# Verbalisierung von Datenbanktransaktionen

Olaf Th. Buck
Fa. Peter Pietsch, Organisationsberatung und Informationstechnologie
Grosse Burgstrasse 55-57
D-23552 Lübeck
Email: olaf.buck@pietsch-luebeck.de
http://www.pietsch-luebeck.de

Volker Linnemann
Institut für Informationssysteme
Universität zu Lübeck
Osterweide 8
D-23562 Lübeck
Email: linnemann@ifis.uni-luebeck.de
http://www.ifis.uni-luebeck.de

**Abstract:** Datenbanktransaktionen bestehen in der Regel aus mehreren Datenbankanweisungen und haben vielfältige Veränderungen des Datenbestandes zur Folge. Diese Veränderungen sind in der Regel für einen Anwender, der die Details der Transaktion nicht kennt, nicht direkt nachvollziehbar. Daher ist es wünschenswert, wenn dem Anwender eine natürlichsprachliche Erklärung der Auswirkung an die Hand gegeben werden kann, mit Hilfe derer er die Transaktion auf einem hohen Abstraktionsniveau nachvollziehen kann. Darüber hinaus ist es zur Erfüllung von Rechtsvorschriften, wie zum Beispiel für die Sicherstellung von Revisionssicherheit, notwendig, Strategien und Funktionen zur Verbalisierung von Datenbanktransaktionen zu entwickeln. Es wird in dieser Arbeit gezeigt, wie Transaktionen in einem geeigneten Format aufgezeichnet und mit Hilfe von Schablonen in natürliche Sprache übersetzt werden können. Abschließend wird ein System zur Realisierung dieser Funktionen vom Entwurf bis zur Implementierung beschrieben. Dieses wird zur Zeit bei mehreren Anwendern erfolgreich eingesetzt.

## 1 Einleitung

Benutzer eines Systems mit Datenbankunterstützung navigieren selten direkt auf der Datenbank durch Verwendung einer Anfragesprache wie SQL. In der Regel ist eine Applikation mit statischen Eingabemasken oder ähnlichem vorhanden, in deren Quellcode die entsprechenden Anfragen gekapselt sind. So werden Änderungen in der Datenbank und deren Auswirkungen für Benutzer des Systems wenig transparent. Dies wird auch dadurch verstärkt, dass im Allgemeinen Benutzer nicht exklusiv mit einer Datenbank arbeiten, sondern diese mehreren Benutzern gleichzeitig zur Verfügung steht. Darüber hinaus sind Datenbankschemata selten statisch, sondern sie verhalten sich gemäß des Softwarelebenszyklus von darauf operierenden Applikationen dynamisch, wobei sich das Aussehen beispielsweise von Eingabemasken nicht zwingend verändern muss. Veränderungen des Schemas können nicht nur durch veränderte Anforderungen,

sondern auch zum Beispiel durch Refactoring oder Optimierung auftreten. Dies führt zu einer weiteren Verminderung der Transparenz. In dieser Arbeit soll aufgezeigt werden, wie durch Verwendung natürlichsprachlicher Ausdrücke Änderungen in einer Datenbank wieder transparent gemacht werden können. Eine grundlegende Anforderung an das zu realisierende System war, dass auch für Nichtexperten die Änderungen in Datenbeständen verständlich sind, und in einer „Historie“ für jeden Datensatz angezeigt werden können. Hierfür werden durchgeführte Datenbanktransaktionen aufgezeichnet und in grammatisch und im Kontext korrekte, verbale Ausdrücke übersetzt. Ein Beispiel eines solchen generierten natürlichsprachlichen Ausdrucks ist der folgende:

*Die Rufnummer des Telefonbucheintrags mit dem Namen „Herr Müller“ wurde von 0451 654321 in 0451 123456 geändert.*

Ein entsprechendes System zur Aufzeichnung von Datenbanktransaktionen und anschliessenden Übersetzung in natürlichsprachliche Ausdrücke wurde im Rahmen einer Diplomarbeit realisiert, auf welcher diese Arbeit basiert [Bu02]. Die Diplomarbeit am Institut für Informationssysteme der Universität zu Lübeck wurde bei der Fa. Pietsch - Organisationsberatung und Informationstechnologie - durchgeführt. Bei der Fa. Pietsch handelt es sich um ein Lübecker Unternehmen, tätig in der Beratung und der Systementwicklung für die Bereiche des Umweltschutzes, der Arbeitssicherheit, des vorbeugenden Brandschutzes sowie des Liegenschaftsmanagements. Das Unternehmen versteht sich aufgrund der konsequenten Spezialisierung im Gesundheitswesen als Dienstleister für das Krankenhaus. Um optimierte Lösungen krankenhausspezifischer Aufgabenstellungen vornehmlich für den Bereich der Aufbau- und Ablauforganisation verwirklichen zu können, wurden spezielle Managementsysteme durch die Fa. Pietsch projektiert und entwickelt, die sich unter anderem auch in mehreren Universitätskliniken im Einsatz befinden. Das hier entwickelte System wurde nach Durchführung der Diplomarbeit weiterentwickelt, und ist seitdem ebenfalls in mehreren Universitätskliniken deutschlandweit im Einsatz.

Eine weitere wesentliche Anforderung an das System war, dass die Performanz bestehender Softwarekomponenten nicht wesentlich beeinträchtigt werden durfte. Zur Darstellung eines Verlaufs von Feldwerten wurde in einem Dialog den Benutzern das Verfolgen der generierten lesbaren Texte ermöglicht.

## 2 Aufzeichnung von Transaktionen

Bevor eine Übersetzung von Datenbanktransaktionen in natürliche Sprache vorgenommen werden kann, müssen diese in einem dafür geeigneten Format aufgezeichnet werden. Alternativ könnte dies auch direkt nach Durchführung der Transaktion geschehen, indem die entsprechenden Daten an eine Systemkomponente übergeben werden. Dies ist jedoch aufgrund der Anforderungen an die Performance und die Flexibilität sowie Wartbarkeit des Systems wenig vorteilhaft.

Änderungen in einer Datenbank geschehen stets durch eine einer sequentiellen Abfolge äquivalenten Abarbeitung von schreibenden Transaktionen. Diese werden üblicherweise in einem sogenannten Logfile protokolliert. Konzeptuell ist dies in der Regel eine herkömmliche Tabelle, an die sequentiell entsprechend der zeitlichen Abfolge Einträge

angehängt werden. Dazu notwendige Algorithmen und Systeme zum Management werden in der Literatur, wie zum Beispiel [GR93] oder [HR01], beschrieben. Dabei sind die Inhalte der Einträge im Logfile abhängig von den Anforderungen an die Transaktionsaufzeichnung. Auf die Aufzeichnung lesender Operationen kann für die Verbalisierung verzichtet werden, da diese keine Änderungen in der Datenbank verursachen. Für schreibende Transaktionen werden hier stets sämtliche durchgeführten Änderungen an der Datenbank aufgezeichnet, wobei zwischen den drei grundlegenden Zurgriffsoperationen APPEND (Neuanlage von Datensätzen/Feldern), CHANGE (Verändern von Datensätzen/Feldern) und DELETE (Löschen von Datensätzen) unterschieden wird. Ein Logfile, mit dem diese Anforderungen für relationale Datenbanken realisiert werden können, sollte aus einer Abfolge von folgenden Tupeln bestehen: Transaktion (Eindeutiger Schlüssel für Zusammengehörigkeit von Operationen), Ursprung (Zeigt Urheberschaft der Operation auf, z.B. Zeitpunkt und Benutzer), Tabelle, Datensatz, Spalte, Alter Wert und Neuer Wert. Jedes Tupel identifiziert hierbei eine einzelne, elementare Zugriffsoperation, die zu einer Feldänderung geführt hat. Jedes Feld einer relationalen Datenbank kann hierbei durch seine Position (Spalte,Datensatz,Tabelle) eindeutig identifiziert werden. Zur Darstellung werden die elementaren Zugriffsoperationen READ(field) und WRITE(field,value) verwendet, welche den Wert eines Feldes lesen bzw. schreiben (vgl. hierzu auch [Vos00]).

## 3 Generierung natürlichsprachlicher Übersetzungen

In der Computerlinguistik ist ein grundlegendes systematisches Vorgehen die Unterscheidung zwischen Inhaltsbestimmung (deciding what to say) und einer anschließenden Formbestimmung (deciding how to say it). Diese Vorgehensweise wurde auch hier gewählt.

Eine wesentliche Aufgabe bei der Inhaltsbestimmung zur Sprachgenerierung ist die Wortwahl. Dazu muss für ein Konzept der internen Repräsentation ein passendes Lexem ausgewählt werden. Als Lexem wird hier eine lexikalische Bedeutungseinheit bezeichnet, die eine fassbare Reflexion eines zugehörigen Gedankens bezeichnet. Diese Begrifflichkeit ist bei verschiedenen Autoren jedoch teilweise anders belegt[1], in jedem Fall haben Lexeme jedoch eine eindeutige sprachliche Identität. Zur Bestimmung des Inhaltes wurde für relationale Datenbanken das E/R-Modell nach Peter Chen gemäß [BCN91] zugrundegelegt. Sowohl in der Modellierungsphase, als auch in der schematischen Abbildung auf ein DBMS werden Bezeichner zur eindeutigen Identifizierung von Elementen wie Attributen, Entitäten oder Tabellen verwendet. Ein konkretes Beispiel wäre der Primärschlüssel einer Tabelle. Diese Bezeichner sollten einen klaren Ausdruck über den Inhalt des Elementes vermitteln, sind allerdings nicht notwendigerweise von der Zielsprache der Verbalisierung belegt. In der Praxis werden oft Abkürzungen oder aber auch Worte anderer Sprachen verwendet, z.B. Anglizismen wie User für den deutschen Begriff Benutzer. Das folgende Beispiel veranschaulicht dieses:

[1] Vielfach werden Lexeme auch als Spezialisierungen von Wortarten betrachtet.

*Der eindeutige Bezeichner PERS_TBL, der eine Tabelle bezeichnet, liefert eine gewisse Information über die Funktion dieses Elementes des Datenbankschemas. Er ist aber sprachlich nicht eindeutig belegt, und somit ist PERS_TBL für eine Übersetzung ungeeignet.*

Wie im Abschnitt 2 beschrieben, kann man Felder in relationalen Datenbanken durch Tabellen, Spalten und Zeilen eindeutig identifizieren. Für diese drei Elemente sollen nun eindeutige Lexeme gefunden werden. Diese „Namen" sollen weiterhin nicht notwendigerweise mit den Bezeichnern des Schemas oder der Modellierung übereinstimmen müssen. Durch die Verwendung von Lexemen ist gesichert, dass diese auch für eine Übersetzung weiterverwendet werden können, da jedes eine eindeutige sprachliche Identität besitzt. Lexeme für Tabellen sollen hier als $L_x$ definiert werden. Ein Lexem $L_x$ für eine Spalte ist analog dem Bezeichner für die Tabelle definiert. Ein Feld kann eindeutig identifiziert werden, wenn die zugehörige Tabelle $T_z$, Zeile $R_y$ und Spalte $C_x$ bekannt sind. Sind eindeutige Bezeichner hierfür bekannt, so kann die Position eines Feldes auch durch diese eindeutig bestimmt werden, z.B.:

*Die Rufnummer des Telefonbucheintrages mit dem Namen „Herr Müller" ist 12345.*

Die sequentielle Abfolge von Datensätzen (Zeilen) in einer Tabelle wird gewöhnlich durch eine Sortierung mit Indizes bestimmt. Ein Index wird über eine oder mehrere Spalten (Attribute) einer Tabelle definiert und enthält die entsprechenden Feldwerte. Verfügt eine Tabelle über einen eindeutigen, dichten[2] Index, so kann jeder Datensatz anhand des Indexeintrages eindeutig identifiziert werden [Vos00]. Solch ein eindeutiger, dichter Index kann zum Beispiel der Primärschlüssel einer Relation sein. Das Finden eines Bezeichners für Zeilen in Form eines eindeutigen Lexems ist aber aufgrund der Variabilität der Indexeinträge nicht möglich. Ein eindeutiger Bezeichner kann aber für die zugrundeliegenden Spalten analog zu den existierenden Lexemen für die Einzelspalten gefunden werden. Dieser eindeutige Bezeichner $L_y$ wird mit dem Indexeintrag kombiniert, welcher sich aus den zugehörigen Feldwerten zusammensetzt. Eindeutige Bezeichner können ebenso in objektorientierten Datenbanken gefunden werden.

Ein weiterer Teil der Inhaltsbestimmung ist neben der Ermittlung lexikalischer Inhalte die Identifizierung von pragmatischen Zusammenhängen. Zur Bestimmung von Beziehungen und Inhalten in einer Datenbank sollen daher klar definierte Kontexte verwendet werden. Ein Kontext beschreibt einen Zusammenhang zwischen verschiedenen Feldern oder Änderungen innerhalb einer Datenbank. Ein Kontext kann abhängig von sämtlichen Parametern sein, die sich in einer Aufzeichnung wie z.B. einem Logfile finden lassen. Der zugehörige Teil des Übersetzungssystems muss also in der Lage sein, Kontexte anhand der Einträge des Logfiles zu identifizieren. Dies kann durch den Vergleich mit vorher abgelegten Mustern für jeden Kontext (bestehend z.B. aus Feld, Tabelle, Wert, ...) geschehen. Für eine praktische Umsetzung würde sich bei einer großen Anzahl von Kontexten eine Indizierung der Datenstruktur anbieten, um diese effektiv auffinden zu können.

---

[2] In einem dichten (dense) Index findet sich im Gegensatz zu einem dünnen (sparse) Index für jeden Wert der betreffenden Attribute ein Eintrag.

Datenbanktransaktionen setzen sich aus mehreren Einzeloperationen zusammen (vgl. Abschnitt 2). Somit kann eine Transaktion durch Übersetzung der zugehörigen Einzeloperationen in natürliche Sprache übertragen werden.

Durch Generierung von natürlichsprachlichen Übersetzungen für jeden dadurch entstehenden neuen Kontext kann dann eine Historie der zugehörigen Datensätze respektive Feldwerte aufgebaut werden. Die so generierten Beschreibungen können in einer eigenen Tabelle oder anderen Form mit den zugehörigen Bezügen abgelegt werden. Von dort sind diese dann bei Bedarf durch einfache Anfragen oder Einschränkungen entsprechend der zugehörigen Elemente des Datenbankschemas abrufbar.

Neben der Inhaltsbestimmung ist eine weitere Aufgabe von Sprachgenerierung die Erzeugung von Kennzeichnungen zur Repräsentation von sprachlichen Elementen. Hier soll die deutsche Sprache verwendet werden, da das realisierte System ebenfalls deutschsprachig ist. Die im folgenden vorgestellten Verfahren können aber auch für alle Sprachen verwendet werden, die über eine feste Form und Grammatik verfügen.

Durch die Auswahl von exemplarischen Beispielen für den jeweiligen Kontext kann eine Satzstruktur durch eine Syntaxanalyse abgeleitet werden. Die syntaktische Struktur der Beispielsätze wird dann in eine Abfolge von Symbolen zerlegt, welche jeweils einzelne lexikalische Elemente repräsentieren. Diese können dann bei der eigentlichen Generierung durch Ersetzungsregeln durch die zugehörigen natürlichsprachlichen Ausdrücke ersetzt werden. Aus diesen Symbolen lassen sich dann für jeden Kontext Schablonen zur Generierung von grammatisch korrekten Sätzen erzeugen. Kontexte in Datenbanken können zum Beispiel Änderung, Neueinträge oder Löschungen sein. Zur Beschreibung eines Kontextes werden neben den zugehörigen Bezeichnern für die Position des Feldes, auf das sich die Beschreibung bezieht, auch Werte für dieses Feld benötigt. Eine Übersetzung wird daher auf Grundlage mehrerer Kontexte gebildet. Jedem dieser Kontexte ist eine Schablone zugeordnet, die sich aus einer Sequenz von Symbolen zusammensetzt, welche passende Übersetzungsregeln repräsentieren.

Schablonen, feststehende Lexeme und Ersetzungsregeln eines Generators zur Erzeugung von Übersetzungen können in einer geeigneten Datenstruktur, wie einer Tabelle, gespeichert werden. Wortbildungsfunktionen wie zum Beispiel die Deklinationsfunktion oder die Bildung von Artikeln können auch, je nach Sprache, durch Verwendung von hierfür geeigneten Algorithmen gebildet werden. Das Auffinden solcher Formalismen ist in den Bereichen der Linguistik bzw. Computerlinguistik Gegenstand aktueller Forschungen. Die Entscheidung, ob Wortbildungsfunktionen in einer Datenstruktur abgelegt werden, die dann entsprechend interpretiert wird, oder durch geeignete einfache Funktionen bzw. Algorithmen in ein Übersetzungssystem fest integriert werden, sollte je nach Umfang und Komplexität der zu generierenden Wortformen getroffen werden. Dies ist vor allem auch abhängig von der Differenziertheit der Wortbildungsmöglichkeiten der Zielsprache. Verfügt eine Sprache zum Beispiel über sehr viele unregelmäßig gebildete Wortformen, so ist die zusätzliche Verwaltung eines entsprechenden Wörterbuches neben einem Regelwerk unerlässlich.

Die Schaffung eines komplexen Generatorsystems für grammatische Formen ist für eine geringe Anzahl von Kontexten und verwendeten Lexemen sicher nicht gerechtfertigt. Stattdessen können Lexeme in einem einheitlichen Wörterbuch verwaltet werden. Beim Hinzufügen neuer Einträge sollten grammatische Formen, wie zum Beispiel die Deklination, durch grundlegende Regeln basierend auf elementaren Eigenschaften wie Genus oder Numerus gebildet und anschließend durch Benutzer des Systems überprüft und eventuell korrigiert werden. Die Verwendung von Schablonen stellt eine einfach zu realisierende Lösung dar, die ebenfalls zu einer sehr guten Performance des Gesamtsystems führt. Einzelne Elemente wie Symbolfolgen werden hierbei, ähnlich einzelner Ansätzen von Expertensystemen, jedoch mehrfach verwendet oder gespeichert. Durch die Verwendung von Formalismen, wie zum Beispiel Produktionsregeln, ließe sich dies vermeiden. Ein anderer Ansatz als die direkte Verwendung von vorher abgelegten Schablonen wäre also der Aufbau von Schablonen mit Hilfe einer regulären Grammatik, wobei noch zu überprüfen wäre, inwieweit dies auch bei Expertensystemen Verwendung finden kann. Hierzu soll dann anstatt der gesamten Schablone eine entsprechende Startproduktion entsprechend des Kontextes gewählt werden. Die Produktionsregeln von Grammatiken erzeugen dann die fest determinierte Satzstruktur, welche die Positionen einzelner Elemente strikt festlegt. Terminalsymbole entsprechen den Symbolen der Schablonen, wie sie bereits festgelegt wurden. Die zugehörigen Ersetzungsregeln füllen diese Symbole dann mit lexikalischem Material. Nonterminalsymbole können zum Teil den syntaktischen Elementen des zu generierenden Satzes entsprechen.

Für die Verbalisierung „gewöhnlicher“ Datenbanktransaktionen, auch mit einigen Sonderfällen, ist die direkte Verwendung von Schablonen ausreichend. Diese können dann zum Beispiel parallel zu den Mustern zur Kontexterkennung abgelegt werden. Da innerhalb größerer Anwendungssysteme eine große Zahl von Transaktionen durchgeführt wird, liegt in der Verwendung von Schablonen nicht nur ein Vorteil für die Performance des Gesamtsystems, sondern es resultiert auch eine bessere Wartbarkeit und Wiederverwendbarkeit. Bei der Verwendung von Grammatikformalismen zum Aufbau von Schablonen sollte dann stattdessen eine Speicherung der zugehörigen Startsymbole und der entsprechenden Produktionen erfolgen.

Ein Kriterium, das die allgemeine Verständlichkeit der generierten Sätze wesentlich beeinflusst, sind die Inhalte der Feldwerte. Bestehen diese aus Zeichenketten, wie zum Beispiel Eigennamen, so ist dies unkritisch zu betrachten. Auch Zahlenwerte wie zum Beispiel Währungsbeträge können in einen sinnvollen Zusammenhang gebracht werden. Repräsentiert ein Feldwert jedoch einen Schlüssel aus einer anderen Tabelle so ist eine für den Menschen allgemeinverständliche Übersetzung nicht direkt möglich.

*Der Titel der Person mit dem Namen „Herr Müller“ wurde von 1 in 2 geändert.*

In Datenbanken können verschiedene Tabellen durch Einführen von Fremdschlüsseln miteinander verknüpft werden. Diese werden durch einen eindeutigen, dichten Index geordnet und können auch den Primärschlüssel repräsentieren. Diese Werte können dann in ein entsprechendes Feld des gleichen Typs einer anderen Tabelle übernommen werden, um entsprechende Verknüpfungen abzubilden. Werden in den

Datenbanktabellen Fremdschlüssel verwendet, so können diese durch den entsprechenden Primärschlüssel der zugehörigen Verknüpfungstabelle ersetzt werden:

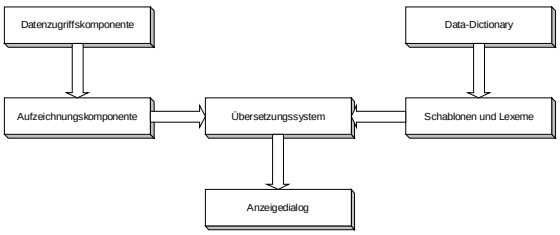
*Die Rufnummer des Telefonbucheintrages mit der Person 4 wurde von 4 in 3 geändert.*

Dieser generierte Text ist zwar für den zugehörigen Kontext korrekt, verfügt aber über eine geringe Verständlichkeit. Ersetzt man nun sämtliche Einträge von Feldwerten, die Fremdschlüssel von anderen Tabellen repräsentieren, durch den zugehörigen Datensatzbezeichner der Fremdtabelle, so wird die Verständlichkeit wesentlich erhöht:

*Die Rufnummer des Telefonbucheintrages mit dem Namen „Herr Meier“ wurde von 0451 987654 in 0451 456789 geändert.*

# 4 Implementierung und Performance-Aussagen

Die Implementierung des Systems erfolgte objektorientiert unter Verwendung der Borland Delphi IDE unter Microsoft® Windows NT / 2000. Dieses setzt sich aus drei wesentlichen Komponenten zusammen: Die Erweiterung eines bestehenden Data-Dictionaries zur Verwaltung von Lexemen und Patterns, eine Komponente zur Protokollierung abgeschlossener Datenbanktransaktionen, eine Serverapplikation[3] zur Generierung der Übersetzungen und ein Anzeigedialog für die übersetzten Einträge



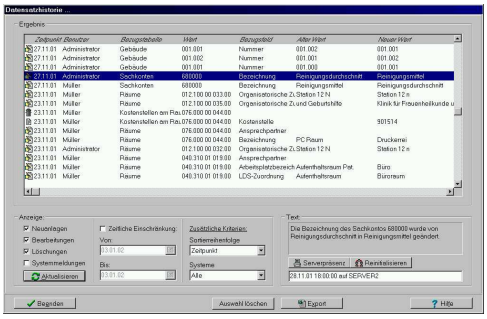

Bei der Erweiterung des bestehenden Data-Dictionaries handelte es sich im wesentlichen um die Bereitstellung von Eingabemasken für die benötigten Daten. Hierzu zählen ein Wörterbuch für die Lexeme, die Verknüpfung dieser Lexeme an bestehende Metadaten wie Tabellen oder Spalten sowie eine Möglichkeit zur Verwaltung von Schablonen zur Übersetzungsgenerierung.

Das System verwendet eine clientbasierte Protokollierungskomponente, die unter der Verwendung des Listener-Musters (vgl. hierzu [GHJV96]) auf bestehende Datenbankzugriffskomponenten aufsetzt, ohne diese zu verändern. Erfolgreich durchgeführte Transaktionen, und damit die zugehörigen Feldänderungen, werden in eine sequentiellen Tabelle als Logfile gespeichert. In einer ersten, testweisen Implementierung wurde festgestellt, dass eine Protokollierungskomponente ohne jegliche Zwischenspeicherung erhebliche Performanceverluste zur Folge hatte. Dies lag darin begründet, dass jede einzelne aufgezeichnete Operation mehrere schreibende Operationen für die Eintragung in das Logfile nach sich zog und auf mehreren Clients zu fast identischen Zeitpunkten auftrat. Dadurch wurde das DBMS durch Spitzen in der Anzahl der Zugriffe stark belastet. Dies verlangsamte nicht nur das Antwortverhalten der entsprechenden Einzelapplikation merklich, sondern beeinflusste auch das Antwortverhalten von auf derselben Datenbank operierenden Anwendungen negativ.

[3] Eine Serverapplikation ist eine Anwendung, die jeweils nur einmal auf einem Applikationsserver ausgeführt wird. Diese kann entweder ein direkt lauffähiges Programm sein, oder z.B. ein NT-Dienst oder Unix-Daemon.

Dieses entsprach nicht den Anforderungen. Daher musste eine Möglichkeit zur Zwischenspeicherung des Logfiles vor Ablage in die Datenbank geschaffen werden, welche Leerlaufzeiten der Einzelapplikationen und des DBMS ausnutzt. Hierzu bietet das Konzept des Threads[4] (Leichtgewichtsprozess, vgl. hierzu [So98]) unter Windows NT eine mögliche Lösung. Threads mit der niedrigsten Prioritätsstufe erhalten fast ausschließlich nur bei „Leerlauf“ des Betriebssystems Rechenzeit zugeteilt. Die Protokollierungskomponente wurde daher zweigeteilt: Die eigentliche Aufzeichnungskomponente bleibt als Hauptobjekt in die Applikation eingebunden, während der für die Logfileeintragungen zuständige Teil als diesem zugeordneter Thread gestaltet wird. Beide verwenden einen gemeinsamen Speicherbereich, in dem sich eine Liste mit den zu protokollierenden Änderungen befindet. Diese wird von der Protokollierungskomponente bei Auslösen der entsprechenden Ereignisse gefüllt, während ein Thread diese in das Logfile einträgt. Da dies aufgrund der niedrigen Priorität nur in Leerlaufzeiten der zugehörigen Applikation geschieht, konnte der Performanceverlust so wesentlich verringert werden. Allerdings werden zur Synchronisation weitere Mechanismen benötigt, um Verklemmungen durch gleichzeitigen Zugriff auf die gemeinsame Liste zu verhindern. Hierzu wurde eine Semaphor-ähnliche Lösung zum wechselseitigen Ausschluss verwendet. Die Gesamtsystemkomponente zeigte dann ein akzeptables Laufzeitverhalten.

Zur Verbalisierung werden von der Serverapplikation die im Data-Dictionary definierten Schablonen und Lexeme verwendet. Die verbalisierten Texte werden dann in einer weiteren, separaten Tabelle abgelegt. Die eigentliche Applikation wird durch zwei vertikal angeordnete Klassen aufgebaut. Die obere der Klassen ist ein Formular mit einer Verlaufsansicht zur Überwachung der Applikation. Sie ist ferner für eine zyklische Abfrage der Logfile-Tabelle, die Kontextidentifikation und das Schreiben der übersetzen Einträge zuständig. Die zweite Klasse übernimmt das Übersetzen der Einträge. Übersetzt werden die Kontexte APPEND (Hinzufügen von Datensätzen), CHANGE (Ändern von Feldwerten) und DELETE (Löschen von Datensätzen). Für jeden Kontext existiert eine Schablone, die in entsprechenden Methoden fest codiert ist. Dazu wird ein im Data-Dictionary definiertes Pattern gesucht, welches Ersetzungsregeln für die einzelnen Symbole der Schablonen liefert. Zur Sicherstellung der Revisionssicherheit werden an jeder Übersetzung zusätzlich Benutzer, Uhrzeit und Teilsystem vermerkt, die ursächlich für die Datenänderung waren.




Die mit der Serverapplikation generierten Übersetzungen sollen für Benutzer angezeigt werden. Hierzu wird für diesen Teil der Benutzerschnittstelle ein Standarddialog verwendet, der in einem Anzeigegitter die übersetzten Texte anzeigt. Diese werden durch ein SQL-Statement abgefragt, in dessen „WHERE-Bereich“ sich vom Benutzer festgelegte Bedingungen finden. Angezeigt werden

[4] Analoge Konzepte existieren auch unter anderen Betriebssystemen, wie z.B. Unix.

neben dem generierten Text auch die Bezeichner (Lexeme) von an der Übersetzung beteiligten Feldwerten, sowie der Zeitpunkt. Ferner werden Einschränkungs- und Anordnungsmöglichkeiten geschaffen, die eine bessere Übersichtlichkeit verschaffen sollen. Zum Aufruf dieses Dialoges existieren zwei Methoden: Aufruf ohne Parameter (sämtliche generierten Texte werden abgefragt), sowie Aufruf mit den Parametern „Tabellenname“, „Datensatznummer“ (nur die Einträge der Tabelle und des zugehörigen Datensatzes werden abgefragt). Letzteres stellt die eigentliche *Datensatzhistorie* dar.

## 5 Vergleich mit anderen Arbeiten

Auf die Aufzeichnung von Datenbanktransaktionen in Logfiles wird in der Standardliteratur, wie [Vos00], [GR93] oder [HR01] ausführlich eingegangen. In [GR93] werden verschiedene Modelle und Mechanismen zum Management von Transaktionsaufzeichnungen angeführt und erläutert.

Zum Aufbau der Produktionsregeln und Symbolmengen, und somit der automatischen Zergliederung eines Satzes, können Algorithmen und Strategien aus dem Bereich der Computerlinguistik verwendet werden. Solche Systeme werden u.a. auch als Parser bezeichnet. [Co01] gibt einen Überblick über bestehende Lösungsansätze sowie aktuelle Forschungen.

Weitere Arbeiten zum Thema Sprachgenerierung sind [RD95] sowie [RD00], welche einen Überblick über Generierung natürlicher Sprache aus Sicht der praktischen Systementwicklung gibt. Auch im Bereich der KI[5] werden schon seit 25 Jahren Systeme zur Generierung natürlicher Sprache entwickelt. Insbesondere gilt dies für Erklärungskomponenten von Expertensystemen [BS84]. Diese sind in der Regel sehr mächtig und daher aus Effizienzgründen für die Verbalisierung von Datenbanktransaktion wenig geeignet, da hierbei für sehr viele Transaktionen sehr rasch Übersetzungen generiert werden müssen. [MB98] enthält unter anderem detaillierte Beiträge zu den Themen der Generierung und Verarbeitung von Sprache. In der „*Bibliography of Research in Natural Language Generation*“ findet sich eine umfassende Sammlung von Referenzen zur Generierung natürlicher Sprache: http://liinwww.ira.uka.de/bibliography/Ai/nlg.html.

## 6 Ausblick

Nicht immer werden in einem System sämtliche Daten in einer Datenbank gespeichert, auch wenn dies vor allem dem Ziel einer weitgehenden Datenunabhängigkeit dienlich wäre. Oftmals ist dies der Fall bei rechnerspezifischen Einstellungen (z.B. der Bildschirmauflösung). Zur Erweiterung des Systems könnten auch solche Änderungen durch Verwendung einer weiteren Methode aufgezeichnet werden. Dies erfordert allerdings stets einen gewissen Implementierungsaufwand, so dass dies nur für ausgewählte Änderungen getan werden sollte.

[5] Künstliche Intelligenz

Die durch die Serverapplikation generierten Texte können anhand von weiteren Parametern der Schablonen gefiltert werden. Bestimmte so selektierte Übersetzungen können dann in Form von Nachrichten an einzelne Benutzer oder Benutzergruppen versendet werden. Hierzu können bereits bestehende Nachrichtendienste verwendet werden. Mit Hilfe dieses Mechanismus können einzelne Benutzer, z.B. Administratoren, bei einzelnen Datenänderungen gezielt informiert werden. Nützlich könnte dies u. A. auch zur Erhöhung der Sicherheit des Gesamtsystems sein. Auch revisionsrelevante Daten, wie Kosten bestimmter Dienstleistungen können so durch weitere Personen direkt nachvollzogen und eventuell erneut nachgeprüft werden.

## Literaturverzeichnis


[ABHS99] Antos G.; Brinker, K.; Heidemann, W.; Sager, S.F.: Text und Gesprächslinguistik, 2. Band "Gesprächslinguistik" de Gruyter Berlin 1999

[BCN91] Batini, C.; Cesi, S.; Navathe, S.B.: Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings Pub. Co. 1991

[BS84] Buchanan, B.; Shortliffe, E. (Eds): Rule-Based Reasoning: the MYCIN Experiment . Addison-Wesley, Reading, MA, 1984

[Ch00] Christodoulakis, Dimitris (Ed.): Natural Language Processing - NLP 2000. Second Int. Conference, Patras, Greece, June 2-4, 2000, Proceedings. Lecture Notes in Computer Science 1835 Springer 2000

[BDDS99] Brandt, P.; Dettmer, D.; Dietrich, R.A.; Schön, G.: Sprachwissenschaft. Böhlau Verlag 1999

[Bu02] Buck, Olaf: Verbalisierung von Datenbanktransaktionen. Diplomarbeit am Institut für Informationssysteme der Universität zu Lübeck 2002, erschienen als Techn. Bericht B-02-05 der Institute für Informatik und Mathematik der Universität zu Lübeck 2002

[Co91] Cole; R.A.: Survey of the State of the Art in Human Language Technology. http://cslu.cse.ogi.edu/HLTsurvey

[GHJV96] Gamma; Helm; Johnson; Vlissides: Entwurfsmuster. Addison Wesley 1996

[GR93] Gray, J.; Reuter A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers 1993

[Hau92] Hausser, R.: Complexity in Left-Associative Grammar Theoretical Computer Science 106(2), 283-308. Dortrecht: Elsevier

[Hoe99] Hoeppner, W.: Der Mensch-Maschine Dialog http://www.uni-duisburg.de/FB3/CL/seitenD/Veroeffentlichungen/MM-Dialog.html und in [ABHS99]

[HR01] Härder, T.; Rahm E.: Datenbanksysteme – Konzepte und Techniken der Implementierung. Springer Verlag 2001

[MB98] McDonald, D.D.; Bolc L. (Ed.): Natural Language Generation Systems. Springer-Verlag 1998

[RD95] Reiter, Ehud; Dale, Robert: Building Applied Natural Language Generation Systems. Cambridge University Press 1995

[RD00] Reiter, Ehud; Dale, Robert: Building Natural Language Generation Systems. Cambridge University Press 2000

[So98] Solomon, David: Inside Windows NT. Microsoft Press 1998

[Vos00] Vossen, Gottfried: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. München, Oldenbourg Verlag