantisch ambig, wenn er mit verschiedenen Datenbankobjekten korrespondiert. Vgl. Kapitel **4.4.1**.

Menü-basierte NLIDB sind zwar Prinzip-bedingt nicht von diesem Problem betroffen, allerdings auch nur solange effizient einsetzbar, als die Auswahl der verfügbaren natürlichsprachlichen Konstrukte eine gewisse kritische Grenze nicht überschreitet.

Was die „Natürlichkeit" der natürlichsprachlichen Schnittstelle angeht, so scheinen einerseits hinsichtlich der Eingabemöglichkeiten gewisse Einschränkungen zu bestehen, andererseits hinsichtlich des gesamten Systemverhaltens ebendiese zu fehlen:
- [Dahl et al. 00] äussert die Vermutung, dass ein Grund für den Misserfolg kommerziell vertriebener Produkte wie INTELLECT, Symantec's Q&A und BBN's PARLANCE darin liegen könnte, dass sie, trotz entsprechender Nachfrage auf Kundenseite, keine Möglichkeit zur Verarbeitung gesprochener Sprache boten. Die dazu notwendigen Technologien sind zumindest in der ersten Hälfte der 1990er-Jahre noch zu wenig weit entwickelt.
- Zumindest der naive Benutzer scheint oft insofern eine überhöhte Erwartungshaltung gegenüber dem NLIDB zu haben, als er vom System einen gewissen Grad an Intelligenz erwartet – z.B. im Sinne des „guten Menschenverstandes" oder der Fähigkeit, über vorhandenen Daten logische Schlussfolgerungen anzustellen. Man vermutet, dass die anthropomorphe Sprachfähigkeit des Systems den Benutzer zu dieser Fehlannahme verleitet. Bei der Verwendung formaler Abfragesprachen oder graphischer/Formular-basierter Schnittstellen ist eine derartige Überschätzung Prinzip-bedingt ausgeschlossen.

5.2.3 Kosteneffektivität

Aufgrund der Komplexität der natürlichsprachlichen Analyse im Bereich der NLIDB und mangels Verfügbarkeit standardisierter Komponenten ist die Entwicklung eines NLIDB eine teure Angelegenheit. Dazu kommt der hohe Zusatzaufwand der domänenspezifischen Konfigurierung, der sowohl bei der Inbetriebnahme als auch bei der Domänen-Portierung trotz aller Bemühungen zur Steigerung der Portabilität durch Automatisierung bestimmter Teilprozesse anfällt. [Winiwarter 97] spricht vom „knowledge acquisition bottleneck":

> One of the main difficulties for the practicable use of natural language interfaces is the required high amount of manual knowledge acquisition. For each portation to a new application domain the developer has to repeat this elaborate process. Furthermore, the interfaces are often part of dynamic environments with constant changes concerning the addressed topics or the user population.

Da NLIDB-Systeme nur Untermengen der natürlichen Sprache verarbeiten, muss der Benutzer zudem geschult werden: Schliesslich benötigt er Informationen darüber, welche Art von Fragen das System verarbeiten kann.
Kosteneffektivität, vereinfacht ausgedrückt, bedeutet die Nivellierung von Kosten und Mehrwert, die durch das System entstehen. [Dahl et al. 00] In Anbetracht der hohen Kosten bei Entwicklung und Konfiguration eines NLIDBs fällt ein diesbezüglicher Vergleich zwischen graphischen Benutzerschnittstellen und NLIDB-Systemen für letztere in den meisten Fällen äusserst ungünstig aus. Dies ist wohl ein wesentlicher Grund dafür, dass die meisten kommerziell einsetzbaren Produkte bis Mitte der 1990er-Jahre vom Markt verschwunden sind.

6 Neue Lösungsansätze

Prinzipiell gilt, dass bei einem NLIDB die Ansprüche ans Sprachverstehen aufgrund der typischerweise begrenzten Anwendungsdomäne wesentlich geringer sind als bei anderen NLU-Systemen, wenn auch

ähnliche Probleme auftauchen. Wie in Kapitel **5** beschrieben, kommt es bei einer natürlichsprachlichen Benutzerschnittstelle allerdings nicht alleine auf deren „linguistische Performanz“, sondern, dies eine relativ neue Perspektive, auch darauf an, dass der Benutzer die Möglichkeiten der Schnittstelle optimal einsetzen kann.
Die vielerorts geäusserte Kritik am Front-End-System der 1980er-Jahre ist nicht ungehört verhallt: Verschiedene neuere Ansätze mit z.T. divergierenden Fokussen versuchen, das Problem des Sprachverstehens, das Problem der Habitabilität und das Problem der Kosteneffektivität zu lösen. In den folgenden Unterkapiteln sollen einerseits einige dieser Ansätze theoretisch vorgestellt und andererseits, nicht zuletzt unter dem Aspekt der Kosteneffektivität, die Ergebnisse der praktischen Arbeit mit einem „modernen“ Front-End-System präsentiert werden.

6.1 Dialogsysteme

Bretan und Karlgren führen das häufige Scheitern der Mensch-Maschine-Kommunikation an natürlichsprachlichen Retrievalsystemen auf das sog. *„One-shot“-Problem* zurück:

> The whole premise that a user would be able to frame a query with a well-defined content in terms of the system knowledge base without a discourse context is alien to the nature of natural language – one-shot dialogues occur rarely in natural discourse. [Bretan and Karlgren 93: 8]

Aus den Beobachtungen, dass Fehlanalysen nicht nur bei den linguistisch und konzeptuell eingeschränkten NLI, sondern auch bei der Mensch-Mensch-Kommunikation auftreten, dass gerade letztere jedoch Möglichkeiten bietet, trotzdem einen erfolgreichen Diskurs zu führen, ziehen sie die Erkenntnis, dass der Einsatz solcher erfolgsversprechender Diskursstrategien auch im Bereich der NLI zu besseren Resultaten hinsichtlich der Habitabilität und des Gelingens der natürlichsprachlichen Analyse führen sollte.
Während die Möglichkeiten der Diskursgestaltung an einem NLIDB, wie bei der Diskussion der Front-End-Systeme bereits beobachtet, durch die Beschränkung auf mehr oder weniger isolierte Frage-Antwort-Einheiten meistens sehr limitiert sind, basiert der natürliche Diskurs auf dem sowohl *interaktiven* als auch *inkrementellen Dialog*. Inkrementell bedeutet, dass der bereits geführte Diskurs bei der Analyse bzw. Generierung eines Diskursbeitrages mit berücksichtigt wird, wodurch der Wissensstand der Diskursteilnehmer kontinuierlich steigt. Im Gegensatz zur „One-shot“-Strategie, bei welcher die Benutzerabsicht vollständig und explizit in einer einzigen (potentiell komplexen) Anfrage ausgedrückt werden muss, ermöglicht die Dialog-Strategie eine stetige interaktive Präzisierung der Benutzerabsicht auf der Basis von einfachen (Teil-)Anfragen auf Benutzerseite sowie von kooperativen Antworten als diskursleitende Systemreaktionen.
In Bezug auf letztere wird bereits in Kapitel **4.2.3** erwähnt, dass z.B. das Problem der inkonsistenten Benutzerannahmen nur unter Berücksichtigung eines grösseren Diskurskontextes adäquat behandelt werden kann – was folglich eine Implementierung von Strategien der Dialogführung und Dialogrepräsentation notwendig erscheinen lässt.

6.1.1 Die Architektur eines typischen Dialogsystems

Voraussetzung für das Führen eines Dialogs ist, dass alle Diskursteilnehmer über entsprechende kommunikative Fähigkeiten verfügen. Bezüglich eines NLIDB-Systems bedeutet dies, dass die Analyse-Komponente für das *Sprachverstehen* und die häufig vernachlässigte Generierungs- bzw. Antwort-Komponente für die *Sprachgenerierung* gleichermassen ausgereifte Fähigkeiten aufweisen müssen.

Um dieser Anforderung möglichst kosteneffektiv zu entsprechen, sind *Dialogsysteme* meistens mehr oder weniger symmetrisch konzipiert:

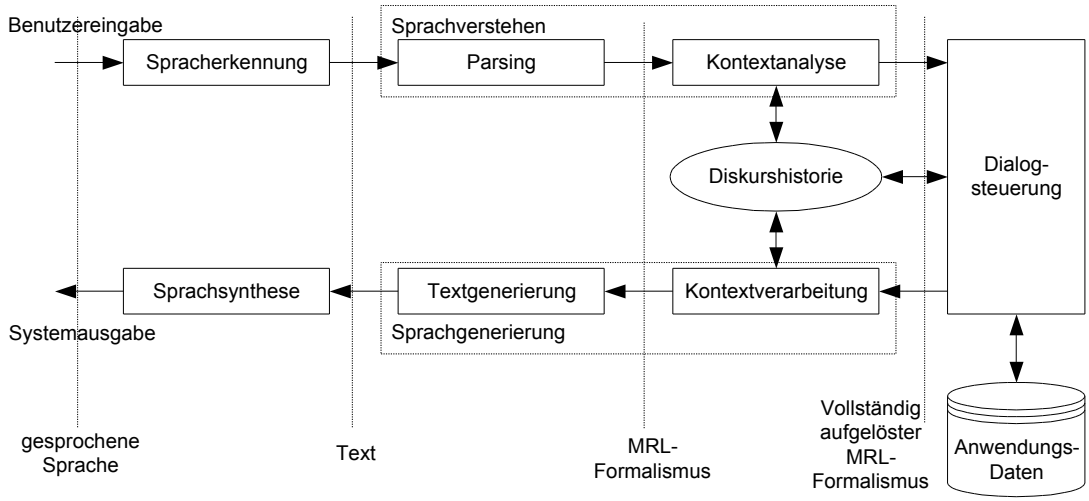


Abbildung 3 - Systemarchitektur eines Dialogsystems mit Unterstützung gesprochener Sprache[56]

Im Unterschied zur Front-End-Architektur verfügt das Dialogsystem über eine zentrale Komponente zur Steuerung der Interaktion mit dem Benutzer: die *Dialogsteuerung*. Zu deren Aufgaben gehört einerseits die *Diskursmodellierung*, d.h. die Verwaltung der *Diskurshistorie* und die *Benutzermodellierung*,[57] und andererseits die *Aktionsplanung*, welche unter Berücksichtigung sowohl der Interpretation der Benutzeräusserung als auch der Diskurshistorie eine möglichst angemessene Systemreaktion festlegt. Angemessene Reaktionen sind je nach Kommunikationssituation z.B. die Präsentation der Ergebnisse, die Initiierung von Klärungsdialogen zur Disambiguierung, die Anzeige von Paraphrasen zur Verifikation der Interpretation oder die Rückgabe informativer Fehlermeldungen.

6.1.2 KDA: ein Front-End-System mit Dialogfähigkeiten

Das „intelligente“ Datenbank-Front-End KDA („Knowledge-based Database Assistant“) [Wu et al. 96] stellt weniger ein „typisches“ modernes Dialogsystem (vgl. oben) als vielmehr den Prototypen eines Front-End-Systems, welches über erweiterte Fähigkeiten zur Analyse natürlichsprachlichen „echten“ Dialogs verfügt, dar.

Die Benutzerschnittstelle von KDA bietet zur Formulierung der natürlichsprachlichen Anfrage sowohl einen Eingabezeilen-Modus als auch einen Menü-basierten Modus.

Das System basiert auf dem *semantischen Netzwerk S-Net*, dem Domänenmodell, welches das für die Interpretation der Benutzeranfrage benötigte Wissen repräsentiert:

- Domänenspezifische Konzepte.
- Die korrespondierenden Datenbank-Konzepte.
- Regeln, welche die domänenspezifischen Konzepte auf Datenbankkonzepte abbilden.

[56] Vgl. [Carstensen et al. 01: 486], leicht modifizierte Fassung.
[57] Zur Benutzermodellierung vgl. [Carstensen et al. 01: 316-330].

- Informationen, welche die für die Analyse verwendeten domänenspezifischen Konzepte auf diejenigen beschränken, welche mit der jeweiligen *Benutzerperspektive*, einer Repräsentation der Interessen einer Benutzergruppe, assoziiert sind.
- Informationen, welche die sog. *Query Patterns* (siehe unten) mit domänenspezifischen Konzepten assoziieren.

Der Parser generiert entsprechend S-Net-konforme Bedeutungsrepräsentationen, und der Datenbankabfrage-Generator transformiert vollständig aufgelöste S-Net-Formalismen in ausführbare SQL-Abfragen.

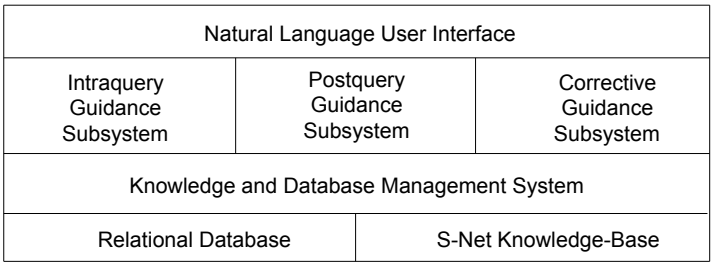


Abbildung 4 - Systemarchitektur von KDA[58]

KDA hebt sich durch drei Subsysteme, welche auf je spezifische Art und Weise dafür sorgen, dass der Benutzer innerhalb der linguistischen und konzeptuellen Abdeckung des Systems bleibt, von konventionellen Front-End-Systemen wie z.B. SYSTEM X ab:

- Das Menü-basierte *Intraquery Guidance Subsystem* erleichtert dem ungeübten Benutzer die Formulierung einer gültigen Anfrage, indem es eine Auswahl von Anfrage-Mustern (*Query Patterns*), d.h. Vorlagen gültiger Satzstrukturen („Find the ( ) of ( ) who ( ) ( )“), zur Verfügung stellt. Der Benutzer vervollständigt die Satzvorlagen durch Einfügen passender semantischer Konzepte, Attribute von semantischen Konzepten und konstanter Werte, wobei auch unvollständige Anfragen analysiert werden können.
- Das *Postquery Guidance Subsystem* unterstützt den Benutzer beim Formulieren einer gänzlich neuen Anfrage und beim Präzisieren oder Modifizieren einer vorangehenden Anfrage, indem es sog. „Anfrage-Skelette“ (*Query Skeletons*) anzeigt. Diese Anfrage-Skelette werden automatisch und kontextsensitiv, d.h. abhängig von der vorangehenden Anfrage sowie von der gewünschten Aktion (neue Anfrage stellen – Anfrage präzisieren – Anfrage modifizieren) erzeugt und beschreiben einen Teil der darunterliegenden Datenbank respektive des semantischen Domänenmodells in natürlichsprachlicher Form („A professor directs a laboratory“).
- Das *Corrective Guidance Subsystem* ist für die Generierung kooperativer Antworten zuständig. Im Gegensatz zu Systemen wie z.B. LOQUI erkennt es nicht nur falsche, sondern auch inkonsistente Benutzerannahmen und generiert entsprechende informative Fehlermeldungen.

[58] Vgl. [Wu et al. 96: 61].

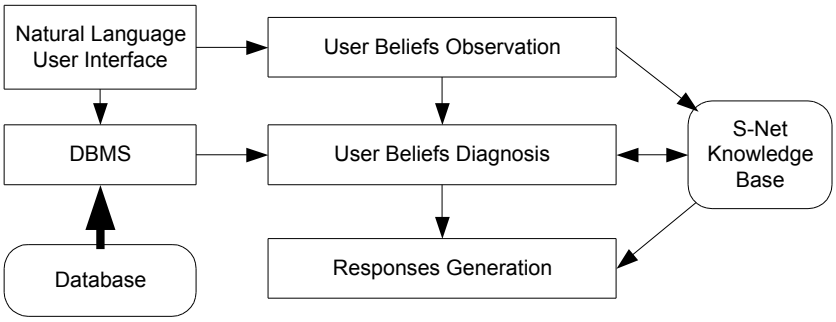


Abbildung 5 - Corrective Guidance Subsystem von KDA[59]

Das *User Beliefs Observation Module* extrahiert aus den natürlichsprachlichen Anfragen die Benutzerannahmen (Präsuppositionen) und generiert ein dynamisches „Kurzzeit"-Modell der Benutzerannahmen in Form einer sog. *Users-Beliefs-Hierarchy*.
Das *User Beliefs Diagnosis Module* wird aktiviert, sobald das Resultat einer Anfrage der NULL-Wert ist. Es überprüft Korrektheit und Konsistenz der Benutzerannahmen (bzw. der User-Beliefs-Hierarchy) anhand der S-Net-Wissensbasis und der relationalen Datenbank bzw. unter Auswertung von *Konsistenzregeln* (*Contraint Rules*) der Datenbank.[60]
Das *Responses Generation Module* wird vom *User Beliefs Diagnosis Module* aktiviert, falls falsche oder inkonsistente Benutzerannahmen existieren. Es generiert detaillierte natürlichsprachliche Fehlermeldungen – ein Beispiel der Korrektur einer inkonsistenten Benutzerannahme ist in Kapitel **4.2.3.3** aufgeführt.

Abgesehen von der erweiterten Funktionalität im Bereich der kooperativen Antworten ist KDA auch als Konzept einer *hybriden* Schnittstelle interessant. Hybride Schnittstellen kombinieren natürlichsprachliche Texteingabefelder mit sog. *deiktische* Kontrollelementen[61] wie Buttons, Menüs usw. und besitzen gegenüber rein Menü-basierten Systemen wie NLMENU [Androutsopoulos et al. 95] den Vorteil, dass die Effizienz und Flexibilität (kontrollierter) natürlicher Sprache bei ähnlich positiver Wirkung auf die Habitabilität nicht eingeschränkt wird.
Bei KDA braucht die Anfrage dank der *Query Patterns* nicht Wort für Wort bzw. Konstituente für Konstituente zusammengestellt zu werden, und die einzufügenden Konzepte können innerhalb der konzeptuellen Grenzen des Domänenmodells relativ frei gewählt werden.

6.2 Multimodalität

Der übliche Ansatz, der Probleme des Sprachverstehens bei NLIDB-Systemen Herr zu werden, besteht darin, die linguistischen Analysewerkzeuge der Komplexität der Aufgabenstellung anzupassen. Diese an anderen NLP-Anwendungsbereichen orientierte Vorgehensweise hat zwar im Verlaufe der Jahre zu stetig raffinierteren (und komplexeren) Systemen geführt, aufgrund der gestiegenen Anforderungen u.a. an die Effizienz sowie an die Benutzerfreundlichkeit jedoch die Akzeptanz gegenüber NLIDB-Systemen nicht verbessern können (vgl. Kapitel **5**).

[59] Vgl. [Wu et al. 96: 63].
[60] Vgl. z.B. [Rechenberg und Pomberger 99].
[61] [Hall et al. 96] unterscheidet zwischen sog. „edit controls", d.h. GUI-Komponenten, welche Texteingaben entgegennehmen, und sog. „deictic controls", d.h. GUI- Komponenten, worauf der Benutzer mit einem Zeigegerät (z.B. der Maus) „zeigt" respektive klickt, um eine Option auszuwählen.

Der Ansatz, der mit dem Begriff der *Multimodalität* verbunden ist, geht sowohl Probleme der Mensch-Maschine-Kommunikation als auch Probleme der natürlichsprachlichen Analyse an. Er zielt darauf ab, dem Benutzer neben einer natürlichsprachlichen Eingabemöglichkeit ergänzende *Modalitäten*[62] zur Verfügung zu stellen und dadurch sowohl die Mensch-Maschine-Interaktion effizienter und benutzerfreundlicher zu gestalten als auch die linguistische Analyse zu vereinfachen.

Offensichtlich beinhaltet die Mensch-Mensch-Kommunikation zusätzlich zur natürlichen Sprache Modalitäten wie z.B. Zeigen, Zeichnen und diverse andere Gesten. [Oviatt and Cohen 91] Je nach Kommunikationskontext ist dabei die eine Modalität geeigneter als die andere: Bestimmte Dinge lassen sich am besten durch Zeigen, Auswählen aus einer Liste usw. in die Kommunikation miteinbeziehen.
Die HCI-Forschung überträgt diese Erkenntnis auf den Bereich der Schnittstellen und behauptet, dass natürliche Sprache (in geschriebener oder gesprochener Form) trotz ihrer unbestrittenen Vorzüge nicht für alle Aufgaben an einer Benutzerschnittstelle gleichermassen geeignet ist, weil sie, je nach Kontext, viel zu umständlich ist. [Manaris 98] Als pointierte Beispiele für diese Umständlichkeit seien hier die folgenden (natürlichsprachlichen) Befehle aufgeführt, die angesichts der GUI-Möglichkeiten ziemlich absurd erscheinen: „make this window a little bigger“ bzw. „move the mouse pointer two centimeters to the right“ [Bretan and Karlgren 93: 2].

Multimodale Schnittstellen sollen zur Realisierung verschiedener Kommunikationsabsichten adäquate Modalitäten zur Verfügung stellen. Sie unterscheiden sich insofern von sog. *multimedialen Schnittstellen*, als sie Informationen verschiedener Eingabe- und Ausgabe-Medien (z.B. geschriebene und gesprochene Sprache, Graphik, Video u.a.) in derselben internen Repräsentationssprache darstellen und umgekehrt aus der einheitlichen internen Repräsentation für verschiedene Modalitäten entsprechend formatierte Inhalte generieren können. [Wilson et al. 91]
In den meisten Fällen kombinieren multimodale Schnittstellen die natürlichsprachliche Modalität mit einer graphischen Modalität, z.B. in Form einer sog. *Visual Language*[63]. Während sie die Flexibilität, Natürlichkeit und Effizienz der natürlichsprachlichen Modalität verfügbar machen, unterstützen sie den Benutzer bei der Formulierung der Anfrage in verschiedener Hinsicht zusätzlich [Bretan and Karlgren 93]:
- Eine Visual Language ist *eindeutig*. Entsprechend treten bei der Verwendung der graphischen Modalität keine Ambiguitäten auf, und Ambiguitäten in der natürlichsprachlichen Anfrage lassen sich mittels der Visual Language auflösen.
- Eine Visual Language ist *transparent*. Da die verfügbaren graphischen Objekte in irgendeiner Form dargestellt oder aufgelistet sein müssen, erhält der Benutzer einen Einblick ins semantische Domänenmodell.
- Eine Visual Language begünstigt den *inkrementellen* Interaktionsstil. Die Elemente der Visual Language bilden, im Gegensatz zur natürlichsprachlichen Anfrage, grundsätzlich einen persistenten Diskurskontext. Folgefragen oder Präzisierungen vorheriger Anfragen lassen sich aufgrund der einfachen „Syntax“ der Visual Language oft durch geringe Modifikationen formulieren.

Multimodale Systeme beherrschen häufig die Anzeige der Anfrage-Resultate in verschiedenen Modalitäten. So kann das Laborsystem MMI^2 (Man-Machine Interface for MultiModal Interaction with knowledge

[62] Gemäss [Coutaz and Caelen 91] bezeichnet der Begriff *Modalität* einen für Informationsübermittlung und Informationsaufnahme verwendeten Kommunikationskanal.
[63] Eine Visual Language kann man sich als graphische Notation einer logischen Sprache vorstellen – die semantischen Objekte und Beziehungen der Anwendungsdomäne sind entsprechend als Knoten und Kanten repräsentiert.

based systems) [Wilson et al. 91], konfiguriert für die Domäne „computer network design“, eine Antwort je nach Kontext, Resultat und Benutzerwunsch in natürlicher Sprache (Englisch, Französisch und Spanisch) oder als Diagramm (CAD-Diagramm, Datengraphik) formatieren, wodurch z.B. der folgende Dialog entsteht:

```
user: Display a bar-graph of the cost of the computers.
system: ok <displays bar-graph>

user: Display that as a pie chart.
system: ok <displays pie chart>
```

Im Hintergrund arbeitet ein zentraler *Dialog Controller*, der mit zahlreichen „Experten“-Modulen, d.h. Problem-spezifischen Subsystemen interagiert: So existieren für die verschiedenen Eingabe- und Ausgabe-Modalitäten entsprechende Subsysteme, dazu „Experten“ (u.a.) für den Dialogkontext, für die Benutzermodellierung, für die Kommunikationsplanung sowie für den Zugriff auf die Wissensbasis.

6.3 DOA: Integration in DB-Architektur

In den folgenden Unterkapiteln soll ein Framework vorgestellt werden, welches die in Kapitel **4.2.1** und **4.2.4** beschriebenen Probleme der Analyse und Interpretation zu lösen verspricht.
Winiwarter[64] vertritt den Standpunkt, dass nur eine integrierte Systemarchitektur die konsistente und effiziente Abbildung der semantischen Repräsentation der Benutzeranfrage auf das entsprechende semantische Anwendungsmodell garantiert:

> Nur die vollständige Integration der linguistischen Analyse in die Datenbankarchitektur ermöglicht es, die beiden Repräsentationsschemata auf natürliche und konsistente Weise ineinander überzuführen, womit der Analyseaufwand bei gleichzeitiger Maximierung der Analysequalität auf ein Minimum reduziert wird. [Winiwarter und Tjoa 96]

Der integrative Ansatz wird im sog. *Deductive Object-oriented Approach* (*DOA*), einem Framework für NLIDB-Systeme, konkretisiert. Den „Proof of concept“ liefert dabei die Entwicklung eines NLI für die Referenzsprache Japanisch sowie dessen Integration in die *Question Support Facility* des CSCW-Unterrichtssystems[65] VIENA (VIEw NAtural) CLASSROOM. Neben der Integration des NLP in die Datenbankarchitektur sind die vollständige und effiziente Repräsentation heterogener multimedialer Wissensinhalte (Text, Ton, Bild, Video), die optimale Kombination von natürlicher Sprache mit anderen Interaktions-Modalitäten sowie die effiziente dynamische Wissensrepräsentation bei sich ändernden Wissensinhalten von zentralem Interesse.

6.3.1 VIENA Classroom CSCW

(Die folgenden Angaben stammen hauptsächlich aus [Winiwarter et al. 96] und [Winiwarter und Tjoa 96].)
VIENA CLASSROOM (VC) basiert auf der *Client-Server-Architektur*: Lehrende und Studierende greifen von Clients aus auf den Vorlesungsdatensatz und die *Frequently Asked Questions*-Wissensbasis (FAQ-Wissensbasis), welche auf dem Server residieren, zu.
Die Unterrichtsmaterialien werden vom Lehrenden als Hypermedia-Dokumente generiert und den Studierenden innerhalb der CSCW-Umgebung präsentiert.
Die *Question Support Facility* (mit NLI-Erweiterung) ermöglicht Lehrenden und Studierenden den Zugriff auf die FAQ-Wissensbasis. Die Studierenden können jederzeit und dank der NLI-Erweiterung in

[64] Z.B. [Winiwarter 94], [Winiwarter 97], [Winiwarter und Tjoa 96].
[65] CSCW ist das Akronym zu „computer-supported cooperative work“.

natürlicher Sprache Fragen stellen, welche entweder direkt aufgrund der FAQ-Wissensbasis beantwortet oder zur Bearbeitung an den Lehrenden weitergeleitet werden. Eine Liste mit kontextsensitiven, auf die aktuelle Stelle der Unterrichtsmaterialien abgestimmten Beispiel-FAQs leistet Unterstützung bei der Formulierung der Fragen. Der Anspruch der optimalen Kombination verschiedener Interaktions-Modalitäten äussert sich darin, dass der Student den Kontext für seine Frage deiktisch, d.h. durch Markierung einer entsprechenden Stelle der Unterrichtsmaterialien, z.B. einer Phrase oder eines Bildes, definiert.

Lehrende erhalten durch die anhand der semantischen Repräsentation gruppierten Fragen einen raschen Überblick über die Unterrichtssituation und können Fragen in der Reihenfolge ihrer Häufigkeit respektive Relevanz beantworten. Aus FAQ-Häufigkeitsdaten entnehmen Lehrende Anregungen zur Verbesserung der Lehrmaterialien sowie, unmittelbarer, zur Generierung kontextsensitiver FAQ-Listen; im Idealfall müssen sie eine Frage nur einmal beantworten.

Zusätzlich besteht für Studierende die Möglichkeit, Notizen anzubringen, direkte Online-Verbindungen zum Lehrenden herzustellen sowie mit Unterrichtsmaterialien zu arbeiten, die aufgrund ihres Feedbacks personalisiert worden sind.

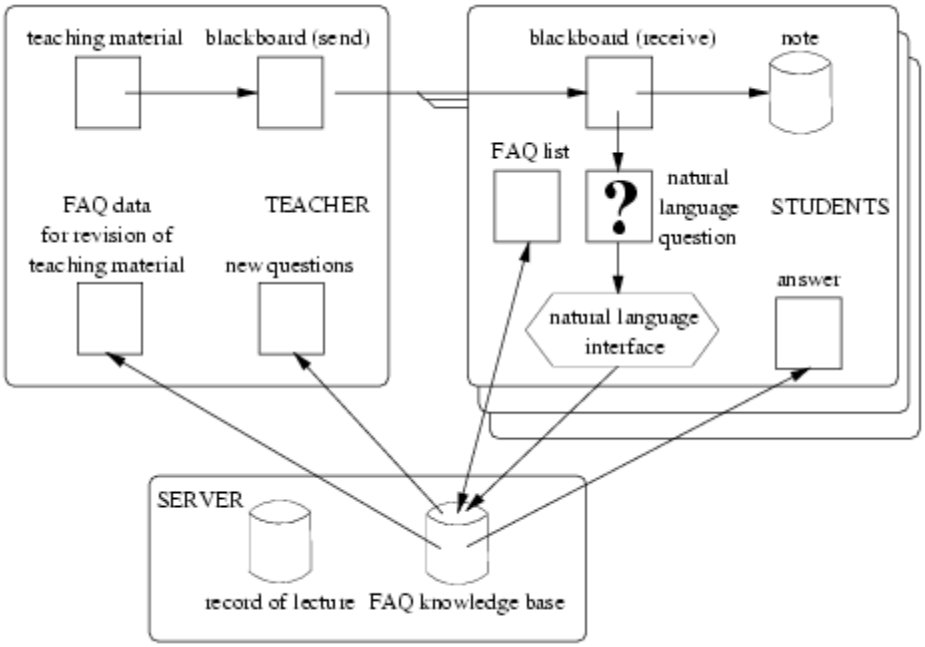


Abbildung 6 - VIENA CLASSROOM inklusive Question Support Facility und NLI[66]

6.3.2 Das NLIDB

(Die folgenden Angaben zur NLIDB-Erweiterung für VIENA CLASSROOM stammen vor allem aus [Winiwarter und Tjoa 96], [Winiwarter 97] und [Winiwarter et al. 97].)

Die Verwendung eines *deduktiven objekt-orientierten Datenbanksystems* (DOODBS)[67] ermöglicht es, die linguistische Analyse in die Datenbankarchitektur zu integrieren. Die logische und objekt-orientierte Programmiersprache kommt sowohl bei der Verarbeitung der natürlichsprachlichen Eingabe als auch bei

[66] Vgl. [Winiwarter et al. 96].

[67] Gemäss [Winiwarter et al. 97] handelt es sich dabei um das an der Heriot-Watt University entwickelte DOODBS ROCK & ROLL.

Datenwiedergewinnung und Datenmanipulation zum Einsatz, und die hierarchische, objekt-orientierte Datenstrukturierung sowie die entsprechenden Vererbungsmechanismen erlauben die kompakte Speicherung des domänenspezifischen Wissens als konzeptuelle Hierarchien.

Drei Hauptkomponenten konstituieren die natürlichsprachliche Schnittstelle:
- Die *lexikalische Komponente* generiert aus dem (japanischen) natürlichsprachlichen Eingabesatz eine Liste sog. *Deep Forms*, d.h. semantischer Repräsentationen einzelner Ausdrücke. Sie residiert auf der Client-Maschine, u.a. um eine gewisse linguistische Personalisierung zu ermöglichen.
- Die *semantische Komponente* generiert aus der *Deep Form List* (*DVL*) eine eindeutige, vollständige und anwendungsspezifische Bedeutungsrepräsentation des Eingabesatzes, d.h. den zu evaluierenden Ausdruck. Wie die lexikalische Komponente operiert sie auf der Client-Maschine und wird durch Gebrauch personalisiert.
- Die *Unterhaltskomponente* (*maintenance component*) dient der Konfiguration des NLI. Sie liegt zentral auf dem Server.

Jede der drei Hauptkomponenten verwendet sog. *adaptive* Technologien, um den Anforderungen der dynamischen Wissensrepräsentation bei sich ändernden Wissensinhalten zu entsprechen bzw. um den für die Konfiguration benötigten Aufwand zu minimieren. So werden z.B. Methoden des *Machine Learning* zur automatischen Wissensakquisition eingesetzt. (Adaptive Technologien sind in der Illustration der NLIDB-Systemarchitektur grau hinterlegt.)

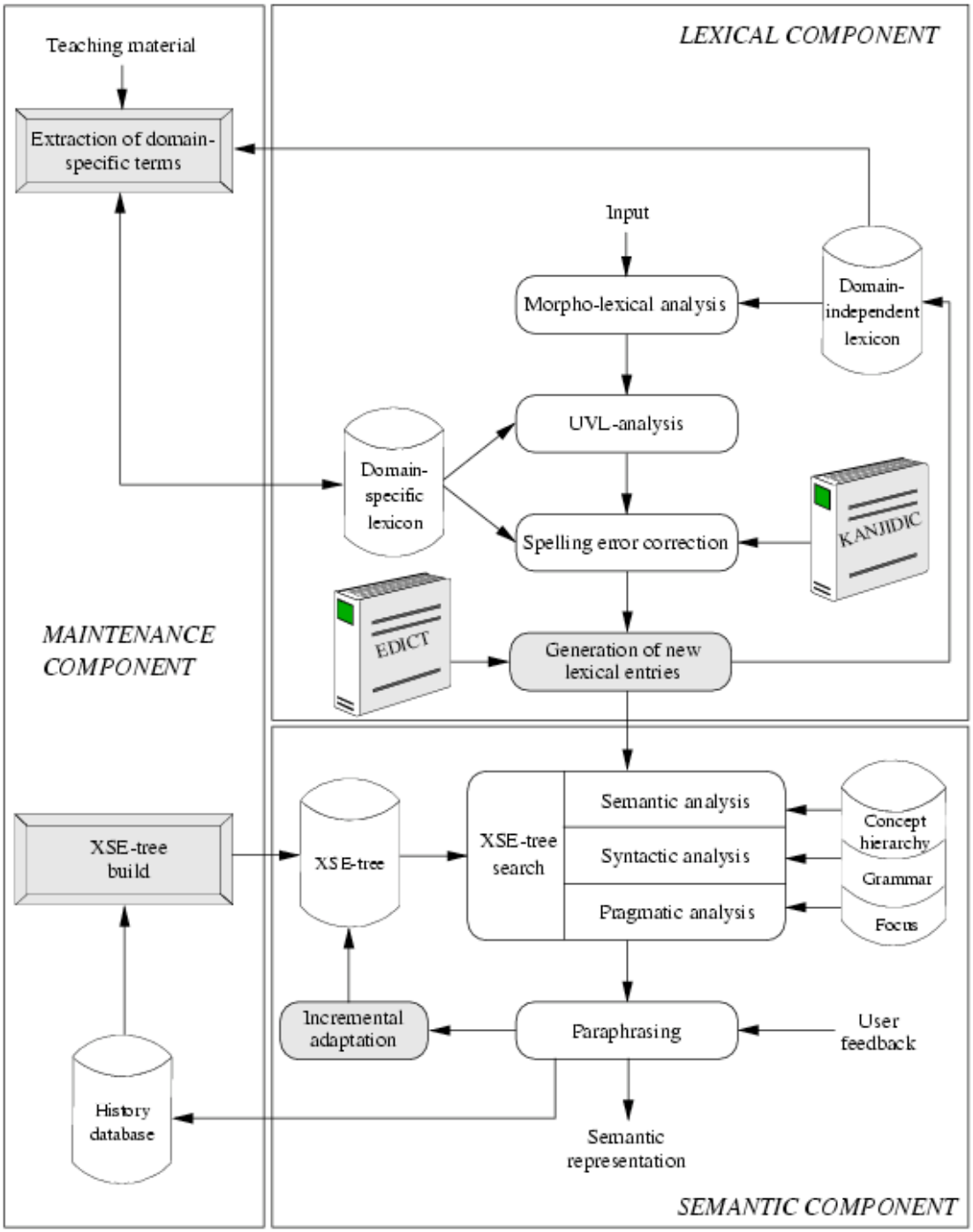


Abbildung 7 - Systemarchitektur des VIENA CLASSROOM NLI[68]

[68] Vgl. [Winiwarter et al. 97].

6.3.2.1 Lexikalische Komponente

Die lexikalische Komponente besteht aus drei NLP-Modulen für die morpho-lexikalische Analyse, die UVL-Analyse sowie die Schreibfehlerkorrektur einerseits und einem adaptiven Modul zur Generierung neuer Lexikoneinträge andererseits.

Die morpho-lexikalische Analyse zerlegt den japanischen Eingabesatz in einzelne Tokens und generiert daraus unter Konsultierung des domänenunabhängigen Wörterbuchs die *Deep Form List* (DFL). Zu den im domänenunabhängigen Wörterbuch enthaltenen Lexemen speichert die DFL die lexikalische Kategorie und Subkategorie[69] sowie, bei bedeutungstragenden Ausdrücken, eine Menge von zugehörigen *Deep Forms* , d.h. semantischen Beschreibungen. Zu den nicht im domänenunabhängigen Wörterbuch enthaltenen Lexemen speichert sie hingegen nur den Kategoriewert „unknown“ und den dem jeweiligen japanischen Schrift-Typ[70] entsprechenden Subkategoriewert.

Die *Unknown Value List*-(*UVL*-)Analyse versucht, Einträge der UVL, einer Zusammenstellung aller „unknown“-Werte der DFL, im domänenspezifischen Wörterbuch aufzufinden. Ist die Suche erfolgreich, wird der UVL-Eintrag gelöscht und die DFL mit den Angaben des domänenspezifischen Wörterbuchs korrigiert, andernfalls die externe Wörterbuch-Datei EDICT konsultiert.

Die Wörterbuch-Datei *EDICT*[71] dient als Zweitlexikon, das Informationen über unbekannte (siehe UVL) Ausdrücke liefert (was zur Korrektur der DFL und UVL führt) und die automatische Generierung neuer lexikalischer Einträge ins domänenunabhängige Lexikon ermöglicht. Da die Anpassung des domänen-unabhängigen Lexikons lokal beim Client geschieht, führt die Auswertung der linguistischen Benutzervorlieben zu einer Art automatischer linguistischer Personalisierung.

Die *automatische Schreibfehlerkorrektur* verwendet ebenfalls das domänenspezifische Lexikon, vergleicht inkorrekte, d.h. auch nach der UVL-Analyse noch „unbekannte“ Eingabewörter mit existierenden Einträgen und ersetzt sie durch passende „korrekte“ Ausdrücke.

Die externe Zeichen-Datei *KANJIDIC*[72] dient der Erhöhung der Präzision bei der Schreibfehlerkorrektur.

Mit der dreistufigen lexikalischen Verarbeitung wird bezweckt, dass die Anzahl unbekannter, d.h. vom System nicht analysierbarer Wörter auf ein Minimum reduziert und dadurch die Chance einer vollständigen und korrekten Interpretation des Eingabesatzes erhöht wird.

6.3.2.2 Semantische Komponente

Während die lexikalische Komponente zu den inhaltlich relevanten Ausdrücken des Eingabesatzes eine Menge von lexikalischen Bedeutungen ermittelt und insofern eine semantische Voranalyse vornimmt, generiert die semantische Komponente unter Verwendung des XSE-Baumes aus den Einzelbedeutungen eine eindeutige Bedeutungsrepräsentation des gesamten Eingabesatzes. [Winiwarter et al. 97]

Die zentrale Datenstruktur der semantischen Komponente stellt der *XSE-Baum*[73] dar, der Bedeutungsrepräsentationen bereits analysierter Eingabesätze in einer erweiterbaren und effizient zu

---

[69] Kategorie und Subkategorie spezifizieren neben der POS-Kategorie auch modale bzw. pragmatische Konnotationen wie z.B. den Höflichkeitsgrad, welche typisch fürs Japanische sind.

[70] Z.B. „kanji“ oder „katakana“ – das Japanische weist ein komplexes Schriftsystem mit mehreren Subsystemen auf.

[71] EDICT wird seit 1991 an der Monash University entwickelt und ist als Freeware erhältlich. Das Japanisch-Englisch-Wörterbuch umfasst 1997 ca. 60'000 Einträge, die jeweils das japanischen Lexem, dessen Betonung, eine Liste englischer Übersetzungen, syntaktische Features, Angaben zum „level of politeness“ usw. beinhalten. Vgl. [Winiwarter 97].

[72] KANJIDIC enthält Einträge zu 6355 Kanji-Zeichen (gemäss dem JIS X 0208-1990 Standard) mit Angaben zu verschiedenen „coding systems“, zur Betonung und zu korrespondierenden englischen Bedeutungen. Vgl. [Winiwarter 97].

verarbeitenden Form speichert und dadurch den *automatischen Wissenserwerb* aus früheren Eingabedaten ermöglicht. Gleichermassen Anwendungsgedächtnis und World Model im Sinne der Selektionsrestriktionen, stellt er die Wissenbasis dar, woraus die semantische Komponente die korrekte semantische Repräsentation für den Eingabesatz ermittelt.

Die Semantikanalyse greift auf eine *Konzepthierarchie* zu, welche die domänenspezifischen Deep Forms aufgrund ihrer semantischen Ähnlichkeit hierarchisch einordnet. Sie unterstützt die Suche im XSE-Baum, indem sie anhand des Grades semantischer Ähnlichkeit zwischen der Deep Form des jeweiligen DFL-Eintrags und den Deep Forms der Konzepthierarchie entscheidet, ob der DFL-Eintrag einem (bekannten) gültigen Konzept entspricht. So kann sie z.B. auch Synonyme zu bestehenden Konzepten erkennen.

Syntaxanalyse erfolgt nur dann, wenn die DFL ambig ist. Dank Bottom-Up-Parsing, basierend auf der *Erweiterten Kategorialen Unifikationsgrammatik* [Winiwarter 95], sollen auch unvollständige und ungrammatikalische Sätze effizient analysiert werden können. Aus den resultierenden Syntax-Bäumen wird unter Anwendung syntaktischer Disambiguierungsregeln die „korrekte" Lesart ausgewählt.

Die Pragmatikanalyse dient der Verarbeitung von Phänomenen wie Ellipsen und Referenzen: Falls die Semantikanalyse keine eindeutige und vollständige Interpretation liefern kann, wird sowohl in der vorherigen Bedeutungsrepräsentation als auch in der *Focus*-Datenstruktur nach den fehlenden Ausdrücken bzw. den Referenzobjekten gesucht und das Resultat für die Bestimmung des aktuellen Kontexts verwendet. Die *Focus*-Datenstruktur repräsentiert deiktische Kontexte, welche der Benutzer durch Auswählen von Bezugsstellen in der multimodalen Unterrichtsumgebung bestimmt hat.

Das Paraphrase-Modul generiert zur Kontrolle „repräsentative Fragen" für die aktuellen semantischen Repräsentationen, die als Resultat der Suche im XSE-Baum geliefert werden. Dazu verwendet es sog. *Templates*, d.h. Satzstruktur-Vorlagen mit variablen Rollen, sowie die Ausdrücke der DFL als Werte der Template-Variablen. Im Falle eines positiven Benutzerfeedbacks wird die korrekte Bedeutungsrepräsentation zur logischen Interpretation ans deduktive DBMS weitergeleitet; im Falle eines negativen Feedbacks, d.h. einer falschen semantischen Repräsentation, kann der Benutzer entweder „from scratch" eine neue semantische Kategorie definieren oder aus den bestehenden FAQ-Fragen eine passende Kategorie auswählen.

Die vom Paraphrase-Modul produzierte (bzw. vom Benutzer generierte) korrekte Semantikrepräsentation wird zusätzlich dem Modul für die inkrementelle Anpassung des lokalen XSE-Baumes und der auf dem Server gelegenen *History-Database*, d.h. der Grundlage für die Server-seitige Anpassung des globalen XSE-Baumes, zugeführt.

6.3.2.3 Unterhaltskomponente

Die Hauptaufgaben der Unterhaltskomponente liegen in der Extraktion domänenspezifischer Ausdrücke und im Bau des globalen XSE-Baumes.

Das adaptive Extraktions-Werkzeug dient der halbautomatischen Erweiterung des domänenspezifischen Lexikons. Es extrahiert neue domänenspezifische Ausdrücke aus domänenspezifischen Dokumenten, in diesem Fall Unterrichtsmaterialien, indem es erstens alle Ausdrücke, welche bereits im domänenspezifischen oder domänenunabhängigen Lexikon enthalten sind, herausfiltert und zweitens die verbleibenden Ausdrücke als geordnete Liste abspeichert und dem Benutzer zur Auswahl vorlegt.

Der globale, d.h. auf dem Server gelegene XSE-Baum wird aufgrund der von den Clients gesammelten Eingabesätze (respektive deren Bedeutungsrepräsentationen) automatisch generiert, periodisch erneuert und an die Clients verteilt. Diese adaptive Technologie stellt folglich sicher, dass den semantischen

[73] Gemäss [Winiwarter et al. 97] sind XSE-Bäume erweiterte SE-Bäume. Der SE-Baum-Ansatz aus dem Bereich des *Machine Learning* generalisiert Entscheidungs-Bäume.

Komponenten der Clients neben dem lokal „gelernten“ Wissen auch das laufend aktualisierte, globale Wissen aller Clients zur Verfügung steht.

6.3.3 Beurteilung des DOA

Der DOA bedeutet vor allem in einer Hinsicht eine Abkehr vom typischen Front-End-Ansatz der 1980er- und frühen 1990er-Jahre: Zugunsten der Qualität des Sprachverstehens lässt man den Anspruch der DBMS-Portabilität fallen und setzt stattdessen (wieder), ganz in der Tradition früherer Prolog-NLIDB-Systeme, auf eine Architektur, welche das linguistische Front-End und die Datenbank integriert.

Die Kombination von objekt-orientierter und deduktiver Datenbanktechnologie erlaubt einerseits die Repräsentation komplex strukturierter Objekte und Objekthierarchien und andererseits deren Manipulation durch objekt-orientierte Methoden, welche in diesem Fall sogar logisches Inferieren, d.h. das Ableiten neuer Daten aus den bestehenden, ermöglichen (vgl. Kapitel **2.4.1**, **2.4.2**). Wie bereits erwähnt, ist dieser Ansatz auch für die Darstellung und Verarbeitung der komplexen Wissensstrukturen des Lexikons und des World Model geeignet. Und da mit denselben Methoden sowohl auf das Domänenwissen der Linguistik-Komponente(n) als auch auf die Anwendungsdaten in der Datenbank zugegriffen wird, fällt das Problem der Abbildung der Bedeutungsrepräsentation einer Anfrage auf die ausführbare Zielrepräsentation prinzipiell weg.
Der Prozess des Sprachverstehens profitiert von drei weiteren Systemeigenschaften:
(1) Es werden nur diejenigen linguistischen Verarbeitungsschritte abgearbeitet, die für eine vollständige Analyse des natürlichsprachlichen Ausdrucks notwendig sind. So wird z.B. die Syntaxanalyse nur bei Bedarf, d.h. nur zur gezielten Disambiguierung angewandt – was einen Effizienzvorteil gegenüber der gewohnten sequenziellen, an den linguistischen Beschreibungsebenen orientierten Verarbeitungsstrategie darstellt.
(3) Das System macht Gebrauch von verfügbaren linguistischen Ressourcen wie maschinenlesbaren Wörterbüchern, was einerseits den Aufwand für die Domänenkonfigurierung reduziert und andererseits das NLIDB-typische Problem der unbekannten Wörter bzw. der Datenbankwerte in der Anfrage entschärft.
(3) Das System setzt Techniken des maschinellen Lernens[74] für den automatischen Erwerb von linguistischem Wissen ein. So extrahiert es aus einem Textkorpus, den Unterrichtsmaterialien, domänenspezifische Lexikoneinträge und erweitert anhand der Eingaben des Benutzers das World Model durch Anwender- und Domänen-spezifisches Wissen. Dadurch wird der Aufwand für die Konfigurierung reduziert, die linguistische und konzeptuelle Abdeckung erhöht und eine stetige (automatische) Anpassung an die Bedürfnisse des Benutzers gewährleistet.

Die oft angestrebte DBMS-Portabilität, d.h. die Eigenschaft eines NLIDB, mit geringem Aufwand in eine andere Anwendungsumgebung versetzt werden zu können, wird beim DOA als problematisch erkannt: Die Portierung setzt in diesem Fall einen enormen Konfigurationsaufwand, meistens sogar Eingriffe in den Programmcode voraus – man denke z.B. an die Anpassung der (domänenunabhängigen) Transformationsregeln für die Generierung des anwendungsspezifischen Formalismus – und konfrontiert einen jeweils von Neuem mit dem Problem der inkonsistenten Repräsentationsmodelle.
Portabilität unter dem Aspekt der Domänen-Portierung ist hingegen wichtige Voraussetzung für den kosteneffektiven Einsatz des NLIDB. Die Massnahmen zur Vereinfachung der Domänenkonfigurierung sind oben unter Punkt (2) und (3) aufgeführt.

---

[74] Eine Übersicht über entsprechende „neuere“ Methoden der Computerlinguistik vermittelt z.B. [Charniak 93] oder [Manning and Schütze 1999].

6.4 Moderne kommerzielle NLIDBs

Die NLIDB-Systeme ENGLISH QUERY und EASYASK werden momentan als kommerzielle Produkte vertrieben. Den Schwerpunkt dieses Kapitels bilden die Erkenntnisse, welche ich aus der praktischen Arbeit mit ENGLISH QUERY gewonnen habe.

6.4.1 English Query

ENGLISH QUERY (EQ) von Microsoft[75] stellt eine Gesamtlösung zur Entwicklung und Konfiguration natürlichsprachlicher Retrievalschnittstellen zu Datenbanken des SQL-Datenbankservers von Microsoft, *SQL Server*, dar. Wie der Name besagt, unterstützt EQ (momentan) ausschliesslich Englisch.
Die folgende Untersuchung basiert auf der Version ENGLISH QUERY 2000, die zusammen mit dem SQL Server 2000 distribuiert wird.[76] EQ 2000 setzt Windows 95/98 oder Windows NT4/2000 als Betriebssystem voraus.
EQ besteht im Wesentlichen aus einer integrierten Entwicklungsumgebung, einer domänenunabhängigen Sprachverarbeitungs-Komponente genannt *Runtime Engine* sowie dem anwendungsspezifischen Domänenmodell. Eine EQ-Anwendung zu erstellen heisst, ein semantisches Modell der Anwendungsdomäne zu generieren und die *Runtime Engine* sowie das semantische Modell in den gewünschten Anwendungskontext einzubinden. In den folgenden Unterkapiteln soll relativ ausführlich auf das semantische Modell, auf die Realisierung einer EQ-Anwendung unter Verwendung der verschiedenen Entwicklungswerkzeuge sowie auf die Fähigkeiten einer EQ-Anwendung im Bereich des Sprachverstehens eingegangen werden. (Die zugrunde liegende Test-Datenbank ist in Kapitel **9.1** skizziert.)

6.4.1.1 Die Bestandteile von EQ

Die Entwicklungsumgebung von EQ, seit EQ 2000 in Microsofts *Visual Studio* 6.0 integriert, besteht aus dem Modell-Editor (*Model Editor*) und der Testumgebung (*Test Tool*). Während der Modell-Editor dem Erstellen eines semantischen Modells dient, ermöglicht die Testumgebung die Prüfung des aktuellen Modell-Status.
Die *Runtime Engine* ist als *COM*-Komponente implementiert. Sie nimmt eine natürlichsprachliche Anfrage in Englisch entgegen und berechnet unter Verwendung des (kompilierten) semantischen Modells der Anwendungsdomäne eine korrespondierende SQL-Anfrage. Sie stellt die zur Verarbeitung natürlicher Sprache benötigten domänenunabhängigen Informationen und Prozeduren zur Verfügung.

[75] Wo nichts anderes vermerkt ist, basieren die Informationen entweder auf [EQ 2000] oder auf eigenen Erfahrungen mit ENGLISH QUERY.
[76] EQ wird seit SQL Server Version 6.5 mitgeliefert. Vgl. [Snyder 00].

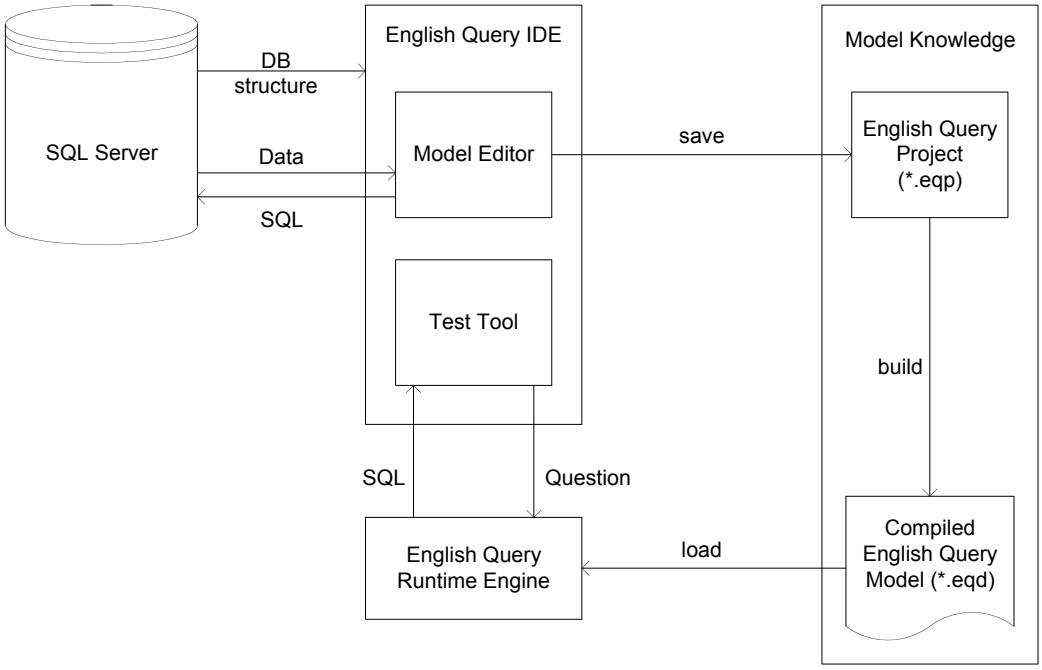


Abbildung 8 - Systemarchitektur von EQ im Verbund mit SQL-Server-DB

Das semantische Modell, das sog. *English Query Model* oder Domänenmodell, repräsentiert im Gegensatz zur *Runtime Engine* den domänenspezifischen, für jede EQ-Anwendung neu zu erstellenden und zu konfigurierenden Teil des NLIDB. Es enthält das Wissen der EQ-Anwendung über die Anwendungsdomäne:

- *Entitäten*. Der Begriff *Entität* bezeichnet ein Objekt der interessierenden Welt (wie z.B. eine Person, ein Ort, ein Ding oder eine Idee), d.h. ein Konzept, worauf man sich mit einem Nomen (bzw. einer Nominalphrase) bezieht. Die Entitäten sind analog zur Datenbankstruktur hierarchisch geordnet: denjenigen Entitäten, welche Relationen repräsentieren, sind diejenigen Entitäten untergeordnet, welche den Attributen der jeweiligen Relation entsprechen. Mit den „Attribut"-Entitäten sind die Werte der korrespondierenden Datenbank-Felder assoziiert.[77] Entitäten sind *semantische Objekte*.
- Beziehungen zwischen den Entitäten. Neben der eigentlichen Assoziation zwischen den an der Beziehung beteiligten Entitäten gehören dazu auch sog. *Phrasierungen* (*phrasings*), d.h. natürlichsprachliche Beschreibungen der jeweiligen Beziehung: z.B. „customers order products from employees". Eine Beziehung kann durch beliebig viele Phrasierungen charakterisiert werden. Durch Hinzufügen zusätzlicher Phrasierungen (z.B. „customers purchase products from employees") wird die konzeptuelle Abdeckung des Systems verbessert, ohne dass neue Beziehungen definiert werden müssten.
Beziehungen sind wie die Entitäten ebenfalls *semantische Objekte*.
- Domänenspezifische *Dictionary Entries* und Werte von Entitäten. Zusätzliche Wörterbuch-Einträge erweitern die Domäne konzeptuell und stellen ebenfalls *semantische Objekte* dar. Die Werte einer Attribut-Entität können aus der Datenbank ins Domänenmodell geladen werden (vgl. Kapitel **6.4.1.4.1**).

[77] Fragt man nach einer Attribut-Entität, so wird EQ zusätzlich die Werte derjenigen Attribute, die bei einer Anfrage nach der Relation geliefert würden, als Resultat zurückgeben. Die Attribut-Entitäten können allerdings als sog. *eigenständige Entitäten* (*standalone entities*) spezifiziert werden, was bewirkt, dass korrespondierende Datenbankwerte im Anfrage-Resultat einzeln, d.h. ohne die übrigen Daten der zugehörigen Relation, angezeigt werden.

- *Datenbankobjekte*: Relationen (tables), Attribute (fields) und virtuelle Relationen, d.h. Ergebnisse von Verbundoperationen (joins) im Falle einer SQL-Datenbank bzw. *Analysis Services*-Würfeln, -Dimensionen und –Ebenen im Falle einer OLAP-Client-Anwendung (vgl. Kapitel **2.5.1**).
- Informationen, welche die semantischen Objekte mit Datenbankobjekten assoziieren. So könnten diese Informationen z.B. die Entitäten *customer* und *product* auf die Relationen *customer* respektive *product* abbilden.
- *Projekt-Eigenschaften*, welche anwendungsspezifische Einstellungen speichern: z.B. Default-Entitäten, Angaben zur Fehlertoleranz bei der Analyse, Angaben zur Aktivierung des Benutzerkontexts, usw.

EQ speichert diese Informationen im XML-konformen *Semantic Modeling Format* (*SMF*) ab, das sowohl umfassenden Zugriff auf die Modellinformationen durch externe Programme gewährleisten als auch direktes Generieren und Manipulieren eines Modells ohne Umweg über die EQ-Entwicklungsumgebung ermöglichen soll.

6.4.1.2 Phrasierungen

Phrasierungen sind quasi-natürlichsprachliche, vom Benutzer zugewiesene Beschreibungen von Beziehungen. Einerseits enthalten sie die an der Beziehung beteiligten Entitäten, d.h. die „Argumente“ der Beziehung, und andererseits spezifizieren sie den Typ der Beziehung, d.h. deren „Funktor“.
EQ unterscheidet sieben Typen von Phrasierungen, wovon einige, so z.B. die Adjektiv-Phrasierungen, den expliziten Bezug zu (natürlichsprachlichen) Wortarten herstellen:
- *Eigennamen- oder ID-Phrasierung* (*name/ID phrasing*): „titles are the names of books“. Dieser Typ von Phrasierung beschreibt Beziehungen, wie sie zwischen einem semantischen Objekt und dessen Namen bzw. Bezeichnung bestehen.
- *Charakterzugs-Phrasierung* (*trait phrasing*): „patients have blood types“. Dieser Typ von Phrasierung beschreibt Beziehungen, wie sie zwischen einem Objekt und dessen Eigenschaften bestehen.
- *Adjektiv-Phrasierung* (*adjective phrasing*): „ages are how old people are“. Einer Charakterzugs-Phrasierung ähnlich, beschreibt dieser Typ von Phrasierung Eigenschaftsbeziehungen, die durch natürlichsprachliche Adjektive ausgedrückt werden.
- *Untermengen-Phrasierung* (*subset phrasing*): Dieser Typ von Phrasierung drückt anhand einer Hyperonym-Hyponym-Beziehung („some mountains are volcanoes“) die direkte Unterordnung einer Entität („volcanoe“) unter eine andere („mountain“) aus. Die entsprechenden Beziehungen bilden eine oder mehrere Entitäts- bzw. Typenhierarchien.
- *Präpositions-Phrasierung* (*preposition phrasing*): „stories are about topics“. Dieser Typ von Phrasierung beschreibt die Beziehung zwischen einer Nominalphrase und einer Präpositionalphrase.
- *Verb-Phrasierung* (*verb phrasing*): Dieser Typ von Phrasierung ist bei Beziehungen angebracht, welche sich unter Verwendung eines Verbs beschreiben lassen. Jeder entsprechenden Beziehung weist man damit eine Satzstruktur im Sinne eines Subkategorisierungsrahmens (z.B. „Subject Verb Object“) sowie Entitäten als Komplemente respektive Adjunkte (bei der optionalen Anbindung von Präpositionalphrasen, hier z.B. die Entität *employee*) zu. So bezeichnet z.B. die (Verb-)Phrasierung „customers order products from employees“ eine Beziehung, die zwischen den Entitäten *customer*, *product* und *employee* besteht, wobei eine Nominalphrase an der durch die Entität *customer* spezifizierten Position die Rolle des Subjekts bzw. Agens und eine weitere Nominalphrase an der durch die Entität *product* spezifizierten Position die Rolle des (direkten) Objekts bzw. Patiens innehat. Die Spezifizierung des Verbs sollte in der Aktiv-Form erfolgen: EQ nimmt intern automatisch eine Aktiv-Passiv-Konvertierung vor.

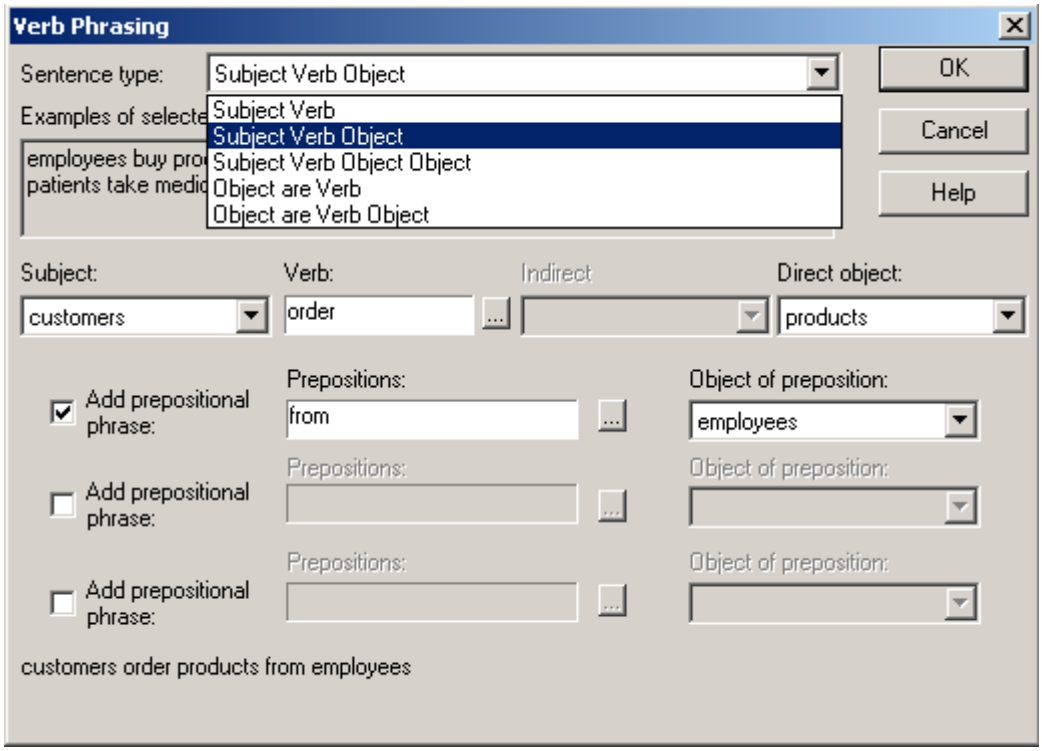


Abbildung 9 - Dialogfenster für die Spezifizierung einer Verb-Phrasierung

- *Befehls-Phrasierung* (*command phrasing*): „send customers products“. Ähnlich wie bei einer Verb-Phrasierung spezifiziert man Verb, Satzstruktur und beteiligte Entitäten. Allerdings wird eine Befehls-Beziehungen von EQ nicht als Aufforderung zur Anzeige einer Tabelle, sondern vielmehr als Befehl zur Ausführung einer Benutzer-definierten, anwendungsspezifischen Prozedur[78] interpretiert. Die obenstehende Phrasierung bewirkt zum Beispiel die Ausführung einer *sell*-Prozedur, welche als Parameter die Entitäten *product*, *customer* und *quantity of product* nimmt respektive (bei obligatorischen Parametern) verlangt. Befehls-Phrasierungen können auch Entitäten verwenden, die nicht mit einem Datenbankobjekt korrespondieren: In „Move <file> into the Recycle Bin“ könnte „the Recycle Bin“ als statische Entität der „Move“-Prozedur definiert sein und der Wert von „<file>“ aus der jeweiligen natürlichsprachlichen Anfrage des Benutzers ermittelt werden.

[78] Eine naheliegende Anwendungsmöglichkeit (und zugleich der Default im Fall einer Web-Anwendung) ist die Ausführung sog. *Stored Procedures* von SQL Server. Eine Stored Procedure fasst im Sinne eines prozeduralen Ausführungsplans, der u.a. auch Selektionen (IF-Anweisungen) erlaubt, eine Gruppe von Transact-SQL-Anweisungen zusammen. Sie ist auf dem Datenbankserver abgelegt und kann dort von Client-Anwendungen aufgerufen werden, um z.B. Berichte zu erstellen, Daten zu visualisieren oder generell vorgefertigte, komplexere Funktionen auszuführen. Stored Procedures verarbeiten Benutzer-spezifizierte Eingabe-Parameter und können mehrere Werte in Form von Ausgabe-Parametern zurückgeben. So könnte man z.B. eine Update- oder Insert-Prozedur implementieren, welche die Beschränkung auf Datenwiedergewinnung aufhebt.
Daneben können prinzipiell beliebige Prozeduren und Funktionen aufgerufen werden. Die Unterstützung der entsprechenden Module muss allerdings von Hand programmiert werden, was im Falle der installierten Web-Anwendung in der Datei „commands.asp“ (im Root-Verzeichnis der Anwendung) vorzunehmen ist.

6.4.1.3 Realisierung einer EQ-Anwendung

Bei der Realisierung einer EQ-Anwendung kann man (grob) zwei Phasen unterscheiden: Während der ersten Phase erstellt und konfiguriert ein Wissensingenieur das domänenspezifische EQ-Modell; im Verlauf der zweiten Phase werden das EQ-Modell, die EQ-Runtime-Engine sowie weitere benötigte Komponenten als EQ-Anwendung in die bestehende IT-Umgebung eingebunden.

EQ unterstützt verschiedene Methoden zur Erstellung und Erweiterung eines EQ-Modells. In den meisten Fällen wird man mit Hilfe der Projekt-Erstellungs-Assistenten ein semantisches Basis-Modell generieren lassen und dieses in der Folge anpassen und verfeinern. Neben dem „manuellen" Hinzufügen von Entitäten und Beziehungen per Menü-Befehlen erlaubt der Modell-Editor komfortables graphisches „Authoring". Die Methode des „Authoring by example" verwendet den sog. *Suggestion Wizard*, der, aus der Testumgebung aufgerufen, Vorschläge zur Definition von Entitäten und Beziehungen macht. Die „programmatische" Methode, welche direkt mit den Modelldaten in „Rohform", d.h. im XML-konformen *Semantic Modeling Format* (*SMF*) arbeitet, empfiehlt sich schliesslich dann, wenn man die Entwicklungsumgebung von EQ umgehen möchte.

Auch die Phase der Systemeinbindung lässt sich unter Umständen enorm verkürzen. Für die Erstellung einer kompletten Web-Anwendung bietet EQ das „One-Click Web Site Deployment feature" an, das im Zusammenspiel mit dem (leider nicht mitgelieferten) ASP-Editor *Interdev*, Bestandteil der *Microsoft Visual Studio*-Entwicklungsumgebung, die automatische Generierung und Installation einer ASP-basierten EQ-Anwendung „auf Knopfdruck" ermöglicht.

Im Übrigen können EQ-Komponenten und –Anwendungen von einem Programmierer in beliebige *Visual Basic*- oder *Visual C++*-Projekte eingebunden werden.

6.4.1.3.1 Modell-Editor

Der graphische Modell-Editor erlaubt das Erstellen und Erweitern eines semantischen EQ-Modells. Der Benutzer kann sich per Drag-and-Drop Beziehungen zwischen Entitäten anzeigen lassen; Einstellungen zu Entitäten, Beziehungen und Wörterbucheinträgen sind nach einem Doppelklick auf GUI-Elemente einsehbar und editierbar.

Die *Semantics*-Ansicht des Modell-Editors zeigt die semantischen Objekte des EQ-Modells an. Im linken Fensterabschnitt erscheinen die bereits definierten Entitäten, Beziehungen (respektive deren Phrasierungen) und Wörterbuch-Einträge jeweils alphabetisch geordnet in entsprechenden Ordnern; im rechten Fensterabschnitt kann der Benutzer die vom linken Fensterabschnitt per Drag-and-Drop geholten Entitäten und Beziehungen frei arrangieren, wobei EQ automatisch die korrespondierenden Beziehungen respektive Entitäten ergänzt.

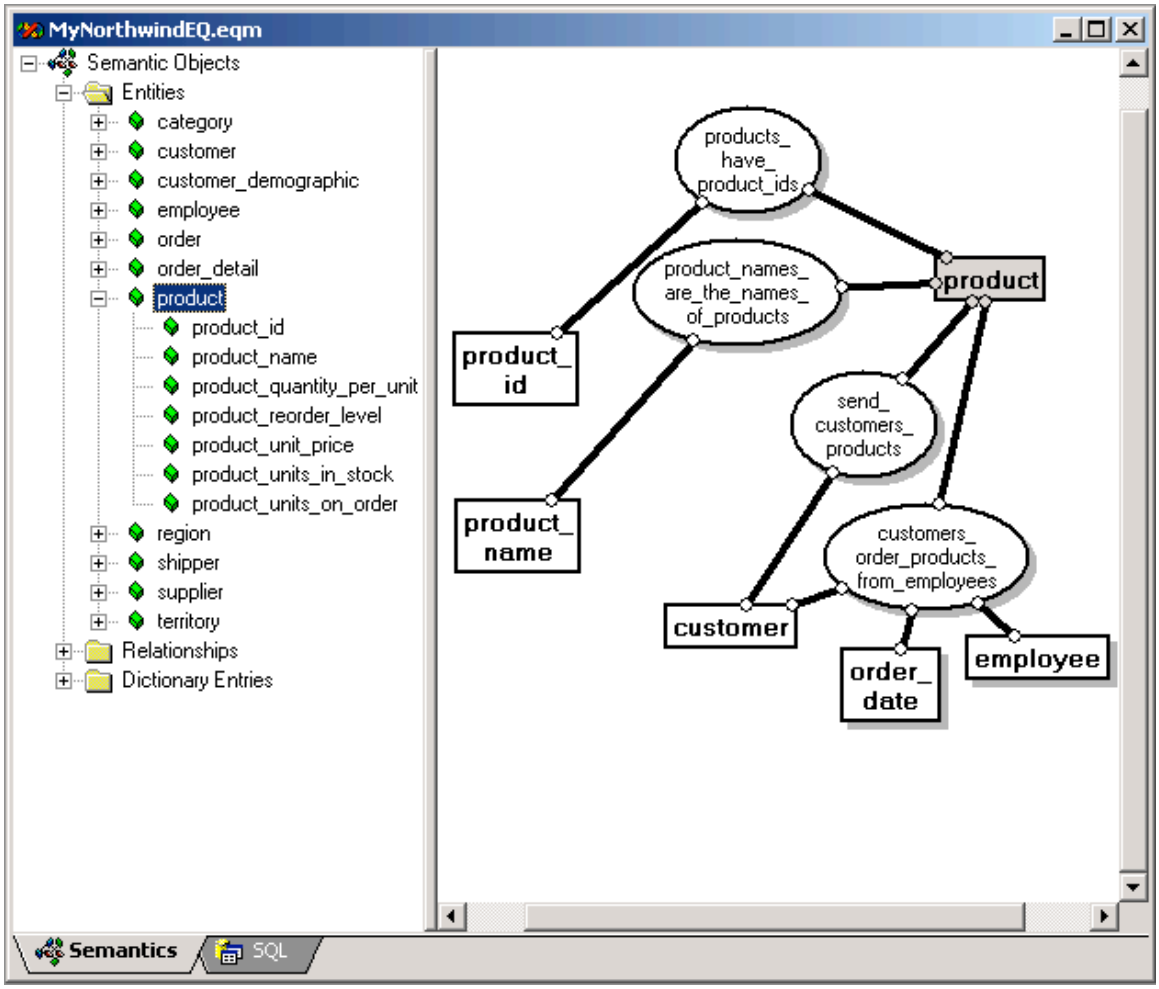


Abbildung 10 - Modell-Editor: Semantics-Ansicht

Im Gegensatz zur *Semantics*-Ansicht zeigt die *SQL*-Ansicht des Modell-Editors die Datenbankobjekte eines EQ-Modells. Relationen (*Tables*) enthalten einerseits die sog. „Felder“ (*Fields*), d.h. Attribute, und andererseits die *Joins*, welche Abhängigkeiten zwischen Relationen spezifizieren. In dieser Ansicht lassen sich (u.a. per Kontextmenü) neue Entitäten mit Datenbankobjekten assoziieren oder bestehende Entitäten, welche mit dem jeweiligen Datenbankobjekt korrespondieren, anzeigen.

6.4.1.3.2 Assistenten zur Erstellung eines Modells

EQ bietet mit den *Project Wizards* Funktionen an, welche das Erstellen eines EQ-Modells wesentlich erleichtern. Diese Assistenten, vor der manuellen Arbeit im Modell-Editor aufgerufen, generieren auf Basis der Datenbankstruktur und unter Berücksichtigung einiger durch den Benutzer erfolgter Einstellungen automatisch ein semantisches Basismodell, d.h. mit Tabellen und Attributen assoziierte Entitäten sowie grundlegende Beziehungen wie Eigenschafts-/Attributsbeziehungen und Namens-/ID-Beziehungen. Entsprechend beschränkt sich die Arbeit des Wissensingenieurs darauf, das Modell zu verfeinern, d.h. durch Assoziierung der Entitäten mit synonymen Bezeichnungen und durch Hinzufügen von Beziehungen, Phrasierungen und Wörterbuch-Einträgen sowohl die linguistische als auch die konzeptuelle Abdeckung des Systems zu erhöhen.

Der *SQL Project Wizard* definiert automatisch Entitäten und Beziehungen basierend auf einer relationalen Datenbank, wobei er auch Joins berücksichtigt. Der optimierte *OLAP Project Wizard* unterstützt sowohl relationale als auch OLAP-Datenbanken.

6.4.1.3.3 Testumgebung

Die Testumgebung bildet neben dem Modell-Editor den zweiten Hauptbestandteil von EQ. Deren wichtigstes Werkzeug, die Modell-Test-Konsole (*Model Test*), besteht aus den beiden Ansichten *Answer* und *Analysis*. Die Antwort-Ansicht zeigt zusätzlich zum Resultat der Anfrage in Tabellenform eine eindeutige Paraphrase der Anfrage, wie sie von EQ interpretiert wird, eine natürlichsprachliche Umschreibung der gelieferten Entitäten sowie die *Transact-SQL*-Form der Anfrage an. Falls EQ die Anfrage nicht „versteht", wird dem Benutzer eine Fehlermeldung präsentiert. Im Falle von Unklarheiten und Mehrdeutigkeiten fordert EQ den Benutzer mit einem Klärungs-Dialogfenster auf, seine Anfrage zu präzisieren.

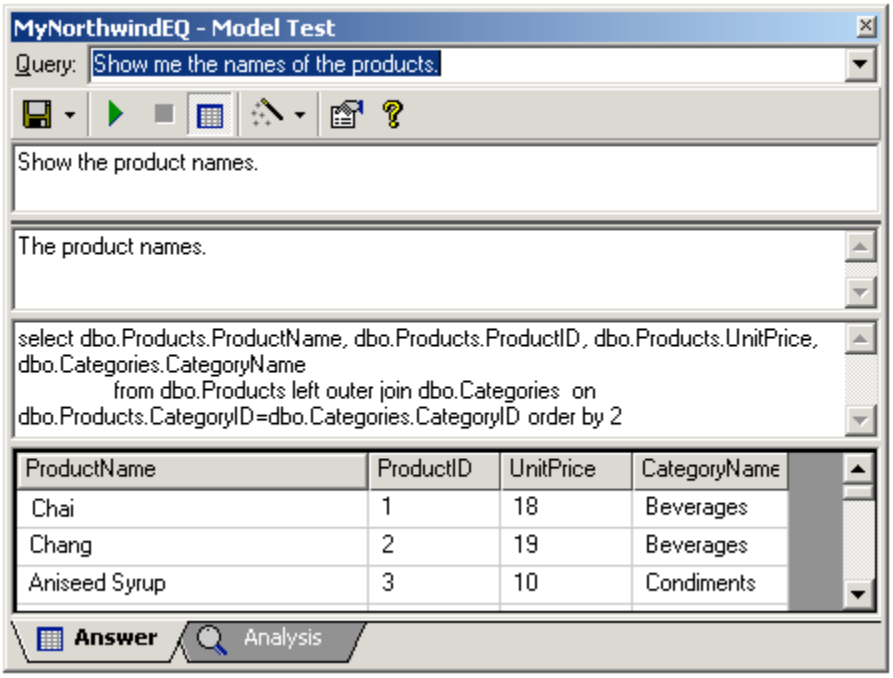


Abbildung 11 - Testumgebung: Antwort-Ansicht des Model-Test-Werkzeugs

Die Analyse-Ansicht zeigt neben der Anfrage-Paraphrase die „verwendeten" Entitäten und Phrasierungen an, d.h. diejenigen semantischen Objekte, welche die Bedeutungsrepräsentation der natürlichsprachlichen Anfrage konstituieren.

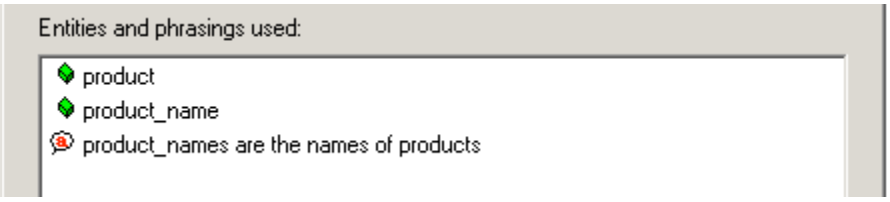


Abbildung 12 - Testumgebung: Analyse-Ansicht des Model-Test-Werkzeugs

Der *Suggestion Wizard*, aus dem Model-Test-Werkzeug aufgerufen, unterstützt den Wissensingenieur beim Erstellen und Verfeinern des EQ-Modells, indem er ihm basierend auf der aktuellen Anfrage (modifizierbare) Vorschläge zur Erweiterung des bestehenden Modells unterbreitet. Die mehr oder weniger vollständigen Vorschläge repräsentieren entweder Vorlagen zur Spezifizierung von Entitäten oder Vorlagen zur Spezifizierung von Phrasierungen, wobei die (unvollständigen) Phrasierungen gemäss den Entitätsdefinitionen komplettiert werden. Aufgrund der Auswahl des Benutzers generiert EQ automatisch die für die Repräsentation der Anfrage benötigten Beziehungen.

Einen weiteren Bestandteil der Testumgebung bilden die Werkzeuge für sog. *Regressions-Tests*. Deren Nutzen besteht darin, dass durch die automatische Analyse und Evaluierung einer Sammlung von anwendungsspezifischen Test-Anfragen gerade bei umfangreichen semantischen Modellen der Test-Prozess enorm beschleunigt wird. Anhand einer Auswertung und Gegenüberstellung der Anfrage-Resultate alter und neuer Testdurchläufe kann nach einer Änderung des Modells effizient geprüft werden, ob die relevanten Testfragen auch mit dem angepassten Modell verarbeitet werden können und ob das Modell (trotz der Änderungen) konsistent ist.
Mit dem entsprechenden Editor lassen sich die XML-konformen *Regressions-Dateien* einsehen, die Sammlungen relevanter Test-Anfragen sowie deren Resultate, Paraphrasen und SQL-Formulierungen beinhalten. Ein Klick auf den Save-Button der Modell-Test-Konsole speichert den zu einer Test-Anfrage gehörigen Datensatz in der aktuellen Regressions-Datei, und aus dem Projektverwaltungs-Fenster werden per Kontextmenü die Regressions-Test-Funktionen aufgerufen.

6.4.1.3.4 Anwendung für den Endbenutzer

Stellvertretend für die möglichen Anwendungskontexte von EQ-Anwendungen (vgl. Kapitel **6.4.1.3**) soll hier das *Web-Szenario* beschrieben werden. Die Schnittstelle für den Endbenutzer ist in diesem Fall durch eine Anzahl von *Active Server Pages* (*ASP*) realisiert, welche einerseits für die Darstellung der Benutzeroberfläche und andererseits für die Kommunikation zwischen Benutzeroberfläche, EQ-Runtime-Engine und Datenbank-Server zuständig sind. Der Benutzer greift über einen *ActiveX*-fähigen Web-Client auf die EQ-Anwendung zu.

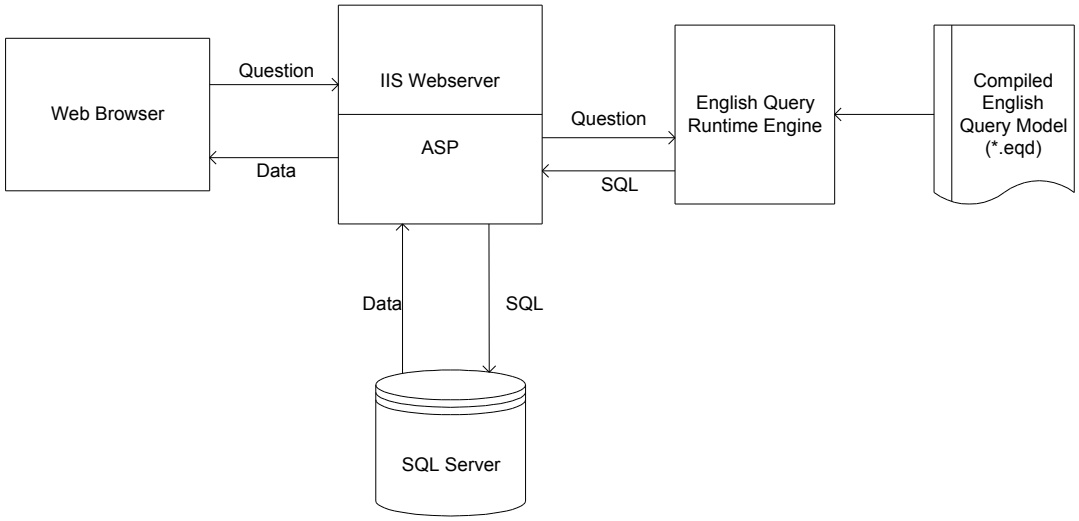


Abbildung 13 - Einsatzszenario: Web-basierter, natürlichsprachlicher Zugriff auf SQL Server

Den *Default-Modus* der Benutzeroberfläche konstituieren zwei Frames, wobei der linke Frame das Eingabe-Formular und der rechte Frame die Ausgabe-Seite beinhaltet. Bereits dieser Modus stellt einen Grossteil der Funktionalität bzw. der Informationen des Modell-Test-Werkzeugs zur Verfügung; spezielle Erwähnung verdienen die Link-Liste mit Beispielanfragen, die einen Anhaltspunkt bezüglich der linguistischen und der konzeptuellen Abdeckung des Systems liefern, die „Store As Sample Question"-Funktion, welche die aktuelle Anfrage den Beispielanfragen hinzufügt,[79] sowie die „Report As Problem Question"-Funktion, welche die aktuelle Anfrage im Sinne eines Feedbacks an den für die Modell-Wartung zuständigen Wissensingenieur in eine Datei mit „Problemfällen" schreibt.

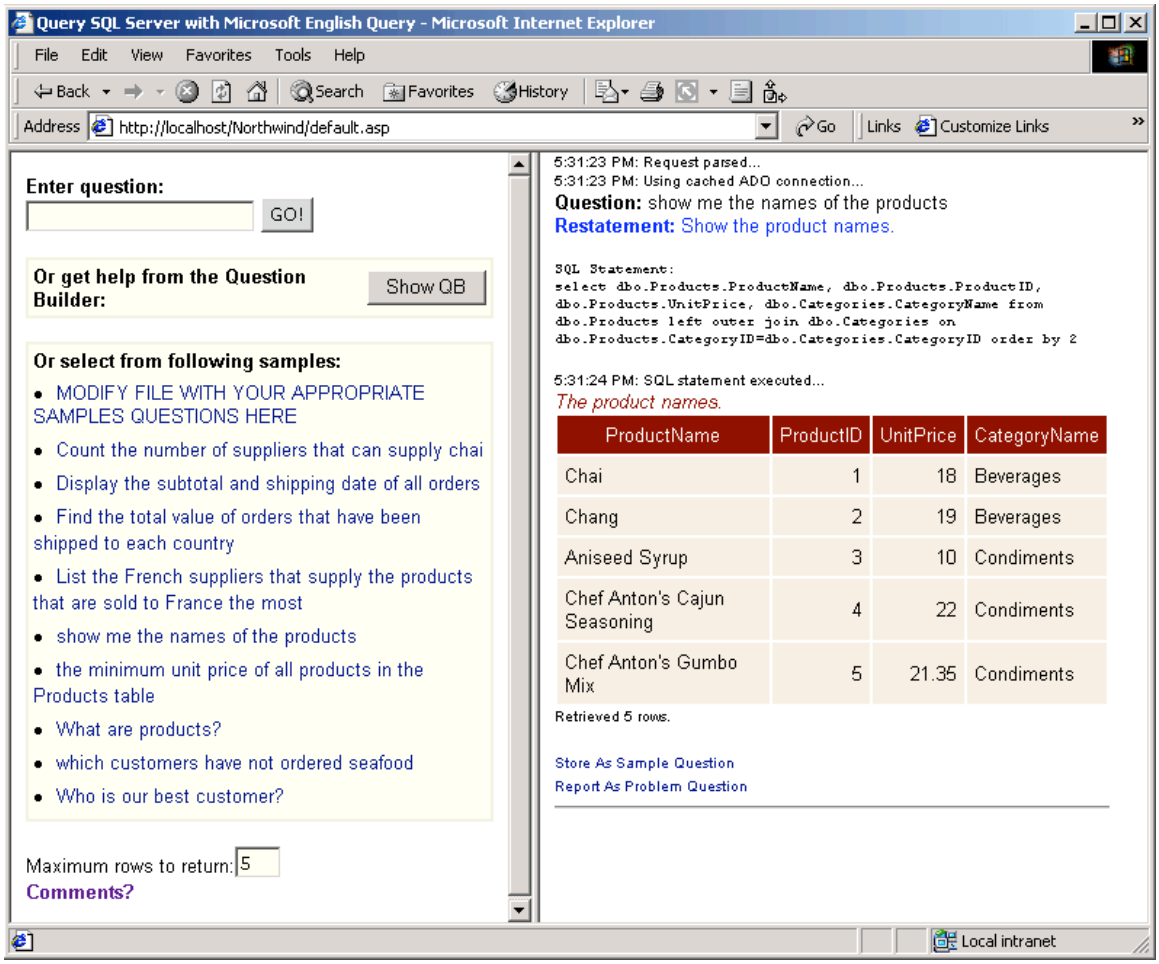


Abbildung 14 - ASP-basierte EQ-Web-Anwendung: Default-Modus

Durch Anklicken des „Show QB"-Buttons schaltet man vom Default-Modus in den *Question-Builder-Modus* um. Dessen zentrales Bedienelement stellt der graphische *Question Builder* dar, eine *ActiveX*

[79] Die Beispielanfragen werden (abgesehen von Zeilenumbrüchen) unformatiert in einer Textdatei gespeichert und sind dort frei editierbar. Leider bietet EQ (meines Wissens) keine automatische Konvertierung der Frage-Elemente einer Regressions-Datei in „Plain-Text"-Anfragezeilen einer Beispielanfragen-Datei an – die Beispielanfragen-Liste muss in jedem Fall separat gepflegt werden und stellt insbesondere bei einer Domänen-Portierung einen Zusatzaufwand dar, der höchstens dadurch gemindert werden kann, dass die (anfangs kurze) Liste während der Verwendung des NLIDB sukzessive erweitert, d.h. um erfolgreiche Anfragen ergänzt wird.

Control-Komponente, die analog zum EQ-Modell-Editor Entitäten und Beziehungen der Domäne anzeigt und das interaktive Generieren von Beispielanfragen per Drag-and-Drop im Web-Client ermöglicht.
Die Question-Builder-gestützte Formulierung einer Anfrage funktioniert folgendermassen:
(1) Der Benutzer zieht eines oder mehrere semantische Objekte von der linken Fläche in die mittlere Fläche.
(2) Der Question-Builder ergänzt die graphische Visualisierung des Modells (mittlere Fläche) um korrespondierende Beziehungen und Entitäten.
(3) Der Question-Builder generiert kontextabhängige, d.h. auf die aktiven Objekte der Modell-Visualisierung abgestimmten Beispielanfragen und zeigt sie in der rechten Fläche an.
(4) Der Benutzer modifiziert die Visualisierung, bis eine seinen Retrieval-Bedürfnissen entsprechende Beispielanfrage in der rechten Fläche erscheint:

- Der Benutzer wählt aus dem Kontextmenü eine passende Darstellungs-Option oder passt den Zoom-Faktor an.
- Der Benutzer löscht semantische Objekte oder fügt weitere hinzu.
- Der Benutzer versieht einzelne Entitäten mit Werten, die sich auf Datenbankwerte beziehen bzw. Bedingungen bezüglich der Datenbankwerte/Tupelauswahl (WHERE-Klausel) spezifizieren.

(5) Entweder führt der Benutzer direkt die passende Beispielanfrage aus (durch Klick auf den „?“-Button der betreffenden Beispielanfrage) oder er fügt die Beispielanfrage ins Anfrage-Formularfeld ein (durch Klick auf die Beispielanfrage), um Resultat-Parameter wie die Anzahl zurückzuliefernder Reihen einzustellen oder um die Beispielanfrage weiter zu modifizieren.
(6) EQ zeigt das Resultat (mit denselben Zusatzinformationen wie im Default-Modus) gemäss Benutzervorgaben im unteren Frame an.

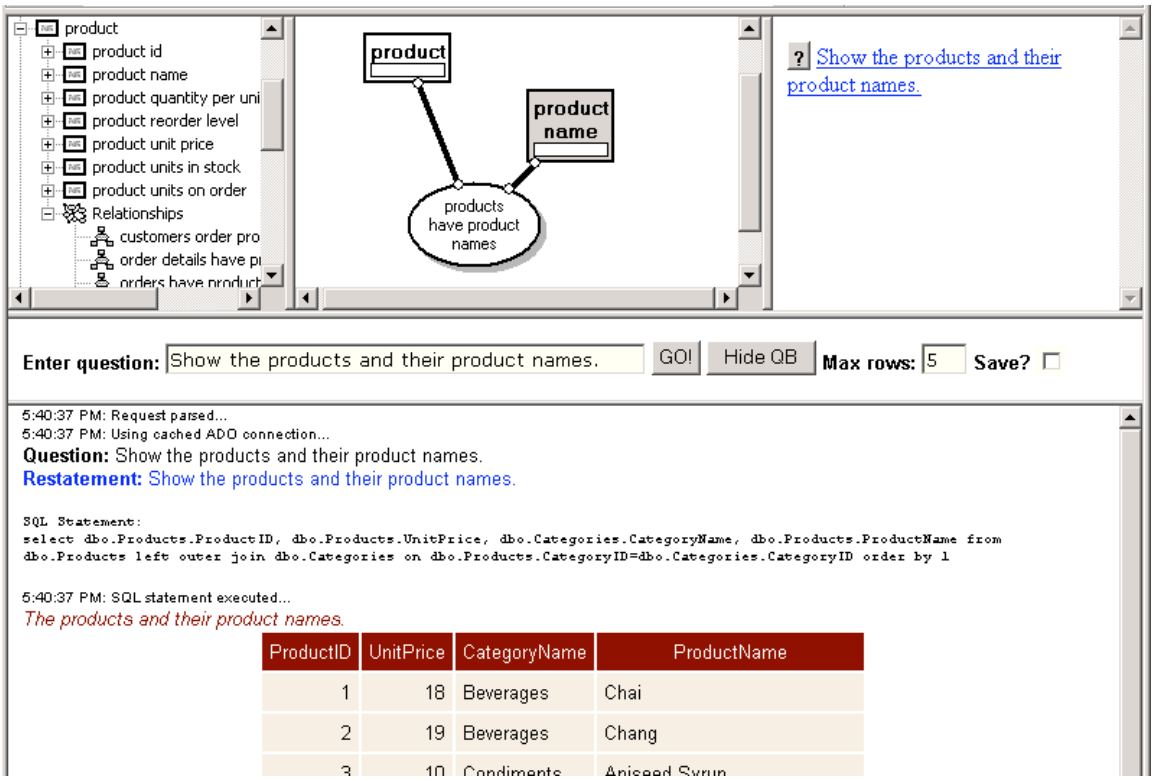


Abbildung 15 - ASP-basierte EQ-Web-Anwendung: Question-Builder-Modus

6.4.1.4 Natürlichsprachliche Analyse und Generierung

In diesem Kapitel geht es darum, EQ „etwas auf den Zahn zu fühlen“. Anhand einiger fingierter Test-Anfragen, welche aufgrund von Ambiguitäten, Unvollständigkeiten oder Inkorrektheiten typische Problemkandidaten darstellen, sollen die Fähigkeiten von EQ im Bereich der Analyse einer natürlichsprachlichen Benutzer-Eingabe und der Generierung einer passenden System-Antwort dokumentiert und illustriert werden.

Aus Effizienzgründen stammen die Resultate der folgenden Untersuchung aus Anfrage-Sitzungen an der Modell-Test-Konsole von EQ. Dessen Verhalten hinsichtlich Antwortpräsentation und Feedback-Mechanismen ist jedoch weitgehend identisch mit demjenigen einer Endbenutzer-Anwendung – insofern lassen sich die Ergebnisse direkt auf ein solches Szenario übertragen.

(Die zugrunde liegende Test-Datenbank sowie weitere Anfrage-Beispiele sind in Kapitel **9.1** respektive **9.2** beschrieben.)

6.4.1.4.1 Datenbankwerte in der Anfrage

EQ erkennt Namen bzw. Datenbankwerte in der Anfrage prinzipiell an ihrer äusseren Form, so z.B. die häufigen Eigennamen an der Grossschreibung, und versucht, sie eindeutig einer Entität zuzuweisen. Beinhaltet das EQ-Modell z.B. eine Beziehung mit der Phrasierung „customers order products“ und stellt der Benutzer die Anfrage „What products does Alfreds Futterkiste order?“, so erkennt EQ anhand der Grossschreibung, dass es sich bei „Alfreds Futterkiste“ um einen (Eigen-)Namen handelt, und interpretiert den Ausdruck gemäss der verwendeten Phrasierung als Wert zur Entität *customer*.

Um die Anzahl Fälle, in welchen der als Name erkannte Ausdruck nicht (eindeutig) einer Entität zugewiesen werden kann, zu reduzieren, kann in den Projekt-Einstellungen unter „English < Assume unknown names refer to:“ für jede EQ-Anwendung festgelegt werden, dass sich unbekannte Namen auf eine (zu spezifizierende) Default-Entität beziehen. Auch lassen sich die Werte aller Entitäten, die mit Attributen bzw. Feldern assoziiert sind und denen ein „Name type“[80] zugewiesen worden ist, explizit dem Modell hinzufügen („Add values of entity to model“, im Entitäten-Eigenschaftsfenster), was deren spätere Erkennung auch ohne semantischen Kontext garantiert.

Falls der Ausdruck zwar (typographisch) als Name erkannt, jedoch weder durch den lexikalischen Look-Up noch durch die semantische Kontextanalyse bestimmt werden kann und falls auch kein Anwendungs-weiter Default gesetzt ist, fordert EQ den Benutzer mit einem Klärungs-Dialogfenster auf, die passende Entität zu spezifizieren:

[80] Den „Name type“ spezifiziert man im Entitäten-Eigenschaftsfenster; zur Auswahl stehen neben „Proper name (John Smith)“ auch die Optionen „Common name (wrench)“, „Classifier name (the Finance department)“, „Model name (the Mustang)“ und „Unique ID (employee 1263)“, wobei in Klammern die jeweiligen Beispiele stehen. Unklar ist, ob mit der Zuweisung eines Name-Types entsprechend angepasste Pattern-Matcher ins Spiel kommen. Vgl. auch [EQ 2000: Entity/New Entity Dialog Box].

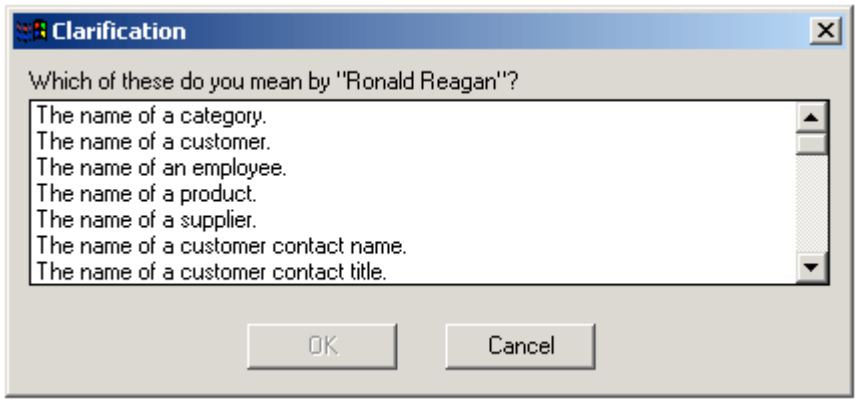


Abbildung 16 - Klärungs-Dialogfenster zur semantischen Bestimmung eines Namens

Im Hinblick auf die weitere Verarbeitung von Namen/Datenbankwerten wurde Folgendes festgestellt:
- EQ bietet keinerlei Mechanismen zur Rechtschreibkorrektur von Datenbankwerten.
- Enthält die Anfrage einen zwar hinsichtlich der Entität bestimmten, jedoch nicht in der Datenbank enthaltenen Datenbankwert, so gibt EQ keine kooperative Fehlermeldung (bezüglich der falschen Existenzpräsupposition wegen fehlendem Denotat), sondern nur den NULL-Wert aus.

6.4.1.4.2 PP-Anbindung

Die folgende Beispielanfrage (transkribiert) soll aufzeigen, wie EQ mit syntaktischer Ambiguität beim Modifier-Attachment (z.B. PPs oder Relativsätze) umgeht.
In diesem Fall ist die syntaktische Ambiguität dadurch gegeben, dass sich die PP „with id 1“ syntaktisch sowohl auf „products“ als auch auf „suppliers“ beziehen könnte, zumal „id“ (absichtlich) lexikalisch ambig als Synonym sowohl für „product id“ als auch für „supplier id“ definiert wurde. Es handelt sich also um einen Fall, der „truly ambiguous“ ist.

```
> Show me the products from all suppliers with id 1.

[Paraphrase] Which products does every supplier 1 sell?
[NL-Resultat] The products every supplier 1 sells are:
[...]
```

Eine erste Erkenntnis besteht darin, dass EQ syntaktische Ambiguität nicht anzeigt – weder in Form von Fehlermeldungen noch in Form einer Auswahl von Lösungen. Allerdings kann EQ die Anfrage verarbeiten, was zumindest die Vermutung zulässt, dass intern eine Disambiguierung stattgefunden hat, z.B. unter Verwendung von Heuristiken wie das „most right association principle“, welches die hier tatsächlich verwendete Struktur bevorzugen würde.
Beseitigt man die lexikalische Ambiguität, indem man „id“ nurmehr als Synonym von „product id“ zulässt, so bezieht EQ die PP „with id 1“ korrekterweise auf „products“. Eine mögliche Erklärung für das Systemverhalten besteht darin, dass die (noch immer) syntaktisch ambige Konstruktion aufgrund semantischer Verfahren, die auf Typenhierarchie- und Selektionsrestriktionen-ähnliche Strukturen des EQ-Modells zugreifen, disambiguiert wird – aufgrund des Fehlens einer Spezifikation der Sprachverarbeitungsprozesse lässt sich dies allerdings kaum verifizieren.

6.4.1.4.3 Skopusambiguitäten

Ähnlich wie im Fall der syntaktisch ambigen Anfragen veranlassen auch Anfragen mit Skopusambiguitäten EQ weder zur Generierung einer Fehlermeldung noch zur Anzeige einer Auswahl von Interpretationen. Bei Anfragen der Art „does every supplier sell a product?“ wählt EQ automatisch die distributive, häufigere Lesart mit weitem Skopus des Allquantors. Falls die zweite Lesart (mit weitem Skopus des Existenzquantors) die intendierte ist, muss dies in der Formulierung der Anfrage explizit ausgedrückt werden: „is there a product every supplier sells?“.

6.4.1.4.4 Konjunktion und Disjunktion

Bei der Übersetzung der natürlichsprachlichen Konjunktion „und/and“ in eine semantische (logische) Zwischenrepräsentation stellt sich das Problem, dass erstere prinzipiell sowohl konjunktiv als auch disjunktiv interpretiert werden kann. Eine einfache Abbildung der natürlichsprachlichen Konjunktion auf die logische Konjunktion ist gerade im Bereich der NLIDB nicht adäquat, da, wie im folgenden Beispiel, häufiger nach Vereinigungsmengen als nach Schnittmengen von Resultaten gesucht wird und entsprechend die disjunktive Lesart die intendierte ist (Beispiel transkribiert):

```
> Show the products supplied by Exotic Liquids and Tokyo Traders.

[Paraphrase] Which products are supplied by Exotic Liquids or are
supplied by Tokyo Traders?
[NL-Resultat] The products that are supplied by Exotic Liquids or are
supplied by Tokyo Traders are:
[...]
```

Sinnvollerweise interpretiert EQ hier „and“ als Disjunktion. Möchte man hingegen auf die Resultat-Schnittmenge zugreifen, so ist man u.U. mit der „stärkeren“ natürlichsprachlichen „sowohl-als-auch“-Formulierung erfolgreich: Die mit der neuen Anfrage „show the products supplied both by Exotic Liquids and Tokyo Traders“ erzwungene konjunktionale Lesart generiert die Fehlermeldung, dass „[a]ccording to the structure of the database, each product is supplied by a single supplier“, d.h., dass gemäss dem Datenbank-Schema ein Produkt gar nicht mehrere „suppliers“ haben kann. Dieser Sachverhalt legt die Vermutung nahe, dass sich EQ unter Berücksichtigung semantischer Einschränkungen, die durch das Datenbank-Schema bzw. durch Konsistenzregeln der Datenbank auferlegt sind, für die wahrscheinlichere, d.h. unproblematische Lesart entscheidet.
Das folgende Beispiel (transkribiert) widerspricht dieser These zumindest nicht:

```
> Show the products ordered from Davolio and Fuller.

[Paraphrase] Which products that are ordered from Fuller are ordered
from Davolio?
[NL-Resultat] The products that are ordered from Davolio and are
ordered from Fuller are:
[...]
```

Da gemäss dem Domänenmodell/Datenbank-Schema Produkte bei verschiedenen Angestellten (namens „Davolio“ oder „Fuller“) bestellt werden können, besteht für EQ kein Anlass, nicht die konjunktive Lesart zu bevorzugen. EQ wählt folglich dann die disjunktive Lesart, wenn (1) dies explizit („from Davolio or Fuller“) gewünscht wird oder wenn (2) die konjunktive Lesart von „and“ zu Verletzungen der Konsistenzregeln führt – was, wie eingangs Kapitel bereits erwähnt, leider nicht unbedingt der Anfrage-„Realität“ gerecht wird.

Konstruktionen wie Subjekt-Koordination im Hauptsatz („What products do Exotic Liquids or Tokyo Traders sell?“) oder Subjekt-Koordination im Relativsatz („Who sells products that customer ALFKI and customer ANTON order?“), welche von LOQUI [LOQUI 91: 18-21] nicht verarbeitet werden können, stellen für EQ hinsichtlich ihrer syntaktischen Struktur kein Problem dar. Im Falle der Koordination zweier oder mehrerer Hauptsätze liefert EQ zu jeder Anfrage separat die korrespondierende Resultatmenge zurück:

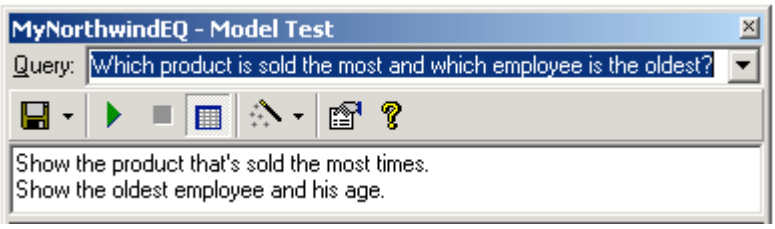


Abbildung 17 - Koordination von zwei Hauptsatz-Anfragen

6.4.1.4.5 Anaphorische Referenzen und Ellipsen

Für die Behandlung sowohl von anaphorischen Referenzen als auch von Ellipsen gilt, dass mittels der Option „Enable user context“ der Projekt-Einstellungen die Berücksichtigung des (sprachlichen) satzübergreifenden Diskurskontexts bei der Verarbeitung der aktuellen Anfrage (de-)aktiviert werden kann.
Die Paraphrase mit aufgelösten anaphorischen Referenzen und Ellipsen ermöglicht die Kontrolle über den Diskursanalyseprozess und bildet im Falle einer Fehlinterpretation Ausgangslage für die Re-Formulierung.
Folgefragen mit anaphorischen Verweisen und elliptischen Konstruktionen werden verwendet, um rasch und innerhalb der bestehenden linguistischen Schranken auch komplexe(re) Anfragen stellen zu können (Beispiel transkribiert):

```
> Show the employee named Davolio.
[...]
> Her title of courtesy.
[...]
> And the home phone.
[Paraphrase] Show the employees named Davolio, their employee titles
of courtesy and their employee home phones.
```

Grundsätzlich gilt, dass EQ bei elliptischen Folgefragen, die mit der Konjunktion „and“ eingeleitet werden, die gewünschten Kolonnen der (von der vorherigen Anfrage) bereits bestehenden Resultat-Tabelle hinzufügt. Hätte die Anfrage nur „the/her home phone“ gelautet, so wäre die Kolonne *TitleOfCourtesy* in der Resultat-Tabelle durch die Kolonne *HomePhone* ersetzt worden.
Bei der Auflösung anaphorische Referenzen auf Nominalphrasen berücksichtigt EQ zwar auf nicht-intuitive Art den Numerus, mangels entsprechender semantischer Informationen zu lexikalischen Einheiten jedoch nicht das Genus. Obwohl rein linguistisch (und intuitiv) der Plural-Verweis in „what are their ages?“ nicht mit dem Ergebnis von „show me the employee named Davolio“, nämlich dem einen Tupel zu „Frau Davolio“, kongruent ist, wird die Anfrage verstanden und sinnvoll interpretiert. Dies hängt damit zusammen, dass das Ergebnis der Anfrage mengenwertig ist, d.h. die Menge aller Tupel, deren Attribut-Wert zu *EmployeeName* „Davolio“ ist, repräsentiert - entsprechend bezieht EQ „their ages“ auf eben diese Menge, die zufälligerweise nur ein Element besitzt. Problematisch ist der umgekehrte Fall: Selektiert man zuerst eine Menge von mindestens zwei Tupeln („show the employees“) und verweist anschliessend mittels einer Singular-Referenz darauf („what’s [his/her/its] home phone?“), so generiert EQ eine Fehlermeldung („Sorry, I didn’t understand that.“).

Anaphorische Verweise lassen sich dazu nutzen, alle Werte einer Entität im Fokus ausser den bereits gefundenen anzuzeigen (Beispiel transkribiert):

```
> Show the employee called Davolio and his age.
[...]
> How old are the others?

[Paraphrase] Show the employees besides Davolio and their age.
[NL-Resultat] The employees besides Davolio and their age.
[...]
```

EQ löst auch Verweise auf ganze Phrasen auf (Beispiel transkribiert):

```
> How do you know it?

[Paraphrase] How do I know the employees besides Davolio and their
age?
[Antwort] The information was in the database.
```

6.4.1.4.6 Präsuppositionen

Die falsche Existenzpräsupposition im folgenden (transkribierten) Beispiel, dass es (in der Anwendungsdomäne) überhaupt einen Kunden gibt, welcher mehr als hundert Produkte bestellt hat, wird von EQ entweder nicht erkannt oder zumindest nicht dem Benutzer mitgeteilt:

```
> Does every customer that orders more than 100 products have a fax?

[Paraphrase] Does every customer that orders more than 100 products
have a customer fax?
[NL-Resultat] Does every customer that orders more than 100 products
have a customer fax?
[Resultat in Tabellenform] False
```

EQ gibt das NULL-Resultat direkt an den Benutzer weiter. Generell scheint die (adäquate) Behandlung von falschen Benutzerannahmen nicht zu den Stärken von EQ zu gehören – was nicht weiter verwundert, da die entsprechenden Mechanismen gerade im Zusammenspiel mit einer SQL-Datenbank aufwändige Vorberechnungen erfordern (vgl. Kapitel **4.2.3.3**).

6.4.1.4.7 Deixis

Für jede EQ-Anwendung lässt sich in den Projekt-Einstellungen diejenige Entität spezifizieren, worauf mittels „I“ bzw. „me“ verwiesen wird. Ist z.B. die Entität „customer“ angegeben, so erwartet EQ als Präzisierung einer Anfrage der Art „what have I ordered?“ die Eingabe eines Kundennamens im Klärungs-Dialogfenster:

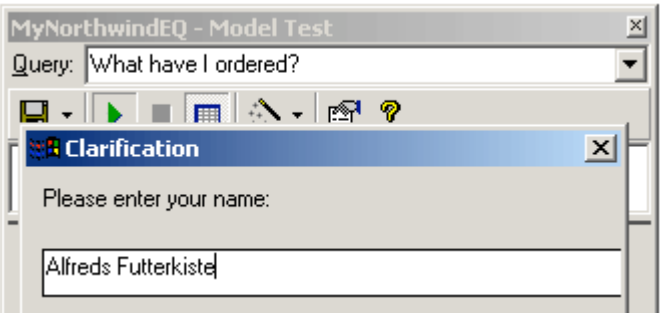


Abbildung 18 - Klärungs-Dialogfenster Deixis

Die Antwort lautet entsprechend „The products you have ordered are: […]“, wobei unbekannte Namen keine Fehlermeldung der Art „there is no customer named xy“, sondern eine leere Resultatmenge bewirken.
Temporal-deiktische Ausdrücke wie „today“, „last year“ usw., zur deren korrekter Interpretation Kenntnis des Äusserungszeitpunkts notwendig ist, behandelt EQ in jedem von mir getesteten Fall korrekt. Für jede Anwendung kann zudem in den Projekt-Einstellungen festgelegt werden, ob sich der Begriff „Jahr“ auf das Kalender- oder auf das Finanzjahr (inklusive Angaben zum Start-Datum) bezieht.

6.4.1.4.8 Fehlertoleranz

Für jede EQ-Applikation kann man festlegen, wie Wörter, welche weder im Lexikon enthalten noch als Datenbankwert identifiziert sind und deshalb potentielle „Non-words“ darstellen, behandelt werden: Wünscht man das Feedback des Benutzers, so wählt man die Option „Ask about spelling mistakes“ in den Projekt-Einstellungen; andernfalls „korrigiert“ EQ die vermeintlichen Fehler automatisch. Die dem Benutzer präsentierte Korrektur-Auswahl ist beschränkt auf diejenigen fünf Wörter des EQ-Lexikons, welche mit dem unbekannten Wort die meisten gemeinsamen Anfangsbuchstaben haben.
Solange der Anfangsbuchstabe des fehlerhaften Wortes mit dem gemeinten übereinstimmt und klein geschrieben ist, funktioniert die automatische Korrektur bei fehlenden, überzähligen oder vertauschten Buchstaben und bei falscher Gross-/Kleinschreibung im Wortinnern im Allgemeinen problemlos. Kritisch sind u.a. Ausdrücke mit grossem Anfangsbuchstaben, die EQ als (Eigen-)Namen und nicht als unbekannte Wörter interpretiert, und Ausdrücke mit falschem Anfangsbuchstaben, deren Korrektur-Mutationen letzteren übernehmen.
Die Rechtschreibkorrektur ist (gemäss meiner Beobachtungen als einzige Funktion) im Falle der *ASP*-Web-Anwendung nicht verfügbar. Stattdessen wird bei unbekannten Wörtern die Fehlermeldung „I don’t know the word ‚xyz’“ angezeigt.

EQ erkennt und korrigiert einfache Fälle der Verletzung von Subjekt-Verb-Numeruskongruenz im Hauptsatz (wie in „which product are in stock?“), während komplexere Fälle z.B. bei der Relativsatz-Anbindung (wie in „show the products that is in stock?“) eine Fehlermeldung generieren („Sorry, I didn’t understand that. Please check your spelling or phrasing.“).

Einen wesentlichen Beitrag zur Fehlertoleranz von EQ leistet der in den Projekt-Einstellungen zuschaltbare *Keyword-Modus* (*Keyword mode*). Dieser liefert einen „best guess“ der Bedeutung von Anfragen, die EQ normalerweise, d.h. unter Berücksichtigung aller Konstituenten, wegen fehlerhafter Syntax und/oder unbekannter Wörter nicht versteht. So „versteht“ EQ z.B. die Aufforderung „list the pretty customers“, auch wenn das Wort „pretty“ bzw. die Adjektiv-Phrasierung „some customers are pretty“ nicht im Modell enthalten ist.

6.4.1.4.9 Meta-Wissen

Versuche, in EQ Meta-Wissen über die Datenbank zu erfragen, sind fehlgeschlagen. Mittels Anfragen der Art „What do you know about ...“ kann man hingegen auf Informationen über Entitäten, d.h. auf Meta-Wissen über das semantische Modell zugreifen – z.B. auf Informationen über die Entität *product*:

```
Help Command
Type: Entity
Object: ENTITY:product
Help Text: A piece of merchandise to be bought or sold.

Summary Text:
products have categories, product names, product unit prices, product ids, product quantity
per units, product reorder levels, product units in stocks, and product units on orders.
products participate in the following relationships:
        products have suppliers
        customers order products from employees
        order details have products
        orders have products
        shippers ship products
```

Abbildung 19 - Meta-Wissen über das semantische Modell

Sofern das semantische Modell die in der Anfrage bezeichnete Entität enthält, liefert EQ diejenigen Modell-Informationen, welche die Entität betreffen: Angaben zur Entität, Angaben zu Attributen der Entität, d.h. zu semantisch abhängigen Entitäten, welche entweder in einer expliziten, im Modell spezifizierten Charakterzugs-Beziehung oder aber in einer semantisch impliziten, im Datenbank-Schema ausgedrückten Attribut-/Eigenschaftsbeziehung zur interessierenden Entität stehen, sowie Angaben zu den übrigen Beziehungen, an denen die Entität teilhat. Falls die interessierende Entität selbst eine Attribut-Entität ist, spezifiziert EQ anstatt der Attribute die Bezugs-Entität: z.B. „product id is an attribute of product“.

6.4.1.5 Beurteilung von EQ

Leider sind die inneren Prozesse der natürlichsprachlichen Analyse- und Generierungskomponente *Runtime Engine* weder einsehbar noch dokumentiert. Folgende Charakteristika (u.a.) rücken EQ allerdings in die Nähe von Front-End-Systemen wie SYSTEM X (vgl. Kapitel **4.4**):

- Trennung von domänenunabhängigem und domänenspezifischem Lexikon. Das umfangreiche domänenunabhängige Lexikon ist vorkonfiguriert und wird z.B. für die Korrektur von Nichtwörtern (und vermutlich bei der Syntaxanalyse) verwendet. Das domänenspezifische Lexikon umfasst einerseits all diejenigen Einträge, die als Werte von Entitäten, als benutzerdefinierte Entitäten(namen), als Synonyme zu Entitäten(namen), als eigene *Dictionary Entries* oder durch ihre Verwendung in einer Phrasierung dem Domänenmodell hinzugefügt werden, und andererseits diejenigen Entitäten(namen), welche bei der (halb)automatischen Generierung des Domänenmodells aus der Datenbank extrahiert werden. Während das domänenspezifische Lexikon Bestandteil des *English Query Model* ist, gehört das domänenunabhängige Lexikon (vermutlich) zur *Runtime Engine*.
- Parsing und Syntaxanalyse. EQ scheint verschiedene linguistische Verarbeitungsebenen zu trennen. Kann EQ einen Satz aufgrund fehlerhafter bzw. problematischer Syntax nicht analysieren, so wird eine Fehlermeldung der Art „Sorry, I didn’t understand that. Please check your spelling or phrasing. […]“ generiert. Falls die Struktur des Satzes unproblematisch, mindestens ein Konzept das Satzes jedoch semantisch nicht definiert ist, so teilt EQ dies dem Benutzer durch eine Fehlermeldung der Art „I don’t know how to do that.“ oder „I haven’t been given any information on [unbekanntes Konzept]“ mit.
- Modell der Anwendungsdomäne. Ähnlich wie bei SYSTEM X kann das Domänenmodell von EQ (halb)automatisch aus dem Datenbank-Schema extrahiert werden, was die Abbildung der semantischen Repräsentation auf die SQL-Anfrage wesentlich konsistenter macht.

Die Hauptvorzüge von EQ lassen sich, abgesehen von der soliden natürlichsprachlichen Analyse und der Generierung relativ differenzierter Fehlermeldungen, mit den Stichworten Flexibilität, Domänen-Portabilität und Habitabilität umschreiben:
- *Flexibilität*. Die wesentlichen domänenunabhängigen Bestandteile einer EQ-Anwendung, die *Runtime Engine* sowie die wichtigsten GUI-Funktionen, liegen als Komponenten vor und können so mit relativ geringem Aufwand in beinahe beliebige Softwareumgebungen, darunter moderne, Web-basierte Client-Server-Systeme, integriert werden. Das domänenspezifische semantische Modell kann je nach Anwendungskontext und Art des Zugriffs im passenden Format zur Verfügung gestellt werden, wobei die XML-konforme *SMF*-Repräsentation besonders offen und flexibel ist.
- *Domänen-Portabilität*. Diverse Assistenten, Test-Werkzeuge und nicht zuletzt die komfortable graphische Benutzeroberfläche reduzieren den Aufwand für die Erstellung und die Wartung eines Domänenmodells und damit den Aufwand für die Erstellung einer kompletten (Web-basierten) EQ-Anwendung massiv.
- *Habitabilität*. Die Benutzerschnittstelle, insbesondere die multimodale GUI-Komponente *Question Builder*, stellt ein dediziert auf die Bedürfnisse sowohl des *gelegentlichen* als auch des *naiven Endbenutzers* zugeschnittenes Retrieval-Werkzeug dar, welches dieser nach einer sehr kurzen Schulungsperiode effizient einsetzen kann. Beispielanfragen, explizite Paraphrasen und die explizite (graphische) Darstellung der semantischen Entitäten und Beziehungen wirken sich äusserst Habitabilitäts-fördernd aus. (Für OLAP-Clients stellt EQ entsprechende Komponenten zur Visualisierung und Analyse von Daten, z.B. in Form einer Pivot-Tabelle, zur Verfügung.)

Der eine wesentliche Nachteil von EQ ist dessen Plattformabhängigkeit. EQ kann prinzipiell nur auf der Microsoft-Windows-Plattform und nur im Zusammenspiel mit dem Microsoft *SQL Server* uneingeschränkt sowie „out of the box“ eingesetzt werden. EQ-Anwendungen lassen sich zwar mit einigem Programmieraufwand an alternative DBMS anbinden, sind jedoch bei der Generierung der SQL-Form auf den *SQL Server*-spezifischen Dialekt *Transact-SQL* festgelegt.
Der andere schwerwiegende Nachteil besteht darin, dass EQ bis anhin „English only“ ist. Berücksichtigt man die zurückhaltende Produktpolitik von Microsoft im Zusammenhang mit EQ (nur zusammen mit einer *SQL Server*-Lizenz zu erwerben) und die Beachtung, welche EQ in der Fachliteratur zuteil wird, (äusserst gering) so scheint es nicht allzu pessimistisch zu sein, innerhalb der nächsten zwei Jahre keine mehrsprachige Version von EQ zu erwarten.
Gesprochene Sprache unterstützt EQ nicht von sich aus; dank der Programmierschnittstellen sollten sich allerdings mit relativ geringem Aufwand entsprechende Module von Fremdherstellern einbinden lassen.

6.4.2 EasyAsk (English Wizard)

Die hier aufgeführten Angaben zu EASYASK bzw. zum Vorgängerprodukt ENGLISH WIZARD[81] stammen aus einer Produktpräsentation [EnglishWizard] sowie von der Website des Herstellers[82] ([Harris 98], [EasyAsk FAQ]), d.h. aus Marketing-Unterlagen, die zwar die Funktionalität der Software beschreiben, sich jedoch nicht als Basis für eine detaillierte (computerlinguistische) Beurteilung eignen. Ich werde deshalb primär den (dokumentierten) Funktionsumfang von EASYASK besprechen.

[81] ENGLISH WIZARD ist, zumindest was die Autorenschaft anbelangt, mit ROBOT und INTELLECT in eine genealogische Reihe zu stellen (vgl. Kapitel **3.3**).
[82] Linguistic Technology Corporation, gegründet 1994 von Dr. Larry R. Harris.

EASYASK wird gemäss Herstellerangaben [Harris 98] in mehr als 6500 Firmen „across industries and computing environments“ verwendet; der Hersteller bezeichnet sich damit als führend im Gebiet der „natural language information access technology and solutions“.

EASYASK basiert auf dem Front-End-Ansatz, wobei effizientes Top-down-left-right-Parsing ohne Backtracking zu einer (nicht weiter spezifizierten) Bedeutungsrepräsentation führt, die anschliessend in eine SQL-Anfrage übersetzt wird. Wie EQ ist EASYASK aufs Englische beschränkt; die Diskursfähigkeit, d.h. die Unterstützung anaphorischer Referenzen und Ellipsen, scheint mit derjenigen von EQ vergleichbar zu sein. Im Gegensatz zu EQ sieht EASYASK explizit die Möglichkeit der „nahtlosen“ Anbindung einer (unbestimmten) Speech-Processing-Software vor.
Die Front-End-typische, zusätzliche Wissensbasis, das Domänenmodell, wird bei EASYASK als *Dictionary* bezeichnet, vermutlich im Sinne eines semantischen Wörterbuchs. Domänen-Portabilität wird auch in diesem Fall als wichtig eingestuft: Ein bedeutender Teil („90%“) des Dictionary einer EASYASK-Anwendung wird automatisch generiert, indem aus der korrespondierenden Datenbank Metadaten, Beziehungen zwischen Relationen sowie die relevanten Daten extrahiert werden. Der eingebaute Thesaurus erweitert die konzeptuelle Abdeckung des Dictionary um eine Menge zusätzlicher Synonymbeziehungen; weitere Anpassungen soll ein Datenbank-Administrator, der über keinerlei linguistische Kenntnisse verfügt, mit dem „full-featured“ *Dictionary Editor* vornehmen können. Eine Datenbank-basierte *Logging Facility* zur Protokollierung der Anfragesitzungen und Werkzeuge für *Regressions-Tests* ermöglichen die effiziente und anwendungsbezogene Wartung und Optimierung des Domänenmodells. Ähnlich wie beim VIENA CLASSROOM NLI können mehrere Dictionaries in einer Hierarchie eingeordnet werden, wobei jeder Benutzer ein individuelles lokales Dictionary unterhält und die Änderungen des globalen Administrator-Dictionary propagiert werden.

Falls die natürlichsprachliche Anfrage von EASYASK nicht vollständig analysiert werden kann, d.h. falls z.B. unbekannte Wörter oder Ambiguitäten entdeckt werden, initiiert EASYASK einen Klärungsdialog mit dem Benutzer. Bei erfolgreicher Analyse präsentiert EASYASK dem Benutzer ein entsprechendes Feedback sowohl in Form einer englischen Paraphrase als auch in Form der SQL-Interpretation. Habitabilitäts-fördernd wirken sich 10 bis 20 Beispiel-FAQs aus, welche Anhaltspunkte zur linguistischen und konzeptuellen Abdeckung liefern und vom Benutzer direkt modifiziert werden können.

Abgesehen von der reinen Retrieval-Funktionalität enthält EASYASK auch einige Werkzeuge zur Analyse und Visualisierung von Daten, u.a. Spreadsheets, Berichte und Pivot-Tabellen. So kann sich der Benutzer das Resultat einer Anfrage z.B. in Form eines Kuchen-Diagramms ausgeben lassen: „Give me a pie chart of sales by category for last month ?“

Hinsichtlich möglicher Einsatzumgebungen ist EASYASK sehr flexibel: Die Software kann als Stand-alone-Anwendung, als Zusatz-Komponente für Anfrage- und Report-Werkzeuge wie z.B. *MS Access*, als eingebettete Komponente einer beliebigen Client-Server-Anwendung, als Teil eines Webservers[83] sowie als vollständige Client-Server-Umgebung[84] verwendet werden, wobei dasselbe Dictionary in allen Architekturen seinen Dienst versieht. Im Gegensatz zu EQ ist EASYASK nicht auf ein Datenbanksystem festgelegt, sondern unterstützt die SQL-Dialekte zahlreicher „Real-World“-RDBS wie MS Access, SQL Server, Oracle, Informix und Sybase. Insofern kann man bei EASYASK von eigentlicher DBMS-Portabilität sprechen.

[83] Direkt unterstützt sind Microsofts *IIS* und der *Netscape Web Server*.
[84] D.h.: Benutzerschnittstelle beim Client, NLP auf dem Server.

Der folgende Screenshot zeigt die Client-Benutzerschnittstelle der Web-basierten Client-Server-Lösung des Vorgänger-Produkts ENGLISH WIZARD. Besondere Beachtung verdienen die folgenden Elemente der Benutzeroberfläche:

- Auswahlliste mit Beispielanfragen, gruppiert nach Thema oder gewünschter Formatierung.
- Liste zur Auswahl der Resultat-Formatierung, in diesem Fall eine (unten angezeigte) manipulierbare, interaktive Pivot-Tabelle.
- Liste zur Auswahl des Feedback-Modus (Paraphrase/SQL-Anfrage).

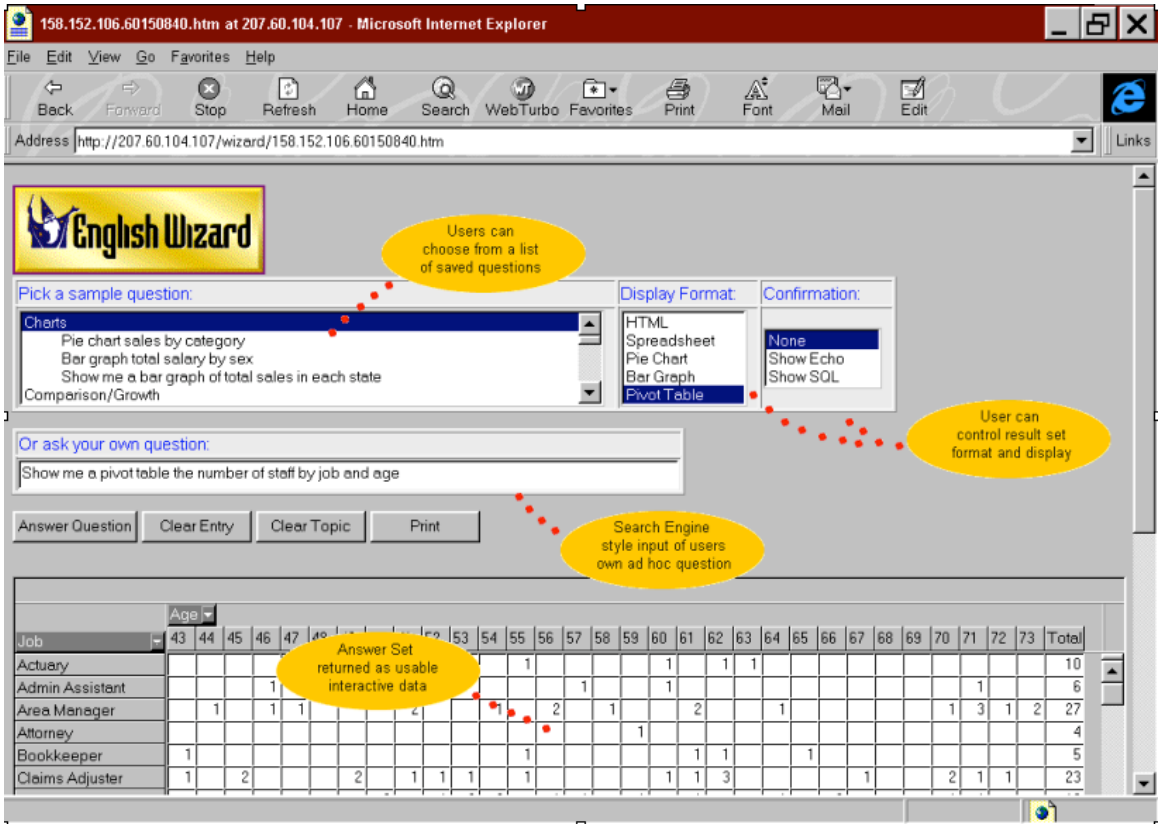


Abbildung 20 - Client-Schnittstelle der Web-basierten Client-Server-Lösung von ENGLISH WIZARD[85]

7 Einsatz von NLIDB-Systemen

Angesichts der Vorteile, welche ein NLIDB gegenüber alternativen Schnittstellen aufzuweisen hat, kann deren prinzipieller Nutzen nur schwer von der Hand gewiesen werden:

- Der Benutzer muss keine formale DML lernen.
- Die natürliche Sprache ist im Vergleich zu graphischen und Menü-basierten Schnittstellen wesentlich flexibler und ausdrucksstärker.

Allerdings scheint die Konzipierung eines „idealen“ NLIDB-Systems, welches die obengenannten Vorteile nicht durch kritische Schwächen im Bereich des Sprachverstehens, im Bereich der Habitabilität sowie im

[85] Vgl. [EnglishWizard].

Bereich der Kosteneffektivität zunichte macht, nach wie vor ein schwieriges Unterfangen zu sein (vgl. Kapitel **5**).
Die folgenden Unterkapitel sollen die wichtigsten Erkenntnisse dieser Arbeit in Bezug auf den sinnvollen Einsatz natürlichsprachlicher Datenbankschnittstellen zusammenfassen. Dabei werden sowohl mögliche Einsatzszenarien als auch Fragen des Entwurfs adäquater natürlichsprachlicher Benutzerschnittstellen diskutiert.

7.1 Einsatzszenarien

Was den allgemeinen Status Quo der Informationstechnologie, aktuelle Entwicklungen der Datenbanktechnologie sowie mögliche Einsatzszenarien für natürlichsprachliche Schnittstellen im Zusammenhang mit Datenbanken betrifft, bin ich im Verlauf meiner Untersuchungen u.a. zu folgenden Erkenntnissen gekommen:
- Während relationale Datenbanken wohl auch heute noch den Markt dominieren, so scheint doch das Feld der Datenbanksysteme aufgrund neuer Anwendungsgebiete und darauf abgestimmter neuer Datenbank-Technologien zunehmend *heterogener* zu werden. (Kapitel **2.4**)
- Neue Anwendungskontexte führen zu neuen Anforderungen an Benutzerschnittstellen, wobei hier besonders auf das Problem *heterogener Informationsquellen*, z.B. im *Internet*, bei eigentlichen *heterogenen Datenbanksystemen* oder auch im Fall der *Data Warehouses*, sowie auf das Problem der ungeschulten Gelegenheitsbenutzer gerade im Bereich des Internets hingewiesen sei. (Kapitel **2.4, 2.5, 3.4.3**)
- Aufgrund ihrer Domänenunabhängigkeit und *Universalität* ist die natürliche Sprache prinzipiell auch bei heterogenen Informationsumgebungen als einheitliches Interaktionsmittel geeignet – ganz abgesehen von ihrer Vertrautheit und Zugänglichkeit. (Kapitel **3.2**)
- Die natürlichsprachliche Modalität, wie sie momentan von NLIDBs zur Verfügung gestellt wird, ist besonders geeignet für den *Gelegenheitsbenutzer*, der Wert auf maximale Flexibilität und Effizienz sowie auf minimalen Einarbeitungsaufwand legt. Für den *naiven* und den *intellektuellen* Endbenutzer wird sich der Aufwand der Installation und der Konfigurierung eines NLIDB nicht in jedem Fall auszahlen: Während der naive Benutzer keiner allzu flexiblen Schnittstelle bedarf, weil er im Prinzip nur eine geringe Anzahl standardisierter und vordefinierter Prozeduren benötigt, wird der intellektuelle Benutzer je nach Verfügbarkeit und konkreter Anfragesituation entweder die flexible natürlichsprachliche Modalität oder die explizite und eindeutige graphische Modalität oder aber, bei problematischen Fällen, die formalsprachliche Modalität bevorzugen, welche maximale Kontrolle bei der Formulierung der Anfrage bietet. (vgl. Kapitel **3.4.3**)
- Eine Antwort auf die Frage nach Anwendungsbereichen, in denen der Einsatz einer natürlichsprachlichen Schnittstelle sinnvoll erscheint, fällt je nach Perspektive ziemlich unterschiedlich aus: Der (theoretischen) Aussage von [Hall et al. 96], dass sich (vereinfacht ausgedrückt) der Einsatz eines NLI dann empfiehlt, wenn alternative (graphische/deiktische) Schnittstellen versagen, d.h. wenn die Domäne aus einer grossen, ungeordneten und nicht vertrauten Menge von Objekten besteht, widerspricht die Beobachtung, dass natürlichsprachliche Schnittstellen gerade bei Datenbanken aufgrund deren *begrenzten Anwendungsdomänen* in der Praxis eher funktionieren als in anderen Kontexten.

Ein konkretes, einleuchtendes Einsatzszenario stellt die Anbindung an ein *Data Warehouse* dar, wie es bei den „modernen“ NLIDB-Systemen ENGLISH QUERY und EASYASK propagiert wird. Sowohl dem Datenanalytiker, d.h. dem intellektuellen Benutzer, als auch dem Entscheidungsträger, d.h. dem Gelegenheitsbenutzer, wird ein flexibles und natürliches Werkzeug geboten, wobei die Schnittstelle um zusätzliche Eingabe- und Ausgabe-Modalitäten ergänzt ist bzw. sein sollte. Bei diesem Szenario wirkt sich auch die Beschränkung auf Datenwiedergewinnung nicht negativ aus: Bei der Datenanalyse werden typischerweise bestehende Daten ausgewertet und nicht neue Daten hinzugefügt.

Berücksichtigt man die Probleme, die sich bei der Entwicklung, Konfiguration und Verwendung natürlichsprachlicher Datenbankschnittstellen im konventionellen Sinne, d.h. für (isolierte) relationale Datenbanken, stellen, so drängt sich die Frage auf, ob der Nutzen den Zusatz-Aufwand überhaupt rechtfertigt, ob der Bedarf nach NLI beim vielfältigen Angebot an Alternativen überhaupt vorhanden ist.
Eine andere Ausgangslage ist hingegen im Fall der heterogenen Datenbankumgebungen gegeben: Disparate Datenquellen sollen unter einer einheitlichen Benutzerschnittstelle zusammengefasst werden, wobei von bestehenden formalen Abfragesprachen aufgrund deren Verschiedenartigkeit abstrahiert und eine Datenquellen-unabhängige Integrationsinstanz geschaffen werden muss. Da das Konzept der universellen, anwendungsunabhängigen Zwischenrepräsentation und deren Übersetzung in einen anwendungsspezifischen Formalismus im Bereich der konventionellen NLIDB intensiv erforscht und geprüft worden ist, liegt es nahe, entsprechende Erkenntnisse wiederzuverwenden – z.B. bei der Entwicklung der Benutzerschnittstelle und der *Mediator*-Komponente eines *Daten-Integrationssystems* (vgl. Kapitel **2.5.2**).
Ein *Daten-Integrationssystem* besteht im Wesentlichen aus der Benutzerschnittstelle, aus dem *Mediator*, aus Schnittstellen zu disparaten Datenquellen sowie aus einer Sammlung von Metadaten[86] zu den Datenquellen:

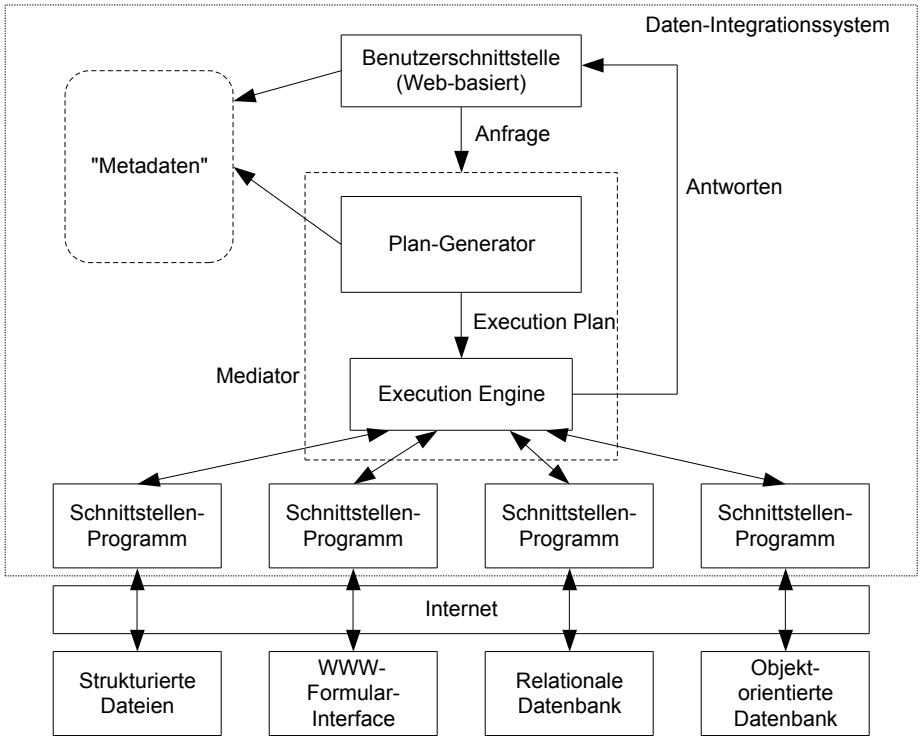


Abbildung 21 - Architektur eines Daten-Integrationssystems[87]

[86] „Metadaten“ eines Daten-Integrationssystems beschreiben einerseits die Anwendungsdomäne, meistens in Form eines „mediated schema“ [Florescu et al. 98], d.h. einer Sammlung virtueller Datenbankobjekte, welche vom Benutzer für die Formulierung der Anfrage verwendet werden können, und andererseits die Datenquellen, d.h. deren Inhalt und Fähigkeiten.
[87] Vgl. [Florescu et al. 98].

Der Mediator stellt einen vereinheitlichten Zugang zu ungleichartigen Daten zur Verfügung. [Saelee et al.] Er nimmt die Anfrage des Benutzers entgegen, generiert einen Ausführungs-Plan, der die verschiedenen anwendungsspezifischen Anfragen spezifiziert, leitet letztere an die verschiedenen Datenquellen (bzw. an die entsprechenden Schnittstellen) weiter, kombiniert die Resulate und liefert sie dem Benutzer zurück.
Sind die zugrundeliegenden Daten primär bezüglich ihres Formats disparat, nicht hingegen bezüglich der Anwendungsdomäne, so sollte die Anbindung einer bestehenden Sprachverstehens-Komponente an den Mediator mit relativ hohem Anpassungsaufwand[88] machbar sein. Noch interessanter scheint ein „integrativer Ansatz" zu sein, der auf einer Zwischenrepräsentation basiert, welche einerseits das Ergebnis der natürlichsprachlichen Analyse darstellt und andererseits direkt vom Plan-Generator weiter verarbeitet werden kann. Fraglich ist, ob man bei einem solchen Szenario noch von einem „NLIDB" sprechen kann – schliesslich verschwimmen gerade beim Daten-Integrationssystem die Grenzen zwischen Information-Retrieval-, Text-Retrieval- und Datenbanksystem zusehends. (Dies gilt in besonderem Masse auch für das Thema *Modellierung und Befragung des WWW* (Kapitel **2.5.2**), das ich hier nicht weiter ausführen werde.)

7.2 Entwurf eines NLIDB

Durch eine Begrenzung der Anwendungsdomäne kann Mensch-Maschine-Kommunikation auch dann gelingen, wenn nicht alle linguistischen Phänomene und Probleme erschöpfend behandelt bzw. gelöst werden. Hier soll zusammengestellt werden, welche Anforderungen an ein NLIDB gestellt werden, wie ein entsprechendes System konzipiert sein könnte und worauf beim Entwurf eines NLIDB generell zu achten ist.
- *Datenbanken*. ENGLISH QUERY zeigt, dass die Kombination von relationaler Datenbank und Front-End-System durchaus praxistauglich ist – was angesichts der diesbezüglichen langjährigen Forschungsbemühungen eigentlich den Erwartungen entspricht. Eine wichtige Voraussetzung für den effizienten und effektiven Einsatz eines linguistischen Front-Ends besteht allerdings darin, dass auch die zugrunde liegende Datenbank „logisch", d.h. im Hinblick auf die semantischen Objekte und Beziehungen der Anwendungsdomäne, strukturiert ist. So sollte eine Relation z.B. höchstens ein semantisches Objekt repräsentieren (und nicht deren zwei, inklusive einer Beziehung, wie dies der Fall ist beim *„Doctor-on-board"*-Beispiel (Kapitel **4.2.4**)).
- *Modularität*. Das Stichwort „Modularität" ist im Bereich der NLIDB relativ umstritten. Während man im Prinzip, d.h. unter dem Aspekt der DBMS-Portabilität, eigentlich für die Front-End-typische Trennung von Linguistik-Komponente und Datenbank-Komponente plädieren müsste, stellt sich in der Praxis eben diese Trennung als problematisch heraus. Einfach ausgedrückt, besteht das Problem darin, dass die Linguistik-Komponente in vielen Fällen nicht bzw. nicht effizient auf die Datenbank-Komponente respektive deren semantisches Modell zugreifen kann – integrierte Systeme wie MASQUE/PRO (Prolog-Datenbank), LOQUI (SQL-Datenbank) und VIENA CLASSROOM NLI (deduktive objekt-orientierte DB) sind davon Prinzip-bedingt nicht betroffen. Systeme wie SYSTEM X/SYSTEMX und EASYASK, deren Analyse auf einem (modifizierten) semantischen Modell der Datenbank basiert, zeigen, dass sich DBMS-Portabilität und effizienter Zugriff auf die Datenbank-Komponente zwar nicht kategorisch ausschliessen, dass jedoch Kompromisse in Bezug auf die deduktiven Fähigkeiten des Systems einzugehen sind – was sich z.B. im Bereich der kooperativen Antworten manifestiert.

[88] Ein hoher Aufwand fällt beim Generieren/Konfigurieren des Domänenmodells an, da die Wissensextraktion aus den „Metadaten" wohl wesentlich komplexer als im Falle eines Relationenschemas ist und da der Zugriff auf die eigentlichen Daten nur über die anwendungsspezifischen Schnittstellen erfolgt. Zusätzlich muss ein Modul für die Übersetzung der Zwischenrepräsentation in einen Formalismus, welchen der Plan-Generator versteht, entwickelt werden.

- *Robustheit*. Robuste(re) NLP-Techniken sind gerade bei „Real-World"-Anwendungen zentral. Während z.B. LOQUI zumindest im Syntaxbereich keinerlei Fehler toleriert, ist ENGLISH QUERY wesentlich nachsichtiger und korrigiert – je nach Projektvorgaben – spezifische Fehler sogar ohne Dazutun des Benutzers. Die Grundlage für eine effektive Sprachverarbeitung bilden unter anderem ausgereifte Grammatik-Formalismen, so z.B. der beim neueren SYSTEMX eingesetzte HPSG-Formalismus [Cercone et al. 93], der sich nicht zuletzt dadurch von den verbreiteten ad-hoc-Grammatiken [Androutsopoulos and Ritchie 00] abhebt, dass er die parallele Analyse von Syntax und Semantik zulässt. So lassen sich z.B. Ambiguitäten effektiver behandeln, als dies im Fall der strikten Trennung der linguistischen Ebenen möglich wäre.[89]
- *Erhöhung der linguistischen Abdeckung*. Die zunehmende Verfügbarkeit umfangreicher linguistischer Ressourcen wie Textkorpora, maschinenlesbare Wörterbücher[90] usw. sowie von (stochastischen oder konnektionistischen) Methoden zu deren Verarbeitung ermöglicht Sprachverstehens-Systeme mit deutlich besserer linguistischer Grundausstattung, was die Robustheit erhöht und gleichzeitig den Konfigurierungs-Aufwand reduziert. Methoden des *Machine Learning*, wie sie z.B. beim VIENA CLASSROOM NLI zum Einsatz kommen, ermöglichen die Erweiterung des linguistischen und konzeptuellen Wissens während des Gebrauchs der Anwendung – das ideale NLIDB-System ist „lernfähig" und passt sich dem Benutzer an.
- *Diskursanalyse*. Diskursphänomene wie anaphorische Referenzen und Ellipsen treten häufig und unabhängig vom Umfang der Anwendungsdomäne auf. Der darauf basierende Folgefragen-Mechanismus stellt einen bedeutenden Effizienzfaktor dar und sollte von einem NLIDB angeboten werden.
- *Verarbeitung gesprochener Sprache*. In Anbetracht der verfügbaren Rechenleistung bei moderner Hardware sollte der Einsatz von Modulen, die gesprochene Sprache effizient verarbeiten, problemlos sein. Durch Unterstützung von *Speech Processing* erschliessen sich neue Einsatzgebiete, z.B. im Zusammenhang mit telefonischen Auskunftssystemen.
- *Portabilität*. Ausgereifte NLIDB-Systeme können gewöhnlich mittels dedizierten Portierungswerkzeugen für verschiedene Anwendungsdomänen konfiguriert werden. Diese Domänen-Portierung ist zwar nach wie vor relativ aufwändig, Zahlen wie die bei EASYASK genannten[91] sowie eigene Erfahrungen mit English Query deuten jedoch darauf hin, dass das Problem durch Methoden der (halb)automatischen Wissensextraktion wesentlich entschärft werden kann.
- *Habitabilität*. Ein wichtiger Aspekt der Mensch-Maschinen-Kommunikation ist das Verständnis des Benutzers für die Fähigkeiten und die Grenzen des Systems. Diese sollten möglichst offensichtlich sein:
- Das System darf dem Benutzer keine „Intelligenz" vortäuschen, indem es z.B. elaborierte Sprachgenerierung verwendet. Besser geeignet sind standardisierte, einfache Antworten, System-Statusanzeigen (ENGLISH QUERY: „5:40:37 PM: Request parsed...") sowie, kombiniert mit einfacher Textausgabe, die Ausgabe von Tabellen, um die Datenbankstruktur zu verdeutlichen.
- Das System sollte Informationen über die linguistische Abdeckung zur Verfügung stellen, was im einfachsten Fall durch Anzeige einer Menge von Beispielanfragen geschieht. Dies liefert dem Benutzer einen Anhaltspunkt, wie er seine Fragen System-gerecht formulieren kann.
- Das System sollte Informationen über die konzeptuelle Abdeckung bereitstellen, d.h. Informationen darüber, wonach man fragen kann. Auch dafür eignen sich Beispielanfragen.

---

[89] Siehe auch Kapitel **6.3.2** (VIENA CLASSROOM NLI).
[90] [Dahl et al. 00] beschreibt ein System, das u.a. ein Comlex-basiertes externes syntaktisches Wörterbuch sowie ein WordNet-basiertes externes semantisches Wörterbuch verwendet.
[91] „In most cases, this automated process results in a working dictionary approximately 90% complete. […] On average, this entire dictionary construction process should require less than one week for every 40+ tables. The recommended Dictionary Administration skills are that of typical a Database Administrator. Absolutely no knowledge of linguistics is required." Vgl. [EasyAsk FAQ].

- Das System sollte informative Fehlermeldungen generieren. Dieses Postulat umfasst sowohl den Fall eines Fehlschlags des Verstehensprozesses als auch den Fall eines Fehlschlags des Daten-Retrievals. Der Benutzer sollte im Idealfall der Fehlermeldung entnehmen können, ob (und inwiefern) die Struktur und/oder der Inhalt seiner Anfrage den Fehlschlag der Analyse verursacht hat oder ob zwar die Anfrage verarbeitet, jedoch keine entsprechenden Ergebnisse gefunden werden konnten. Falls keine Klarheit schaffende Fehlermeldung erzeugt wird, versucht der Benutzer u.U., eine Fehlerquelle zu beseitigen, die gar nicht existiert, während die tatsächliche Ursache unentdeckt bleibt.
- *Eingabe- und Ausgabe-Modalitäten*. Systeme wie ENGLISH QUERY zeigen, wie und dass sich (zumindest eingeschränkt) multimodale Umgebungen sowohl bei der Domänenkonfigurierung als auch beim Gebrauch der Schnittstelle nutzbringend einsetzen lassen. Nutzbringend deshalb, weil sowohl die Habitabilität als auch die generelle „Benutzerfreundlichkeit“ verbessert werden. Gerade im Bereich der Datenanalyse und des *Data Mining* sind Ausgabe-Modalitäten gefordert, die nur wenig mit den Kommandozeilen-Antworten älterer NLIDB-Systeme gemeinsam haben. Systeme wie SYSTEMX, ENGLISH QUERY und EASYASK verfügen z.B. über mehr oder weniger ausgereifte Funktionen zur Datenvisualisierung und zur Generierung standardisierter Berichte.[92]
- Weitere Anforderungen. Eine ausführliche Behandlung der spezifischen Anforderungen an moderne, netzwerktaugliche und verteilte „Real-World“-Anwendungen würde den Rahmen dieser Untersuchung sprengen (und die Kompetenz des Autors übersteigen). Für einen Überblick sei hier verwiesen auf [Dahl et al. 00].
- *Entwicklungsmethodologie*. Manaris [Manaris 98] betont die Wichtigkeit des „user-centered approach“: Während des gesamten Entwicklungsprozesses soll der Endbenutzer mit einbezogen werden, was mit der empirischen Ermittlung von natürlichsprachlichen Anfragedaten zur Bestimmung der benötigten Untermenge natürlicher Sprache („sublanguage analysis“) beginnt und in der stetigen Auswertung von Benutzer-Feedback seine Fortsetzung findet. Empirische Daten sammelt man durch Interviews (z.B. bei SYSTEMX [Cercone et al. 93]) oder, etwas aufwändiger, in sog. *Wizard-of-Oz Experimenten*[93]; Fehlerprotokolle (EASYASK) oder vom Benutzer generierte Listen von Anfragen, welche das System nicht „versteht“, (ENGLISH QUERY) dienen als Anhaltspunkte für die stetige Verfeinerung linguistischer und semantischer Modelle.

Den Anfang der vorliegenden Arbeit bildet die Feststellung, dass NLIDBs in Forscherkreisen „out of fashion“ geraten sind – was allerdings, so müsste man hinzufügen, nur die halbe Wahrheit ist. Zwar scheint das Konzept der natürlichsprachlichen Schnittstelle zu einer konventionellen, isolierten SQL-Datenbank aufgrund mächtiger Schnittstellen-Alternativen und mangels entsprechender Anwendungsfälle tatsächlich in den Hintergrund gerückt worden zu sein, an dessen Stelle sind jedoch neue „Visionen“ getreten, welche nicht zuletzt den Entwicklungen im Datenbanksektor und im Bereich der Schnittstellen-Technologie Rechnung tragen. „Integration“ steht diesbezüglich nicht nur für das Verschmelzen von natürlichsprachlichen Schnittstellen und Datenbanken zu deduktiven, „wissensbasierten“ Systemen, sondern generell für die Vereinigung verschiedenster Technologien aus verschiedenen Forschungsdisziplinen zu Datenbank-gestützten, multimodalen Informationssystemen. Wu liefert dazu das passende Fazit:

---

[92] SYSTEMX: vgl. [Cercone et al. 93], ENGLISH QUERY: vgl. [EQ 2000], EASYASK: vgl. [Harris 98].
[93] Vgl. z.B. [Murray et al. 98]. Wizard-of-Oz Experimente simulieren ein nicht vorhandenes System, z.B. ein NLIDB, indem sie anstelle der System-„Intelligenz“ einen Menschen, der die Maschinen-Intelligenz imitiert, auf die Anfragen der Testpersonen reagieren lassen. Das Ergebnis ist eine Sammlung von Anfragen, die insofern ziemlich repräsentativ ist, als sie auf einer der Mensch-Maschine-Kommunikation nachempfundenen Situation beruht.

The general shape of the next generation of natural language database access systems is becoming clear. In such a system, the user would not be questioning an alien database, but would instead be engaging in a conversation with a friendly, responsive information system. The natural language interpretation and generation routines of this information system would have to emulate many of the discourse techniques humans use in communicating with one another. The information access routines would have to be able to access much more than just extensional data, and to do so even during the interpretation and generation of natural language. In short, the next generation of natural language front ends to databases will not be front ends at all, but will be integrated understanding systems, and will not be querying databases, but will be flexibly interfacing with knowledge bases. [Wu et al. 96 : 5]

8 Appendix A: Bibliographie

[Androutsopoulos 92] Ioannis Androutsopoulos (1992). Interfacing a Natural Language Front-End to a Relational Database. M. Sc. Dissertation. Department of Artificial Intelligence. University of Edinburgh. Web: http://citeseer.nj.nec.com/androutsopoulos92interfacing.html.

[Androutsopoulos 96] Ioannis Androutsopoulos (1996). A Principled Framework for Constructing Natural Language Interfaces to Temporal Databases. PhD thesis. Department of Artificial Intelligence, University of Edinburgh.

[Androutsopoulos and Ritchie 00] Ion Androutsopoulos and Graeme Ritchie (2000). Database Interfaces. In: R. Dale, H. Moisl, H. Somers (eds.): A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text. Marcel Dekker Inc. 209-240. Web: http://citeseer.nj.nec.com/437934.html.

[Androutsopoulos et al. 93] I. Androutsopoulos, G. Ritchie, P. Thanisch (1993). MASQUE/SQL – An Efficient and Portable Natural Language Query Interface for Relational Databases. In: P.W. Chung, G. Lovegrove, M.Ali (Hg.): Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Edinburgh, U.K. Langhorne, PA, U.S.A. 327-330. Web: http://citeseer.nj.nec.com/145206.html.

[Androutsopoulos et al. 95] I. Androutsopoulos, G. D. Ritchie, P. Thanisch (1995). Natural Language Interfaces to Databases – An Introduction. In: Journal of Natural Language Engineering. Cambridge University Press. 1 (1), 1995. 29-81. Web: http://citeseer.nj.nec.com/androutsopoulos95natural.html.

[Auxerre 86] P. Auxerre (1986). MASQUE Modular Answering System for Queries in English – Programmer's Manual. Technical Report AIAI/SR/11. Artificial Intelligence Applications Institute, University of Edinburgh. March 1986.

[Auxerre and Inder 86] P. Auxerre and R. Inder (1986). MASQUE Modular Answering System for Queries in English – User's Manual. Technical Report AIAI/SR/10. Artificial Intelligence Applications Institute, University of Edinburgh. June 1986.

[Ballard et al. 84] B. W. Ballard, J. C. Lusth, N. L. Tinkham (1984). LDC-1: A Transportable, Knowledge-Based Natural Language Processor for Office Environments. In: ACM Transactions on Office Information Systems. 2(1), January 1984. 1-25.

[Ballard and Stumberger 86] B. Ballard and D. Stumberger (1986). Semantic Acquisition in TELI. In: Proceedings of the 24th Annual Meeting of ACL 1986. New York. 20-29.

[Bates et al. 86] M. Bates, M.G. Moser, D. Stallard (1986). The IRUS Transportable Natural Language Database Interface. In: Expert Database Systems. L. Kerschberg (ed.). Menlo Park, CA. 617-630.

[Binot 91] Jean-Louis Binot (1991). Natural Language Processing and Logic. In: André Thayse (ed.): From Natural Language Processing to Logic for Expert Systems. A Logic Based Approach to Artificial Intelligence. Chichester. 49-116. (= Kapitel 2)

[Binot et al. 88] Jean-Louis Binot, B. Demoen, K. Hanne, L. Solomon, Y. Vassiliou, W. von Hahn, T. Wachtel (1988). LOKI: A Logic Oriented Approach to Data and Knowledge Bases Supporting Natural Language Interaction. In: Proceedings of the ESPRIT 88 Conference, North-Holland, Amsterdam.

[Bretan and Karlgren 93] Ivan Bretan and Jussi Karlgren (1993). Synergy Effects in Natural Language-Based Multimodal Interaction. Proceedings of 1993 ERCIM Workshop on Multimodal Human-Computer Interaction, Nancy. (Auch erhältlich als SICS Research Report R94:04). Web: http://citeseer.nj.nec.com/bretan94synergy.html.

[Bussmann 90] Hadomud Bussmann (1990). Lexikon der Sprachwissenschaft. Stuttgart. (=2., völlig neu bearbeitete Auflage)

[Carstensen et al. 01] Kai-Uwe Carstensen et al. (eds.) (2001). Computerlinguistik und Sprachtechnologie. Eine Einführung. Heidelberg, Berlin.

[Catarci et al. 95] Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, Carlo Batini (1995). Visual Query Systems for Databases: A Survey. Technical Report SI/RR-95/17, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza". Web: http://citeseer.nj.nec.com/catarci95visual.html.

[Cercone et al. 93] Nick Cercone, Paul McFetridge, Fred Popowich, Dan Fass, Chris Groeneboer, Gary Hall (1993). The SystemX Natural Language Interface: Design, Implementation and Evaluation. Technical Report CSS-IS TR 93-03, Centre for Systems Science, Simon Fraser University, Burnaby. Web: http://citeseer.nj.nec.com/106152.html.

[Charniak 93] Eugene Charniak (1993). Statistical Language Learning. Massachusetts.

[Codd 70] E. F. Codd (1970). A Relational Model for Large Shared Data Banks. In: Communications of the ACM. 13:6, June 1970. New York. 377-387.

[Codd 74] E. F. Codd (1974). Seven Steps to RENDEZVOUS with the Casual User. In: Data Base Management. J. Kimbie and K. Koffeman (eds.).

[Coutaz and Caelen 91] J. Coutaz and J. Caelen (1991). A Taxonomy for Multimedia and Multimodal User Interfaces. In: Proceedings of the ERCIM Workshop on User Interfaces and Multimedia. European Research Consortium for Informatics and Mathematics, Lisbon, Portugal. 143-147.

[Dahl et al. 00] Deborah A. Dahl, Lewis M. Norton, K. W. Scholz (2000). Commercialization of Natural Language Processing Technology. In: Communications of the ACM 43:7, July 2000. New York.

[Damerau 81] F. Damerau (1981). Operating Statistics for the Transformational Question Answering System. In: American Journal of Computational Linguistics. 7/1981. 30-42.

[Damerau 85] F. Damerau (1985). Problems and Some Solutions in Customization of Natural Language Front Ends. In: ACM Transactions on Office Information Systems. 3(2), April 1985. 165-184.

[EasyAsk FAQ] Easy Ask Frequently Asked Questions. Web: http://www.ccasoftware.co.uk/ewfaq.htm.

[Elmasri and Navathe 94] Ramez Elmasri and Shamkant B. Navathe (1994). Fundamentals of Database Systems. Second Edition. Redwood City, California.

[EnglishWizard] English Wizard by Linguistic Technology Corporation. Powerpoint-Präsentation. Web: http://www.ccasoftware.co.uk.

[EQ 2000] English Query 2000. English Query Books Online, English Query Tutorials. (Im Lieferumfang von Microsoft English Query 2000.)

[Florescu et al. 98] Daniela Florescu, Alon Levy, Alberto Mendelzon (1998). Database Techniques for the World-Wide Web: A Survey. In: SIGMOD Record (ACM Special Interest Group on Management of Data). 27 (3), 1998. 59-74. Web: http://citeseer.nj.nec.com/article/florescu98database.html.

[Grosz et al. 87] B. J. Grosz, D. E. Appelt, P. A. Martin, F. C. N. Pereira (1987). TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. In: Artificial Intelligence. Vol. 32, no. 2, 1987. 173-243.

[Hafner 84] C. D. Hafner (1984). Interaction of Knowledge Sources in a Portable Natural Language Interface. In: Proceedings of the 22nd Annual Meeting of ACL 1984. Stanford, CA. 57-60.

[Hafner and Godden 85] C. D. Hafner and K. Godden (1985). Portability of Syntax and Semantics in Datalog. In: ACM Transactions on Office Information Systems. 3(2), April 1985. 141-164.

[Hall et al. 96] Gary Hall, Fred Popowich, Dan Fass (1996). Natural Language Edit Controls: Constrained Natural Language Devices in User Interfaces. In: Proceedings of IEEE 8th International Conference on Tools with Artificial Intelligence. IEEE Computer Society Press, Los Alamitos, California. 475-477. Web: http://citeseer.nj.nec.com/188003.html.

[Hammer 98] Joachim Hammer (1998). Data Warehousing. Overview, Terminology, and Research Issues. Seminarunterlagen. Web: http://www.cise.ufl.edu/~jhammer/classes/wh-seminar/Overview/index.htm.

[Harris 77] Larry R. Harris (1977). User Oriented Database Query with the ROBOT Natural Language Query System. In: International J. Man-Machine Studies. Vol. 9, 1977. 697-713.

[Harris 98] Larry R. Harris (1998). Building a Self Service Data Warehouse. Reaching Beyond the Power User. EasyAsk White Paper. Web: http://www.ccasoftware.co.uk/ewwp.htm.

[Hendrix et al. 78] G. Hendrix, E. Sacerdoti, D. Sagalowicz, J. Slocum (1978). Developing a Natural Language Interface to Complex Data. In : ACM Transactions on Database Systems. Vol. 3, no. 2, 1978. 105-147.

[Ishikawa et al. 87] H. Ishikawa, Y. Izumida, T. Yoshino, T. Hoshiai, A. Makinouchi (1987). KID Designing a Knowledge-Based Natural Language Interface. In: IEEE EXPERT. Vol. 2, no. 2, 1987. 57-71.

[Kaplan 84] S. J. Kaplan (1984). Designing a Portable Natural Language Database Query System. In: ACM Transactions on Database Systems. Vol. 9, no. 1, 1984. 1-19.

[LIM 92] Language Industry Monitor (1992). BIM: testing the NLP waters. Web: http://www.lim.nl/monitor/bim.html.

[Linke et al. 94] Angelika Linke, Markus Nussbaumer, Paul R. Portmann (1994). Studienbuch Linguistik. Tübingen. (= 2. Auflage, ergänzt um ein Kapitel „Phonetik und Phonologie“ von Urs Willi)

[LOQUI 91] Loqui – A Natural Language Query System for Databases. User Guidelines. July 1991.

[Lucas 88] Robert Lucas (1988). Database Applications Using Prolog. Chichester. (Ellis Horwood Books in Computing Science. Series in Computers and Their Applications)

[Manaris 98] Bill Z. Manaris (1998). Natural Language Processing: A Human-Computer Interaction Perspective. Preprint of chapter appearing in Advances in Computers vol. 47. Marvin V. Zelkovitz (ed.). Computer Science Department, University of Southwestern Louisiana, Lafayette, Louisiana. 1-66. Web: http://www.cacs.louisiana.edu/~manaris/publications/advanced-in-computers-vol-47.html.

[Manning and Schütze 1999] Christopher D. Manning and Hinrich Schütze (1999). Foundations of Statistical Natural Language Processing. Massachusetts.

[Marcus 86] Claudia Marcus (1986). Prolog Programming: Applications for Database Systems, Expert Systems, and Natural Language Systems. Reading, MA et. al.

[MASTER ProLog] Web: http://www.itmasters.com/Products/MP/index .html.

[Melton and Simon 93] J. Melton and A. R. Simon (1993). Understanding the New SQL: A Complete Guide. San Francisco.

[Murray et al. 98] Norman Murray, Carole Goble, Norman W. Paton (1998). A Framework for Describing Visual Interfaces to Databases. Department of Computer Science, University of Manchester. Web: http://citeseer.nj.nec.com/116696.html.

[NLBean] NLBean Natural Language Database Interface. By Mark Watson Associates. Web: http://www.markwatson.com/opensource/opensource.htm.

[Oviatt and Cohen 91] S. Oviatt and P. R. Cohen (1991). The Contributing Influence of Speech and Interaction on Human Discourse Patterns. In: Intelligent User Interfaces. J. W. Sullivan and S. W. Tyler (eds.). New York. 69-83.

[Parlance Corp.] Web: http://www.genuity.com/announcements/news/press_release_19961205-01.xml.

[Q&A FAQ] Q&A Frequently Asked Questions. Web: http://www.quickanswer.com/faqs.htm.

[Rechenberg und Pomberger 99] Peter Rechenberg und Gustav Pomberger (Hg.) (1999). Informatik-Handbuch. München, Wien. (2., aktualisierte und erweiterte Auflage)

[Resnik 89] P. Resnik (1989). Access to Multiple Underlying Systems in JANUS. BBN report 7142. Bolt Beranek and Newman Inc. Cambridge, Massachusetts. September 1989.

[Saelee et al.] M. Saelee, S. Beitzel, E Jensen, D. Grossman, O. Frieder. Intranet Mediators: A Prototype. Information Retrieval Laboratory, Department of Computer Science, Illinois Institute of Technology. Web: http://www.cs.uregina.ca/Research/techreport.html.

[Sanamrad and Bretan 92] Mohammad A. Sanamrad and Ivan Bretan (1992). IBM SAA LanguageAccess: A Large-Scale Commercial Product Implemented in Prolog. In: Proceedings of the 1st International Conference on the Practical Application of Prolog, 1992.

[Scha 77] R. J. H. Scha (1977). Philips Question Answering System PHILIQA1. In: SIGART Newsletter. No. 61, February 1977. New York.

[Snyder 00] Wayne Snyder (2000). English Query from the Beginning. In: SQL Server Magazine. June 2000. Web: http://www.sqlmag.com/Articles/Index.cfm?ArticleID=8537.

[Templeton and Burger 83] M. Templeton and J. Burger (1983). Problems in Natural Language Interface to DBMS with Examples from EUFID. In: Proceedings of the 1st Conference on Applied Natural Language Processing 1983. Santa Monica, CA. 3-16.

[Thompson and Thompson 85] B. H. Thompson and F. B. Thompson (1985). ASK is Transportable in Half a Dozen Ways. In: ACM Transactions on Office Information Systems. 3(2), April 1985. 185-203.

[Warren and Pereira 82] D. Warren and F. Pereira (1982). An Efficient Easily Adaptable System for Interpreting Natural Language Queries. In: Computational Linguistics. 8(3-4), July-December 1982. 110-122.

[Waltz 78] D. Waltz (1978). An English Language Question Answering System for a Large Relational Database. In: Communications of the ACM. Vol. 13, no. 10, 1978. 526-539.

[Weischedel 89] R. Weischedel (1989). A Hybrid Approach to Representation in the JANUS Natural Language Processor. In: Proceedings of the 27th Annual Meeting of ACL 1989. Vancouver, British Columbia. 116-123.

[Wilson et al. 91] M. D. Wilson, D. Sedlock, J.-L. Binot, P. Falzon (1991). An Architecture for Multimodal Dialogue. In: M. M. Taylor, F. Neel, D. G. Bouwhuis (eds.): Proceedings of the 2nd Vencona

Workshop on Multi-Modal Dialogue, Italy, 1991. Web: http://citeseer.nj.nec.com/wilson91architecture.html.

[Winiwarter 94] Werner Winiwarter (1994). The Integrated Deductive Approach to Natural Language Interfaces. Dissertation. Institute of Applied Computer Science and Information Systems, Department of Information Engineering, School of Social and Economic Sciences, University of Vienna. Elektron.

[Winiwarter 95] Werner Winiwarter (1995). Extended CUG for Free Word Order Languages and its Efficient Implementation within an IDA Architecture. In: Proceedings of the Joint Conference of the 8th Asian Conference on Language, Information and Computation and 2nd Pacific Asia Conference on Formal and Computational Linguistics 1995. Web: http://www.ifs.univie.ac.at/~ww/aclic94.ps.

[Winiwarter 97] Werner Winiwarter (1997). Adaptive Natural Language Interface Design in a DOOD Framework. In: Proceedings of the IPSJ International Symposium on Information Systems and Technologies for Network Society 1997. Fukuoka, Japan. Web: http://www.ifs.univie.ac.at/~ww/ipsj97.ps.

[Winiwarter et al. 96] Werner Winiwarter, Osami Kagawa, Shin'ichi Konomi, Yahiko Kambayashi (1996): Collaborative Hypermedia Education with the VIENA Classroom System. In: Proceedings of the Australasian Conference on Computer Science Education 1996. 613-622. Web: http://www.ifs.univie.ac.at/~ww/acse96.ps.

[Winiwarter et al. 97] Werner Winiwarter, Osami Kagawa, Shin'ichi Konomi, Yahiko Kambayashi (1997): The Use of Natural Language to Ask Questions in a Collaborative Hypermedia Teaching Environment. Web: http://www.ifs.univie.ac.at/~ww/aied97.ps.

[Winiwarter und Tjoa 96] Werner Winiwarter und A Min Tjoa (1996): Ein Vorgehensmodell für die erfolgreiche Entwicklung natürlichsprachlicher Informationssysteme. Web: http://www.ifs.univie.ac.at/~ww/emisa96.ps.

[Woods 73] William A. Woods (1973). Progress in Natural Language Understanding: An Application to Lunar Geology. In: AFIPS Conference Proceedings 42. 441-450.

[Woods et al. 72] William A. Woods, Ronald M. Kaplan, Bonnie Nash-Webber (1972). The Lunar Sciences Natural Language Information System: Final Report. BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

[Wu et al. 96] Xu Wu, Tadao Ichikawa, Nick Cercone (1996). Knowledge-Base Assisted Database Retrieval Systems. Singapore.

9 Appendix B: Materialien zur EQ-Beispielanwendung

9.1 Diagramm der Datenbank

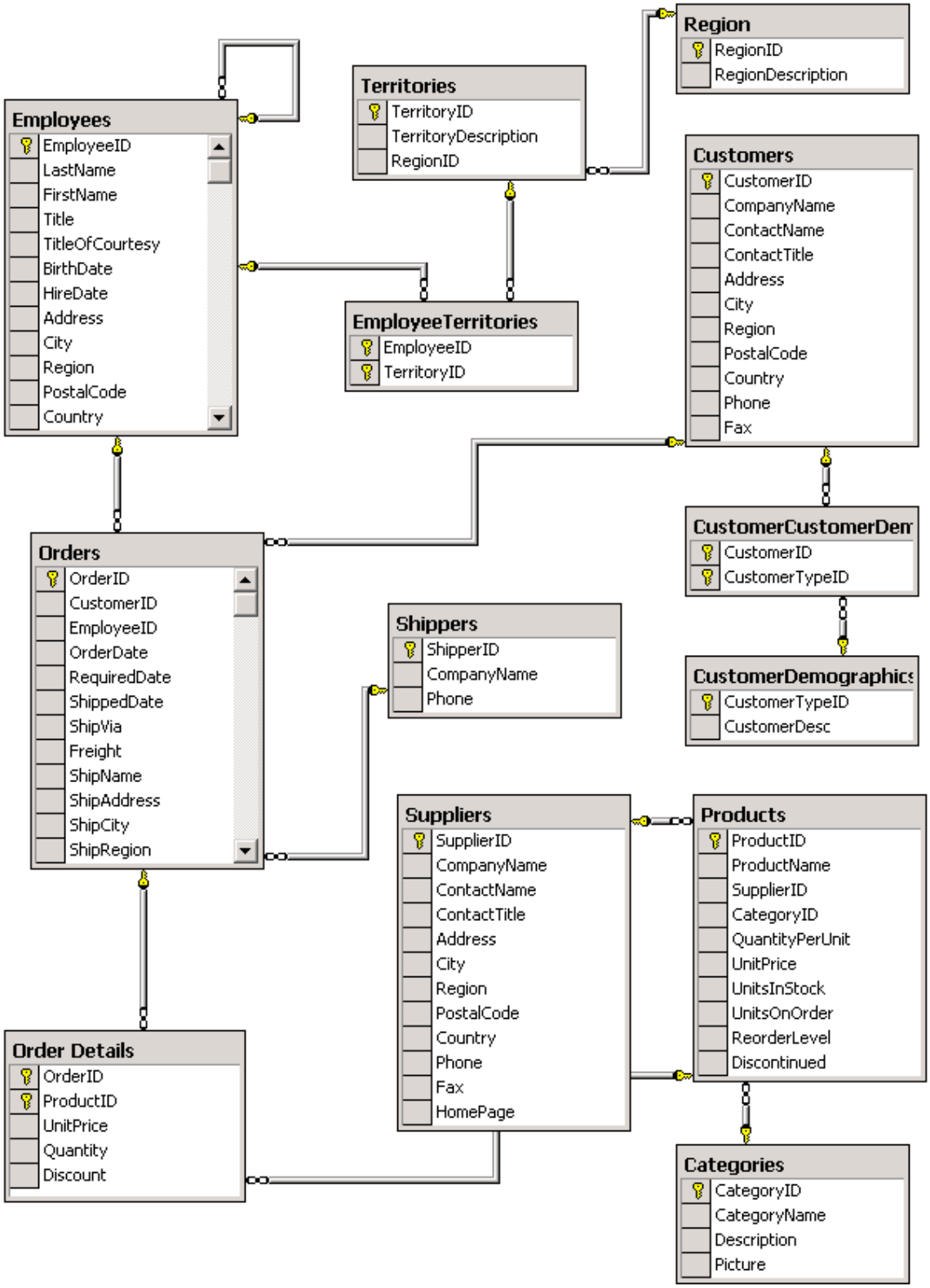


Abbildung 22 - Diagramm der SQL-Server-Datenbank

Bildlegende:
- Beziehungen zwischen Relationen werden durch folgende Elemente beschrieben:
- Linien zwischen Relationen symbolisieren Fremdschlüssel-Beziehungen. Falls beide Endpunkte einer Linie an derselben Relation angebunden sind, handelt es sich um eine reflexive Beziehung. Eine reflexive Beziehung drückt eine Beziehung zwischen Kolonnen/Attributen innerhalb einer Relation aus.
- Feste Linien (im Gegensatz zu gepunkteten Linien) bedeuten, dass das DBMS Referenz-Integrität erzwingt.
- Die Endpunkte der Linien geben an, ob es sich um eine „one-to-many"-Beziehung (Schlüsselsymbol - Unendlichkeitssymbol) oder um eine „one-to-one"-Beziehung (Schlüsselsymbol - Schlüsselsymbol) handelt.

9.2 Beispielanfragen

Die folgenden Beispiele von Anfragesitzungen sollen die Fähigkeiten von ENGLISH QUERY (EQ) etwas ausführlicher dokumentieren, als dies im Rahmen von Kapitel 6.4.1 möglich ist.

- **Aktiv-Passiv-Konvertierung**:

```
> Northwoods Cranberry Sauce is sold by whom?
[Paraphrase] Who sells Northwoods Cranberry Sauce?
```

(Kommentar: Verb-Phrasierungen werden prinzipiell in der Aktiv-Form definiert. EQ nimmt die Aktiv-Passiv-Konvertierung intern automatisch vor.)

- **Relativsatz-Anbindung**:

(Syntaktische ambig)

```
> Show me the suppliers which sell products which are named Bigfoot
Breweries.
[Paraphrase] Which suppliers sell products named Bigfoot Breweries?
```

(Syntaktisch eindeutig)

```
> Show me the suppliers which sell a product which are named Bigfoot
Breweries.
[Paraphrase] Which suppliers whose name is Bigfoot Breweries sell a
product?
```

- **Konjunktion und Disjunktion**:

```
> Show the products named both Chai and Chang.
[Paraphrase] List the products whose name is Chai or whose name is
Chang.
```

(Kommentar: Nicht in jedem Fall lässt sich mit „both – and" die konjunktionale Lesart (und aufgrund der Verletzung der Konsistenzregeln eine entsprechende Fehlermeldung) erzwingen.)

```
> Show the products whose categories are both Beverages and Seafood.
[Paraphrase] List the beverageses and list the seafood.
```

(Kommentar: Interessant ist in diesem Fall, dass EQ zwei separate Ergebnis-Tabellen bildet – eine für „beverageses“ und eine für „seafood“.)

```
> Show the products whose categories are Beverages and whose
categories are Seafood.
[Paraphrase] List the Beveragess [sic] whose category is Seafood.
[SQL] SELECT null as „ProductName“, null as „ProductID“, null as
„UnitPrice“, null as „CategoryName“ WHERE 1=0 ORDER BY 1
```

(Kommentar: Das Resultat ist in diesem Fall der NULL-Wert. Produkte der Kategorie „Beverages“ können nicht gleichzeitig der Kategorie „Seafood“ angehören. Interessanterweise liefert EQ hier keine Fehlermeldung bezüglich Verletzung der Konsistenz. Die *Analysis*-Ansicht der Modell-Test-Konsole listet zwar die verwendeten Entitäten und Phrasings auf, und die Paraphrase wird korrekt erstellt, die SQL-Anfrage liefert jedoch unter keinen Umständen ein Ergebnis. Das Problem muss entsprechend bei der Abbildung der logischen Repräsentation der Anfrage auf Datenbank-Objekte liegen.)

- **Nominalkomposita**:

```
> Show the product customers.
[Paraphrase] Which customers order products?
```

(Kommentar: Aus Entitäten des semantischen Modells lassen sich Komposita kreieren, die EQ folgendermassen interpretiert: Das letzte Wort des Kompositum wird als Grundwort, und die vorhergehenden Wörter werden als Bestimmungswörter interpretiert. Die „Bedeutung“ der Komposition wird aus der semantischen Beziehung zwischen den beteiligten Entitäten ermittelt.)

```
> Show the customer Alfreds Futterkiste orders.
[Paraphrase] Which orders is Alfreds Futterkiste the customer of?
```

(Kommentar: Der Nominalkomposita-Mechanismus lässt sich, falls man die Beziehungen zwischen den beteiligten Entitäten kennt, zur effizienten Formulierung von Anfragen verwenden.)

```
> Show the customer territories.
[Paraphrase] Show the customers that order products since fiscal 1997
and their territories.
[Antwort] Customers don’t have territories. Employees have
territories.
```

(Kommentar: Da keine Beziehung zwischen „customer“ und „territory“ definiert ist, zeigt EQ eine Fehlermeldung an. Diese ist insofern informativ/kooperativ, als der Grund für das Fehlschlagen der Generierung einer SQL-Anfrage angegeben und zusätzlich ein Hinweis zur konzeptuellen Abdeckung des Systems, d.h. ein Hinweis bezüglich semantischer Beziehungen, an denen das Grundwort „territory“ teilnimmt, gemacht wird.)

- **Telegraphische Äusserungen**:

```
> Product.
[Paraphrase] List the products.

> Orders, customer, Alfreds Futterkiste.
[Paraphrase] Show the orders and their customers named Alfreds
Futterkiste.
```

(Kommentar: EQ kommt auch mit syntaktisch unvollständigen bzw. sogar mit Entitätenlisten „zurecht“, was sich u.U. Effizienz-steigernd auswirkt.)

- **Arithmetische Operationen**:

```
> 5 + 8 * 3
[Antwort] 29.

> Employee ages + 100.
[Paraphrase] Show the employees and their ages.
```

(Kommentar: EQ kann sowohl reine arithmetische Ausdrücke als auch gemischte Ausdrücke evaluieren. Im zweiten Beispiel entspricht jeder Wert der Kolonne „Age plus 100“ (der Ergebnis-Tabelle) dem um 100 erhöhten Wert der Ausgangskolonne „Age“.)

- **Aggregatfunktionen**:

```
> What is the average age of the employees?
[Paraphrase] What is the average employee age?
```

(Kommentar: Aggregatfunktionen wie SUM, AVERAGE, COUNT, MINIMUM und MAXIMUM werden von EQ unterstützt.)

- **Funktionen zum Sortieren und Gruppieren**:

```
> Order the employees by (inverse) name.
[Paraphrase] List the employees by (reverse) alphabetical order.
```

(Kommentar: Sowohl ORDER BY (ASC) als auch ORDER BY (DESC) wird unterstützt. In diesem Fall stellt der „employee name“ das Kriterium für die alphabetische Sortierung dar.)

```
> List the number of employees for each region.
[Paraphrase] Show the regions and how many employees belong to
territories that have them.
```

(Kommentar: Auf diese Weise lässt sich ein GROUP BY realisieren. Die Ergebnis-Tabelle enthält in diesem Fall pro „region“ ein Tupel, dessen Wert des Attributs „count“ die Anzahl „employees“ in dieser „region“ angibt.)

- **„Why“-Fragen**:

```
> List the products that have been discontinued.
[NL-Resultat] The products that have been discontinued are:
[...]
> Why?
[Paraphrase] How do I know the products that have been discontinued
are: ?
[Antwort] The information was in the database.
> Why?
[Paraphrase] How do I know?
[Antwort] I don’t know how I know.
> How do you know?
[Paraphrase] How do I know?
[Antwort] This is getting silly.
```

(Kommentar: „Why“-Fragen werden von EQ im Sinne von „Weshalb/Woher weiss ich, dass die Sachlage so und so ist?“, d.h. als Satzellipse zu „Why/How do you know xy?“, interpretiert. Mit der Zeit reagiert EQ allerdings „ungehalten“.)

- **Anfragen mit temporalen Verweisen**:

```
> Which employees will be older than 50 last year?
[Paraphrase] Which employees will be more than 50 years old in fiscal
2001?
```

(Kommentar: EQ ignoriert das im Zusammenhang mit dem Temporaladverb „last year“ ungrammatikalische Tempus (Futur I) und interpretiert den relativen zeitlichen Ausdruck „last year“ im Bezug auf die Systemzeit der Anfragezeitpunktes (als Steuerjahr-Angabe, gemäss den Projekt-Einstellungen). Im Gegensatz zu NLIDBs zu echten temporalen Datenbanken [Androutsopoulos 96] berücksichtigt EQ die semantischen Implikationen der verschiedenen Tempusformen nicht.)

- **Schlussfolgern aus dynamisch eingefügtem Wissen**:

```
> Are you intelligent?
[Antwort] Based on the information I've been given about this
database, I can't answer: „Am I intelligent?“. I haven't been given
any information on intelligentness.
> You are intelligent if you're like Sokrates.
[Antwort] Ok. I'm intelligent if and only if I'm there like Sokrates.
> You're not like Sokrates.
[Antwort] Ok. I'm not there like Sokrates.
> Are you intelligent?
[Paraphrase] Am I intelligent?
[Antwort] No, I'm not intelligent.
> Why?
[Paraphrase] How do I know I'm not intelligent?
[Antwort]      1. I'm not there like Sokrates.
               2. If I'm intelligent then I'm there like Sokrates.
                     1, 2 =>
               3. I'm not intelligent.
```

(Kommentar: Ein klassischer Modus Tollens. Vgl. Titelblatt.)