Universität - Gesamthochschule Paderborn

Thema:

Informationsverteilung in Föderativen Datenbankumgebungen
Konzepte für Archivierungskomponenten in Groupwareanwendungen anhand eines Prototyps

Diplomarbeit

im Fachgebiet 5 Wirtschaftsinformatik
am Lehrstuhl für Wirtschaftsinformatik UNI GH Paderborn

Themensteller: Prof. Dr. Ludwig Nastansky
Betreuer: Dipl.-Inf. Mohammed Drira

vorgelegt von: Jens Winkelmann
Hessenweg 1
33102 Paderborn
05251/409780

Abgabetermin: 23. Januar 1997

Inhaltsverzeichnis

Abkürzungsverzeichnis

ANSI	American Standards Comitee on Computers and Information Processing
API	Anwendungsprogrammierungsschnittstelle (Application Programming Interface)
BLOB	Binary Large Object
CDM	Allgemeines Datenmodel (common or canonical data model)
CLI	Call Level Interface
DB	Datenbank
DBMS	Datenbankmanagementsystem
DDBS	Verteiltes Datenbanksystem (distributed database system)
DDL	Datendefinitionssprache (data definition language)
DLL	Dynamic Link Library
DML	Datenmanipulationssprache (data manipulation language)
FDBS	Föderatives Datenbanksystem (federated database systems)
FDBMS	Föderatives Datenbankmanagementsystem
LS:DO	LotusScript Data Object
LSX	LotusScript Extension
MDBS	Multidatenbanksystem (multidatabase system)
ODBC	Open Database Connectivity
OLE	Object Linking and Embedding
SPARC	Standards Planning and Requirements Comitee
SQL	Structured Query Language
VDB	Verteilte Datenbanken
VTV	Verteilte Transaktionsverarbeitung
WOSA	Windows Open Services Architecture

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

?	Selektion (selection)
?	Projektion (projection)
?	Join
?	Vereinigung (union)
?	Differenz (difference)
?	Duchschnitt (intersection)
?	Aggregationsoperator (aggregate operator - sum; count)
?	Logisch und (and)
?	Logisch oder (or)
?	Logisch nicht (not)

1 Einleitung und Motivation

Heutige Unternehmen verteilen ihre Daten auf die unterschiedlichsten Datencontainer. Neben Relationalen oder Objektorientierten Datenbanken findet man auch dateiorientierte PC-Datenbanken oder Groupwaresysteme wie Lotus Notes. Diese Vielfalt ist nicht nur als Folge eines evolutionärer Prozesses zu sehen. Unterschiedliche Anforderungen an die Datenhaltung verhindern, daß nur ein System mit der Speicherung betraut wird. So setzt die Datenbankkomponente eines Groupwaresystems andere Schwerpunkte bei der Datenhaltung als eine Relationale Datenbank und diese wiederum andere als eine Objektorientierte Datenbank. Dieses begünstigt jedoch Datenredundanz zwischen den Systemen, die ihrerseits wieder die Gefahr von Inkonsistenz birgt. Daneben wünscht man sich weichere Grenzen zwischen den Systemen, so daß ein Datenaustausch zwischen ihnen unproblematischer ist.

Hier setzen Föderative Datenbanken ein, die ein relativ neues Forschungsgebiet der Informatik sind. Sie umfassen die Daten einer Föderation von Datenbanksystemen, um die angesprochenen Probleme zu lösen. Die Selbständigkeit der einzelnen Systeme innerhalb dieses Gebildes soll aber auch weiterhin garantiert bleiben.

Ein im Rahmen dieser Arbeit entwickelter Prototyp greift diese Problematik auf. Es handelt sich um ein Werkzeug für den Datenaustausch, das Elemente aus der Theorie Föderativer Datenbanken adaptiert und umsetzt. Konkret soll eine Brücke zwischen dem Groupwaresystem Lotus Notes und beliebigen Relationalen Datenbanken geschlagen werden.

Der Aufbau dieser Arbeit gliedert sich wie folgt: Nachfolgendes Kapitel führt, nach einer allgemeinen Betrachtung von verteilter Datenhaltung, in die Problematik von Föderativen Datenbanken ein. Kapitel 3 stellt ein Referenzmodell für diese Art von Datenbanken vor. Darauf aufbauend werden im darauffolgenden Kapitel Aufgaben und Problemstellungen bei der Entwicklung einer Föderativen Datenbank betrachtet. Kapitel 5 fast unter der Überschrift „Client-Server Architekturen“ Methoden und Konzepte für die Anbindung an Datenbanken zusammen. Bevor in Kapitel 7 der Prototyp ausführlich vorgestellt wird, führt Kapitel 6 in die Besonderheiten von Lotus Notes sowie deren Anwendungsentwicklung ein.

2 Verteilungsformen

Grob kann man zwei große Bereiche der Informationsverteilung unterscheiden. Auf der einen Seite gibt es die verteilten Datenbanken (VDB), die sich um die Verteilung von Daten beziehungsweise Datenbanken kümmern. Auf der anderen Seite steht die verteilte Transaktionsverarbeitung (VTV), deren Verteilungsobjekt die auf den Daten operierenden Programme beziehungsweise Anwendungen sind. Beim VDB-Konzept werden Daten von entfernten Datenbanken an den Ort der Verarbeitung geholt. Demgegenüber findet bei der verteilten Transaktionsverarbeitung ein Transport der Verarbeitungsleistung zum Ort der Daten statt. Hermann Kudlich klassifiziert beide Formen in seinem Buch „Verteilte Datenbanken“ [i] folgendermaßen:

	VDB-System	VTV-System
Verteilungsobjekte:	Daten	Verarbeitungsleistung
Verteilungssteuerung:	DB-System	Transaktionsmonitor
Auftraggeber:	Anwendung	Anwendung
Auftragnehmer:	DBMS	Partner-Anwendung
Auftragsinhalt:	DML (SELECT, UPDATE)	Funktionsaufruf
Antwort:	DB-Status (+ Daten)	Funktionsergebnis

Tab. 2-1: Verteilte Datenbanken und Transaktionsverarbeitung

Ein VDB-System charakterisiert er durch folgende Eigenschaften:

?? Anwendungsprogramme arbeiten in genau jenem Rechner, an den sich der Benutzer angeschlossen hat,

?? Kommunikationspartner im Netz sind die Datenbanksysteme, die den Zugriff auf die verteilten Datenbanken so koordinieren, als ob die Anwendungen die Daten lokal vorfinden würden.

Betrachtet man ein verteiltes Datenbanksystem näher und vergleicht es mit der traditionellen zentralen Lösung, so sind signifikante Vorteile auszumachen. Die zentralistische Variante belastet schnell den Zentralrechner und verursacht daneben ein hohes Kommunikationsaufkommen auf dem Netz. Dieses kann zu nicht akzeptablen Antwortzeiten des ganzen Systems führen mit der Folge, daß gegebenenfalls eine unkontrollierte Verwaltung eigener lokaler Daten gefördert wird. Diese wilde Datenhaltung kann wiederum zu einem inkonsistenten Datenbestand führen, und der

[i] Vergleiche [KH92] Seite 11 bis 24 - 1 Verteilte Verarbeitung: Ein neues Konzept

große Vorteil eines Datenbanksystem, der redundanzfreien konsistenten Speicherung, wäre verloren. Eine verteiltes Datenbanksystem wirkt mit der Verarbeitungskapazität mehrerer Rechner derartigem Fehlverhalten entgegen; zudem läßt sich die Leistungsfähigkeit durch Erhöhung der Rechneranzahl inkrementell erweitern. Die Aufteilung des Datenbestandes auf verschiedene geographisch verteilte Rechner ermöglicht auch eine Anpassung des Systemstruktur an die jeweilige betriebliche Organisationsstruktur. Die eigenen Daten werden dem Nutzerkreis näher gebracht. Dadurch soll gleichzeitig dessen Verantwortungsbewußtsein für deren sorgfältige Haltung erhöht werden.

2.1 Verteilte Informationssysteme

Mit steigender Globalisierung der Märkte und damit begleitend kürzeren Entwicklungs- und Herstellungszeiten von marktfähigen Produkten wird Information immer mehr zum entscheidenden Produktionsfaktor. Dabei sind „Interworking“, „Konnektivität“ und Cooperative Processing“ alles Begriffe, die eine neue Qualität der Datenverarbeitung beschreiben, nämlich die Fähigkeit, die Leistung unterschiedlicher Rechner über ein Netz zu verknüpfen sowie Anwendungen und Daten vielseitig nutzbar zu machen.

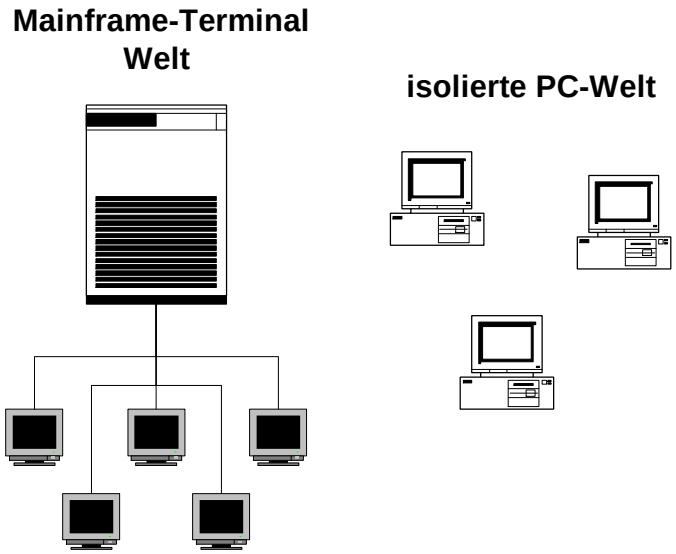


Abb. 2-1: Alte EDV-Welt

Die Zeit des Großrechners, der als Herzstück der Informationsinfrastruktur aus dem Datenverarbeitungszentrum alle Benutzer an ihren Terminals mit Anwendungen und

dazugehörigen Daten versorgt und der dazu parallel existierenden isolierten PC-Welt, neigt sich endgültig dem Ende zu. Eine moderne Informationsarchitektur besteht aus einem Netz heterogener Rechner- und Betriebssystem-Plattformen, die ihre speziellen Vorteile für die jeweilige Aufgabe geschickt nutzen. So hat der PC seine Berechtigung für Anwendungen die die Benutzerschnittstelle in den Mittelpunkt stellen. Mainframes sind jedoch unverzichtbar für die Verarbeitung von Massendaten. Eine Struktur von „lebendigen" Knoten, die durch ein Kommunikationsnetz verbunden sind, löst eine Struktur ab, die aus einer zentralen Komponente, an der sogenannte „dumme" Terminals hängen, besteht.

2.2 Gründe für Verteilung

Die Verteilung der Computerressourcen ist mit hohen Kosten und somit auch großem Risiko verbunden. Somit müssen die organisatorischen Aspekte und technischen Aspekte einer Dezentralisierung der Informationsarchitektur sorgsam ausgewertet werden, um die für das Unternehmen auch strategische Richtungsentscheidung ausreichend einschätzen zu können.

Organisatorische Aspekte

?? Integration isolierter Insellösungen zu einem Gesamtsystem.
So ist es beispielsweise möglich auf den operativen Datenbestand auch aller Filialen oder Zweigstellen jederzeit zuzugreifen. Aber auch Anwendungen stehen nun einem größerem Nutzerkreis zur Verfügung. Andere Softwarelösungen, die gerade eine verteilte Verarbeitung voraussetzen, wie beispielsweise Groupwareprodukte, können nun überhaupt erst realisiert werden.

?? Lokalitätsprinzip verbunden mit Übertragung von Selbstverantwortung.
Datenbestände sollen dort gehalten werden und die Verarbeitung soll dort ablaufen, wo sie ihre organisatorische Entstehung hat und in wessen Verantwortungsbereich sie zuzuordnen ist. Dieses verringert nicht nur Kommunikationszeit und Kommunikationskosten, sondern schärft auch das Verantwortungsbewußtsein für eigene Daten und Prozesse.

?? Schrittweises Wachstum.
Dezentrale Strukturen erlauben ein viel angemesseneres Wachstum oder auch Schrumpfen, so daß auf Veränderte Organisationsstrukturen flexibler reagiert werden kann.

?? Anwendergerechte Präsentation durch Nutzung neuer Oberflächen.
Damit Informationstechnik allgemein akzeptiert wird, muß eine intuitive Benutzerschnittstelle vorausgesetzt werden. Letztere läßt sich technisch nur dezentral befriedigend realsieren.

Technische Aspekte

?? Ausnutzung spezifischer Hardwareeigenschaften.
Im Rahmen dieser kooperativen Verarbeitungsform können die jeweiligen Stärken der unterschiedlichen Hardwaresysteme optimal genutzt werden. So bewahrt man beispielsweise Massendaten eines Datenbanksystems auf dafür dimensionierte Großrechner auf. Die grafische Aufbereitung einzelner Teildaten erfolgt aber besser auf dem Personal Computer am Arbeitsplatz.

?? Erhöhung der Verfügbarkeit und Sicherheit.
Da sich das System auf mehrere Knoten verteilt, ist der Ausfall eines Knotens nicht gleichzusetzen mit dem Ausfall des gesamten Informationssystems. Diese Risikoverteilung wird auch dadurch verbessert, daß einzelne Komponenten mehrfach verteilt auf mehrere Systemknoten vorhanden sind.

?? Leistung
Im sogenannten Leistungsverbund wird die Verarbeitungslast auf mehrere Knoten verteilt. Dadurch wird die Beanspruchung eines einzelnen Knotens verringert.

?? Ausbaufähigkeit
Eine Systemerweiterung kann bedarfsgerecht erfolgen und verursacht dadurch geringere Kosten.

?? Verringerung des Kommunikationsoverheads
Da Daten dort gehalten und verarbeitet werden wo sie auch anfallen, werden weniger Daten über das Kommunikationsnetz transportiert.

?? Leichte Steuerung und Wartung
Ein einzelner Knoten für sich betrachtet ist weniger komplex und Bedarf einer geringeren Wartung und Verwaltung als ein zusammenhängendes zentrales Hostsystem.

Nachteile von verteilten Strukturen

Trotz der Vielzahl von Vorteilen, die eine Dezentralisierung verspricht, darf nicht unterschätzt werden, daß es sich insgesamt um ein viel komplexeres System handelt. In ihm erfahren die Verwaltungsaufgaben im Rahmen von Netzadministration, systemweite Datensicherung sowie die ganze Problematik des Zugriffsschutzes und der Zugriffskontrolle einen höheren Stellenwert.

2.3 Anforderungen an verteilte Datenhaltung

Nachfolgend sollen die von Chris Date formulierten 12 Regeln für verteiltes Datenmanagement abgehandelt werden[i], die er 1987 in Analogie zu den Coddschen Regeln über Relationale Datenbanksysteme veröffentlicht hat. Diese Forderungen an eine verteilte Datenhaltung hat keinerlei Anspruch auf Vollständigkeit. Die einzelnen Punkte sind weder voneinander unabhängig, noch sind sie in sich widerspruchsfrei. Vielmehr werden sie in diesem Zusammenhang erwähnt um aufzuzeigen, daß es nicht „das“ verteilte Datenbanksystem gibt, sondern daß unterschiedliche Probleme unterschiedliche Lösungen erfordern.

?? Jeder Knoten eines verteilten DB-Systems muß zugleich ein für sich autonomes System sein. Autonom bedeutet, daß jedes der im Rechnerverbund zusammengeschlossenen Datenbanksysteme über lokale Daten verfügt, deren Verwaltung in Verantwortung des lokalen Datenbankadministrators liegt. Damit soll gewährleistet werden, daß Anwendungen, die nur auf lokale Daten zugreifen, durch die Einbindung in den Verbund nicht benachteiligt werden. Alle Möglichkeiten, die das System als eventuell isolierte Insel hatte, sollen auch weiterhin im Rechnerverbund garantiert werden.

?? Ein Knoten eines verteilten DB-Systems darf nicht von einem zentralen Knoten abhängig sein. Es darf keinen zentralen Knoten oder Masterknoten für irgendwelche zentralen Daten geben, wie z.B. zentralisierte Anfragebearbeitung oder zentralisierte Transaktionssteuerung.

?? Jeder Knoten eines verteilten DB-Systems muß ununterbrochen Betrieb gewährleisten. Folglich sollten Administrationsarbeiten, wie beispielsweise das Einfügen eines neuen Kno tens in ein bestehendes Rechnernetz, das Entfernen eines Knotens, das Anlegen von Fragmenten sowie die Installation einer neuen Version einer Datenbanksoftware ohne Unterbrechung des gesamten Rechnerverbundes durchführbar sein.

?? Die Verteilung der Datenbanken im Netz ist für den Anwender nicht sichtbar. Diese Verteilungstransparenz (auch als Netztransparenz bezeichnet) soll dem Anwender vorgaukeln, daß ihm alle Daten lokal bereitstehen und diese nicht aus verschiedenen verteilten Datenbanken über das Netz herangeschafft werden müssen. In diesem Zusammenhang fallen noch die Begriffe Ortstransparenz, welches die Verschleierung des physikalischen Speicherorts bezeichnet, und Namenstransparenz, die für eine eindeutige netzwerkweite Namensgebung sorgt.

?? Fragmente einer verteilten Datenbank können an beliebigen Orten gespeichert werden. Als eine Weiterführung der Verteilungstransparenz gilt die Fragmentierungstransparenz. Hierbei werden auch Teile von Tabellen, sogenannte Fragmente, über das Verbundsystem verteilt, wobei dieses dem Anwender ebenfalls verborgen bleibt. Durch ein intell igente

[i] Vergleiche [KH92] Seite 146 bis 154 - 6.1 12 Regeln für verteilte Datenbanken

Fragmentierung lassen sich viele Operationen lokal durchführen, und der Verkehr im Netz wird reduziert (*Siehe 4.4.1* ***Fragmentierung und Replikation***).

?? Ein verteiltes DB-System muß die Speicherung von Replikaten an unterschiedlichen Orten unterstützen.
Replikationen sind identische Kopien von Datenbeständen, die der schon von konventionellen Datenbanken geforderten redundanzfreien Datenhaltung eigentlich widersprechen. Dieses nimmt man jedoch in Kauf, da sich die Verfügbarkeit der replizierten Daten erhöht, und somit Zugriffe schneller ausgeführt werden können. Eine Replizierungstransparenz fordert nun, daß Operationen von Anwendern nicht durch Zusatzaufwand in Form von mehrfachen Änderungen oder der Wahl der günstigsten Kopie erschwert werden. Um Probleme im Zusammenhang von Replikationen zu mindern, wird bei einigen Repliken nur ein Lesezugriff erlaubt, so daß Schreiboperationen nur auf dem Original der Masterkopie gewährt werden. Dadurch ist die Masterkopie immer im konsistenten Zustand. Die Read-Only-Kopie, auch Snapshot genannt, zeigt ein Abbild der Masterkopie zur Zeit der letzten Datenangleichung (*Siehe 4.4.1* ***Fragmentierung und Replikation***)

?? Datenbankanforderungen im Gesamtsystem müssen optimiert werden.
Diese Regel besagt, daß ein verteiltes Datenbanksystem Datenbankanforderungen so auflösen sollte, daß eine optimale Durchführung der Datenbankoperationen gewährleistet wird. Diese Forderung, die auch schon bei konventionellen zentralen Datenbanken besteht, gewinnt dadurch an Bedeutung, daß der Datentransfer über Netz ein bedeutender Kostenfaktor ist. Ein System, das hier eklatante Schwächen aufweist, disqualifiziert sich selber für einen praktisch sinnvollen Einsatz.

?? Die Konsistenz rechnerübergreifender Transaktionen ist zu gewährleisten.
Eine Transaktion ist eine Folge logisch zusammengehörender Operationen, die eine Datenbank von einem konsistenten Zustand in einen neuen kosistenten Zustand bringt. Kann eine Transaktion aus irgend einem Grund nicht zu Ende geführt werden, so müssen alle bislang ausgeführten Operationen im Rollback-Verfahren zurückgenommen werden. Diese aus der konventionellen Datenbankwelt bekannte Forderung verlangt man auch für verteilte Datenbanken. Hierzu sind komplexe Protokolle erforderlich wie das 2 Phasen-Comit Protokoll.

?? Die Unabhängigkeit von der Hardware muß gegeben sein.
Ein verteiltes Datenbanksystem soll auf Systemen mit unterschiedlichen Rechnertypen einsetzbar sein.

?? Die Unabhängigkeit vom Betriebssystem muß gegeben sein.
Ein verteiltes Datenbanksystem soll auf Rechnern mit unterschiedlichen Betriebssystemen lauffähig sein.

?? Die Unabhängigkeit vom Netzwerk muß gegeben sein.
Ein verteiltes Datenbanksystem soll unterschiedliche Kommunikationswege mit unterschiedlichen Kommunikationsprotokollen zwischen den verschiedenen Rechnerknoten unterstützen.

?? Ein verteiltes DB-System darf nicht von den eingesetzten Datenbankkomponenten abhängig sein.
Ein verteiltes DBMS sollte sich auch aus verschiedenartigen, heterogenen Datenbanksystemen zusammensetzen können, die auch auf unterschiedlichen Datenbankmodellen beruhen können.

2.4 Kategorien von Datenbanksystemen für verteilte Datenhaltung

Die Anforderungen an verteilte Datenbanken hat gezeigt, daß es nicht ein omnipotentes Datenbanksystem geben kann, das alle Forderungen erfüllen kann. Vielmehr machen die 12 Regeln deutlich, daß für bestimmte Konstellationen und Bedürfnisse unterschiedliche verteilte Systeme benötigt werden. Im Weiteren sollen nun Arten von Datenbanksystemen, die sich mit verteilten Daten beschäftigen, kategorisiert werden.

Da sich in der Literatur noch keine eindeutige Begriffswelt etabliert hat, lehne ich mich an die Definition von Amit P. Sheth und James A. Larson aus ihrem ACM Computing Surveys Artikel [SL90] an. Ausgehend von einem Zentralen Datenbanksystem (centralized DBS) unterscheiden sie:

?? Verteilte Datenbanksysteme (distributed database system DDBS)

?? Multidatenbanksysteme (multidatabase system MDBS)

?? Föderative Datenbanksysteme (federated database systems FDBS)

Systeme, die nur periodischen, nicht transaktionsbasierten Austausch von Daten zwischen Datenbanksystemen anbieten, werden nicht als Datenbanksysteme bezeichnet. Besser trifft die Bezeichnung Datenaustauschsysteme (data exchange system) zu. Weiterhin sind Systeme, die zu einem Zeitpunkt nur Zugriff auf eine einzelne Datenbank erlauben, eher als Datenbankmanagementsystem-Schnittstellen (remote DBMS interface) zu bezeichnen.

2.4.1 Verteilte Datenbanksysteme

Ein Verteiltes Datenbanksystem besteht aus einem einzelnen verteilten Datenbankmanagementsystem, dem viele einzelne Datenbanken unterstellt sind. Die Datenbanken befinden sich auf einen einzelnen Rechner oder sind verteilt über ein Rechnerverbund, der sich

wiederum aus unterschiedlicher Hardware, Systemsoftware und Kommunikationskomponenten zusammensetzen kann.

2.4.2 Multidatenbanksysteme

Anders als bei Verteilten Datenbanken sind einem Multidatenbanksystem keine Datenbanken sondern mehrere Datenbanksysteme unterstellt. Jede Datenbanksystemkomponente wird von einem eigenen Komponentenadministrator verwaltet. Eine Systemkomponente kann wiederum ein zentrales oder ein verteiltes Datenbanksystem sein und auf einem einzelnen Rechner oder einem Rechnerverbund untergebracht sein. Man spricht von einem homogenen System, falls alle Komponentensysteme von der gleichen Art sind. Andernfalls wird das Multidatenbanksystem als heterogen bezeichnet.

2.4.3 Föderative Datenbanken

Multidatenbanksysteme kann man über die Dimensionen Autonomie, Verteilung und Heterogenität in nichtföderative und föderative Datenbanksysteme unterteilen. Die Komponentensysteme von Föderativen Datenbanksystemen bleiben autonom. Auf die Datenbanksystemkomponente kann zugegriffen werden, ohne daß immer der Umweg über das Föderative Datenbanksystem gemacht werden muß. Folglich gibt es bei der föderativen Architektur keine dominierende zentrale Steuerung. Die Komponentensysteme beziehungsweise deren Administratoren behalten Kontrolle über ihre Datenbestände.

Die Informationsinfrastruktur eines Unternehmens hat sich in einem jahrzehntewährenden Prozeß aufgebaut, mit dem Ergebnis, daß Datenbanksysteme der unterschiedlichsten Art in einem gewachsenen Unternehmen zu finden sind. Ebenso wurden bewußt Systeme unterschiedlichster Ausprägung eingekauft, da man auf ihre individuellen Fähigkeiten nicht verzichten wollte. Föderative Datenbanksysteme sind somit als Kompromiß zwischen keiner Integration und Integration dieser Systeme zu verstehen. So erlauben Föderative Datenbanken eine kontrollierte Datenteilung, ohne daß die einbezogenen Systeme angepaßt werden müssen, was eventuell zu nicht akzeptablen Kosten führen würde oder sogar unmöglich wäre.

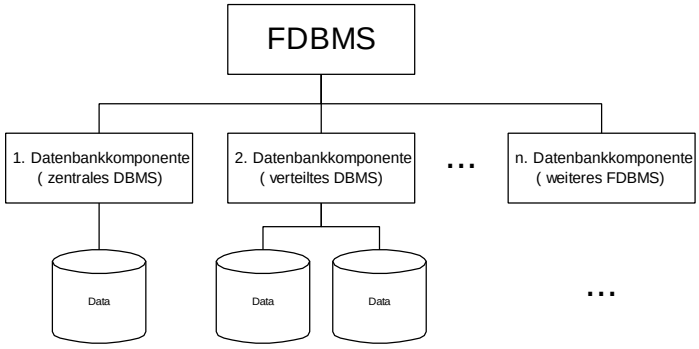


Abb. 2-2: Ein FDBMS und seine Komponenten

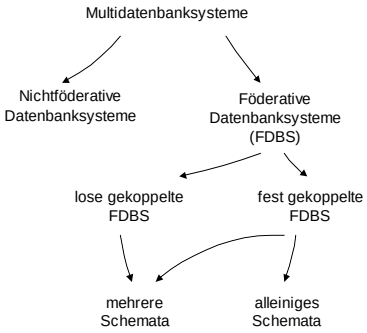
Ein Datenbanksystem kann in mehreren Föderativen Datenbanksystemen eingebunden sein. Dabei spielt es keine Rolle, ob dieses ein zentrales, verteiltes oder auch selbst ein Föderatives System ist. Die einzelnen Komponenten können unterschiedliche Datenmodelle, unterschiedliche Abfragesprachen oder auch unterschiedliche Transaktionsmanagementfähigkeiten besitzen. Man muß zwischen lokalen Operationen, die isoliert innerhalb der Komponente ablaufen, und externen Operationen, die von einer anderen Komponente über das Föderative Datenbanksystem angestoßen werden, unterscheiden.

Charakterisierung

Föderative Datenbanken lassen sich nach dem Grad der Datenverteilung sowie dem Grad der Heterogenität und Autonomität ihrer Komponentsysteme charakterisieren. Die Datenverteilung ist im Gegensatz zu Verteilten Datenbank nicht Ergebnis eines intelligenteren Datenmanagement sondern Grund für die Etablierung eines Föderativen Datenbanksystems. Innerhalb der Heterogenität kann man zwischen Heterogenität aufgrund von unterschiedlichen Datenbankmanagementsystemen und semantischer Heterogenität unterscheiden. Erstere umfaßt unterschiedliche Datenmodelle, unterschiedliche Konsistenzbedingungen oder auch unterschiedliche Abfragesprachen zwischen den beteiligten Systemkomponenten. Semantische Heterogenität beschäftigt sich mit dem komplexen Problem, daß gleiche oder verwandte Daten in unterschiedlichen Systemen unterschiedlich definiert sind (*Siehe 4.3* ***Schema- und Datenheterogenität***). Auch die Autonomie der beteiligten Systemkomponenten kann man differenziert betrachten. Die Planungsautonomie beschäftigt sich mit der Berechtigung der jeweiligen Datenbanksystemkomponente das eigene Datenbankdesign in eigener Regie zu führen. Eine hohe Kommunikationsautonomie überläßt der Komponente die Entscheidung, wie und wann

sie auf Anfragen von anderen Komponenten reagiert. Ausführungsautonomie läßt der Komponente lokale Operationen in Eigenregie durchführen. Vereinigungsautonomie, als letzte der unvollständigen Auflistung, befähigt das einzelne Datenbanksystem selbst zu entscheiden, welche Ressourcen es in den föderativen Verbund mit einbringen will.

Kategorisierung



Die obige Charakterisierung von Föderativen Datenbanken macht deutlich, daß sich unter dem Begriff Föderative Datenbank eine breite Palette von unterschiedlicher Software einordnen läßt. Dieses erfordert eine weitere genauere Kategorisierung von Föderativen Datenbanken. So unterteilt man sie in lose gekoppelte und fest gekoppelte, je nachdem wer die Föderation verwaltet und wie die Komponenten integriert sind. Ein Föderatives Datenbanksystem bezeichnet man als lose gekoppelt, wenn das Erzeugen und Verwalten der jeweiligen Föderation dem Benutzer überlassen bleibt. Übernimmt diese Aufgabe die Föderation beziehungsweise deren Administrator und steuert er dadurch den Zugriff auf die einzelnen Komponenten zentral, so spricht man von einem fest gekoppelten System (*Siehe 3.5* ***Gestaltungsspielräume des Modells***). Angelpunkt einer Föderativen Datenbank ist das föderative Datenbankschema auf dem Operationen ausgeführt werden. Lose gekoppelte Föderative Datenbanken unterstützen immer mehrere Schemata. Bei fest gekoppelten unterscheidet man jene mit mehreren Schemata und jene mit nur einem Schema. Möchte man die Daten stärker kontrollieren und sind übergreifende Konsistenzbedingungen unbedingt erforderlich, so kann man dieses nur mit einem einzelnen föderativen Schema bewerkstelligen.

3 Referenzmodell für Föderative Datenbanken

Nachfolgend soll ein Referenzmodell für die Beschreibung von Föderativen Datenbanken vorgestellt werden [SL90]. Es bildet den Rahmen, um unterschiedliche Architekturmöglichkeiten Föderativer Datenbanken verstehen, kategorisieren und vergleichen zu können. Dabei kann das Modell als Planungsinstrument für zu entwickelnde Systeme dienen, oder man nutzt es zur einheitlichen Architekturbeschreibung bestehender Datenbanksysteme.

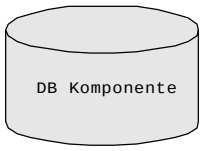
Als erstes werden die erforderlichen Modellelemente vorgestellt. Mit Hilfe des eingeführten Modells soll im weiteren Verlauf die weit verbreitete ANSI/SPARC Drei-Schichten-Architektur dargestellt werden. Abschließend wird diese ANSI/SPARC Architektur auf eine Fünf-Schichten-Architektur für Föderative Datenbanken erweitert.

3.1 Modellelemente

Ein Referenzmodell gibt ein Schema vor, in das sich die zu beschreibende Architektur einfügen muß. Das Schema setzt sich aus drei Hauptkomponenten zusammen:

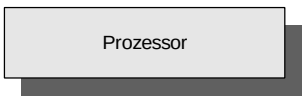
?? **Datenbankkomponente** (Component Database)

Eine Datenbankkomponente ist eine Datenbank, die an der Föderation teilnimmt. Dabei ist es unerheblich, welches Datenmodell die jeweilige Komponente unterstützt.



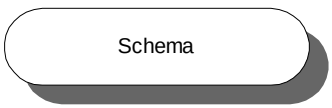
?? **Prozessor** (Processor)

Ein Prozessor ist ein Softwaremodul, das für eine bestimmte Aufgabe innerhalb des Modells zuständig ist.



?? **Schema** (Schema)

Ein Datenbankschema beschreibt Datenobjekte, und wie sie zueinander in Beziehung stehen.



3.2 Prozessortypen

Prozessoren sind innerhalb dieser Modellwelt als Softwaremodule zu sehen, die einer bestimmten Aufgabe dienen. Unterschiedliche Architekturen unterscheiden sich in den unterschiedlichen Arten der verwendeten Prozessoren und der Beziehungen zwischen ihnen. Man unterscheidet vier Arten von Prozessoren, die nun nacheinander kurz beschrieben werden sollen.

Transformationsprozessor *(Transforming Processor)*

Ein Transformationsprozessor hat die Aufgabe, Kommandos von der Quellsprache in die Zielsprache zu übersetzen und Daten vom Quellformat ins Zielformat zu überführen. Somit ist er verantwortlich für die Datenmodelltransparenz, die Kommandos und Datenstruktur des einen Prozessors einem anderen verbirgt. Eine Übersetzung von SQL-Kommandos in CODASYL-Kommandos wäre ein Beispiel. Inverse Transformationsprozessoren erlauben auch eine Rücktransformation von der Zielsprache/-format in die Quellsprache/-format. In unserem Beispiel könnten CODASYL-Kommandos zurück in SQL-Kommandos transformiert werden. Als Hilfsmittel benötigt der Prozessor eine Abbildung (Mapping) zwischen den einzelnen Objekten des Quell- und Zielschemas. Diese Abbildungslogik kann einerseits im Prozessorcode integriert sein oder andererseits als Metadaten in einer separaten Datenstruktur abgelegt sein. Letzteres führte zu Mapping-Metadaten, die bei der Format- und Kommandotransformation vom Transformationsprozessor ausgelesen werden.

Filterprozessor *(Filtering Processor)*

Ein Filterprozeß kann den Zugriff mit bestimmten Kommandos beziehungsweise den Zugriff auf bestimmte Daten einschränken. Auch diese Information kann fest verdrahtet im Code integriert sein oder in externe Datenstrukturen ausgelagert sein. In der Praxis lassen sich so eine Syntakskontrolle, eine Zugriffskontrolle und auch diverse Integritätsbedingungen realisieren.

Konstruktionsprozessor *(Constructing Processor)*

Der Konstruktionsprozessor transformiert eine Operation, die er von einem einzelnen Prozessor bekommt, zu mehreren Operationen, die er an mehrere Prozessoren weitergibt. Umgekehrt fügt er

die zurückgemeldeten Daten zusammen und gibt sie dem Ausgangsprozessor als Resultat zurück. Hiermit können Stufen unterschiedlicher Verteilungstransparenz garantiert werden. Das heißt, das Splitten der Operation und Zusammenführen der Resultate soll für vorgelagerte Prozesse transparent, also nicht sichtbar, erfolgen.

Zugriffsprozessor *(Accessing Processor)*

Für den eigentlichen Zugriff auf die Datenbank ist der Zugriffsprozessor zuständig.

3.3 ANSI/SPARC Drei-Schichten-Architektur

1975 wurde von einer Studiengruppe des Standards Planning and Requirements Comitee (SPARC) of the American Standards Comitee on Computers and Information Processing (ANSI/X3) ein Vorschlag für eine Standardisierung von Datenbankarchitekturen erarbeitet. Gemäß diesem Vorschlag lassen sich drei Betrachtungsebenen in Datenbanksystemen unterscheiden. Die Unterteilung nach konzeptionellem Schema, internem Schema und externem Schema sollte die Anwendung von der physischen Datenbasis trennen.

Konzeptionelle Schema

Das Konzeptionelle Schema umfaßt die logische Gesamtsicht auf die Daten. Alle Datenobjekte und ihre Beziehungen untereinander werden in der Gesamtheit unabhängig von einem zugrundeliegenden Datenbanksystem dargestellt.

Interne Schema

Das interne Schema beschreibt die physische Organisation der Daten, das heißt, wie und wo die Daten physisch gespeichert sind. Die Grundlage bildet das Modell der konzeptionellen Ebene. Hierfür sind die Zugriffspfade, Datenstrukturen, Datenverteilung usw. festzulegen. Dabei sollen beide Schemen unabhängig voneinander sein. Werden Änderungen bei der physischen Speicherung vorgenommen, so soll dieses nicht das konzeptionelle Schema beeinflussen. Umgekehrt soll dieses möglichst auch gelten.

Externe Schema

Nicht alle Benutzer benötigen einen vollen Zugriff auf die ganze Datenbasis. Externe Schemen definieren die Sichten einzelner Benutzer bzw. Anwendungen auf die Datenbank. Dabei umfassen diese Sichten jeweils Ausschnitte aus dem Gesamtdatenmodell. Sie müssen somit aus der konzeptionellen Gesamtsicht ableitbar sein.

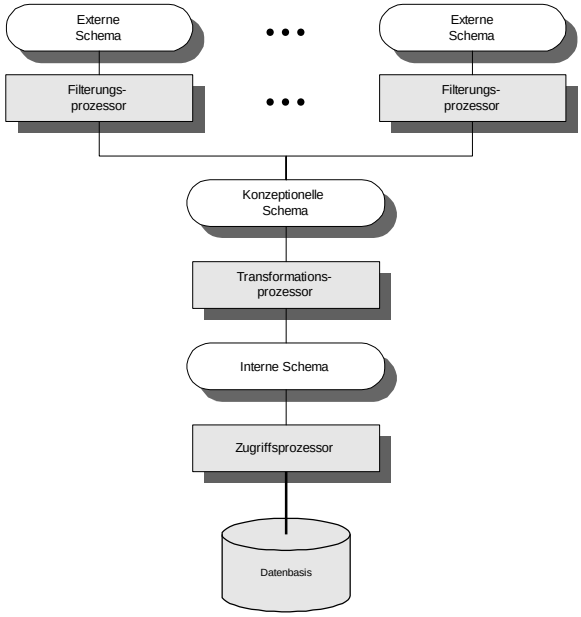


Abb. 3-1: ANSI/SPARC Drei-Schichten-Architektur

Obige Grafik zeigt das Drei-Schichten-Modell mit den im vorherigen Kapitel eingeführten Prozessoren. Filterprozessoren nutzen die Information des jeweiligen externen Schemas, um zu kontrollieren, auf welche Daten bestimmte Benutzer zugreifen dürfen. Der Transformationsprozessor übersetzt Kommandos, die auf das Konzeptionelle Schema zugeschnitten sind, in Kommandos fürs Interne Schema. Der Zugriffsprozessor führt Zugriffe auf das physische Speichermedium aus.

3.4 Fünf-Schichten-Architektur für FDBS

Die Drei-Schichten-Architektur ist angemessen, um konventionelle zentrale Datenbanksysteme zu beschreiben. Um jedoch die drei Dimensionen eines Föderativen Datenbanksystems, Verteilung, Heteroginität und Autonomität, abzubilden, muß die Architektur erweitert werden. Amit Sheth

und James Larson haben ein Modell vorgeschlagen [SL90], das sich aus fünf Schichten zusammensetzt. Es findet in der Literatur über föderative Datenbanken große Beachtung. Die fünf Schemen werden nun aufsteigend von der untersten zur obersten Schicht vorgestellt.

Lokales Schema *(Local Schema)*

Als Lokales Schema bezeichnet man das Konzeptionelle Schema jeder Datenbanksystemkomponente. Die unterschiedliche Systemkomponenten können unterschiedliche Datenmodelle unterstützen, was sich auch in den lokalen Schemen widerspiegelt.

Komponenten Schema *(Component Schema)*

Die Heterogenität der Datenmodelle verlangt, daß man sich auf ein Datenmodell für die gesamte Föderation einigt. Dieses Modell wird, im Sinne eines gemeinsamen Nenners, Allgemeines Datenmodell oder Kanonisches Datenmodell (common or canonical data model CDM) genannt. Das Komponenten Schema entspricht logisch dem Lokalen Schema, wobei jedoch das Kanonische Datenmodell zugrunde gelegt wird. Somit erhält man eine einheitliche Repräsentation für die gesamte Föderation. Weiterhin können semantische Komponenten, die dem lokalen Schema fehlen, ergänzend in dem nun eventuell semantisch reicheren Komponenten Schema hinzugefügt werden. Die Wahl des Kanonischen Datenmodells ist entscheidend für die Güte der Föderativen Datenbank. Einerseits sollte das CDM semantische Aspekte nicht vernachlässigen, andererseits sollte es weitverbreitet mit großer praktischer und theoretischer Bedeutung sein. Für das relationale Modell spräche der zweite Punkt, wobei jedoch Abstriche bei dessen semantischer Beschreibungskraft zu machen sind.

Export Schema *(Export Schema)*

Jede Datenbankkomponente soll selbständig bestimmen, welche Daten sie mit in die Föderation einbringt und welche Operationen sie auf ihre Daten zuläßt. Hierzu wird für jede Komponente ein Exportschema definiert. Als Teilmenge des Komponenten Schemas enthält es nur Daten, die auch in die Föderation eingebracht werden sollen.

***Föderatives Schema** (Federated Schema)*

Faßt man alle Export Schemen zusammen, so erhält man das Föderative Schema. Dieses ist zusätzlich angereichert mit Informationen über die Datenverteilung, also auf welchen Datenbankkomponenten sich ein bestimmtes Datum des Föderativen Schemas befindet. Hält man diese Information separat, so bezeichnet man das daraus entstandene Schema auch als Verteilungsschema (distribution schema) oder Zuteilungsschema (allocation schema). Es besteht keine zwingende Notwendigkeit, daß nur ein Föderatives Schema pro Föderation gebildet wird. Vielmehr kann je Benutzergruppe, mit ihren unterschiedlichen Aufgaben und den daraus resultierenden verschiedenartigen Informationsbedarf, ein eigenes Föderatives Schema gebildet werden.

***Externes Schema** (External Schema)*

Ein Externes Schema spezifiziert wiederum eine Teilmenge des Föderativen Schemas für eine bestimmte Anwenderklasse oder auch Anwendungssoftware. Es besteht somit eine große Ähnlichkeit mit dem Externen Schema des ANSI/SPARC Drei-Schichten-Modells. Zusätzliche Integritätsbedingungen oder auch Zugriffskontrollen auf das Föderative Schema können hier ihren Platz finden. Auch wäre denkbar, daß das Datenmodell eines Externen Schemas nicht dem des kanonischen Datenmodells des Föderativen Schemas entspricht, sondern ein speziell auf die Bedürfnisse des Anwenderkreises ausgerichtetes Datenmodell verwendet wird.

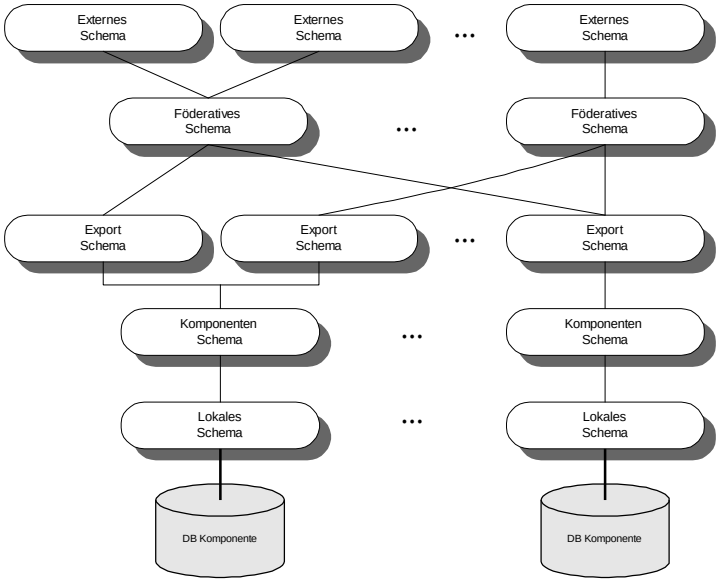


Abb. 3-2: Alle Schemata eines Föderativen Modells

Obige Skizze zeigt alle Schemen innerhalb eines Fünf-Schichten-Modells für Föderative Datenbanken. Jedes Entity eines Schemas findet seine Entsprechung im nachfolgenden Schema. Die entsprechende Abbildung zwischen den logisch gleichen Entities verschiedener Schemen kann entweder innerhalb des Schemas selber oder separat gespeichert werden. Zusammenfassend soll die ideelle Systemarchitektur mit allen möglichen Schemen und Prozessoren dargelegt werden, um das Zusammenspiel aller Komponenten genauer zu studieren.

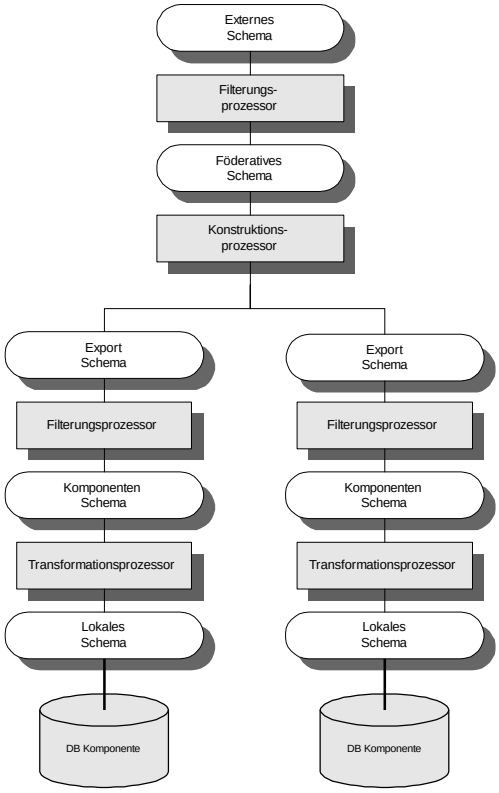


Abb. 3-3: Fünf-Schichten Architektur für Föderative Datenbanken

Fundament einer föderativen Datenbank sind die einzelnen Datenbankkomponenten mit ihren ihnen eigenen lokalen Schemen. So kann in der Föderation eine Relationale Datenbank mit Relationenmodell, eine Objektorientierte Datenbank mit einem Objektmodell sowie eine Lotus Notes Datenbank mit dem speziellen Lotus Notes Datenmodell beteiligt sein. Wählt man als kanonisches Modell (CMD) das Relationenmodell, so ist das Objektmodell und das Lotus Notes Datenmodell ins Relationenmodell zu überführen. Diese Aufgabe übernimmt der Transformationsprozessor, der dieses entweder generisch mit Hilfe von abgelegten Metadaten oder auch statisch mit fest implementierten Regeln leistet. Da die Relationale Datenbank in unserem Beispiel bereits im CMD (canonical or common data model) vorhanden ist, entspricht ihr Komponenten Schema dem Lokalen Schema. Folglich ist bei dieser Konstellation kein Transformationsprozeß notwendig. Jede Datenbankkomponente möchte nur bestimmte ausgewählte Daten in die Föderation mit einbringen. Hierzu wird für jede Komponente ein Export Schema gebildet, das, als Teilmenge des Komponenten Schemas, nur diese entsprechenden Daten enthält. Ein Filterprozeß zwischen Export Schema und Komponenten Schema verhindert unberechtigte Zugriffe und ermöglicht zugleich die Implementierung von Integritätsbedingungen

und Syntaxskontrolle. Faßt man die einzelnen Export Schemen der jeweiligen Datenbankkomponenten zusammen, so erhält man das Föderative Schema. Zwischen beiden Schemen ermöglicht der Konstruktionsprozessor die erforderliche Transparenz, so daß vorgelagerte Prozesse unabhängig vom physischen Speicherort agieren können. Aus dem Föderativen Schema können nun wiederum beliebige Externe Schemen generiert werden, die in Art und Umfang auf spezielle Benutzergruppen zugeschnitten sind. Technisch wird dieses wiederum mit einem Filterprozeß ermöglicht, der zwischen Externen Schema und Föderativen Schema seine Aufgaben verrichtet.

Um ein besseres Verständnis über Autonomie und die Zugriffsverwaltung innerhalb des Föderativen Modells zu bekommen, führen wir den Begriff des Komponenten-Datenbankverwalters und des Föderations-Datenbankverwalters ein. Hinter diesen muß sich nicht zwangsweise eine reale Person verbergen. Jede Datenbankkomponente wird von einem Komponenten-Datenbankverwalter betreut. Er hat die Aufgabe, daß jeweilige Export Schema der Datenbankkomponente zu definieren, um bestimmten Föderations-Benutzergruppen den Zugriff auf bestimmte Daten zu gewähren. Ein Föderations-Datenbankverwalter ist verantwortlich für das Föderationsschema beziehungsweise die Föderationsschemen sowie deren abgeleiteten Externen Schemen.

3.5 Gestaltungsspielräume des Modells

Wie bereits erwähnt, handelt es sich beim Fünf-Schichten-Architekturmodell um ein Referenzmodell zur Beschreibung diverser konkreter Architekturen. Architekturen unterscheiden sich durch das Entfernen oder Hinzufügen bestimmter Modellkomponenten sowie deren unterschiedliche Bedeutung und Stellung innerhalb der Architektur.

Lose und fest gekoppelte FDBS

Grob kann man zwischen lose gekoppelten FDBS und fest gekoppelten FDBS unterscheiden (*Siehe 2.4.3 Föderative Datenbanken*). Beim lose gekoppelten FDBS stellt jeder Benutzer aus allen zur Verfügung stehenden Export Schemen sein eigenes individuelles Föderations Schema zusammen und ist somit gleichzeitig Föderations-Datenbankverwalter seiner Föderation. Im

Gegensatz zu dieser Schemaimportierung steht die Schemaintegration beim fest gekoppelten FDBS. Hier hat der Föderations-Datenbankverwalter eine zentralere und mächtigere Bedeutung. Er steht als Vermittler zwischen den Komponenten-Datenbankverwaltern auf der einen Seite und den Benutzern der Föderation auf der anderen Seite. Durch Verhandlungen mit beiden Gruppen werden Export Schemen und deren Integration zum Föderations Schema, sowie Externe Schemen als Schnittstelle zum Benutzer, entwickelt.

Hieraus läßt sich auch schließen, für welche Anwendungen lose beziehungsweise fest gekoppelte FDBS geeignet sind. Soll sich der Benutzer ein eigenes Föderationsschema dynamisch zusammenstellen können, ohne mehrere Instanzen einzuschalten, so ist ein lose gekoppeltes FDBS vorzuziehen. Hier wird eventuell sogar die Schemadefinition nach Gebrauch wieder verworfen. Fest gekoppelte FDBS sind auf längere Dauer angelegt und umfassen eine zentral ausgelegte Zugriffskontrolle. Bezahlt wird diese statische Variante mit einer vergleichsweise geringeren Flexibilität.

4 Entwicklung einer Föderativen Datenbank

Die Literatur unterscheidet zwei Methoden bei der Entwicklung einer Föderativen Datenbank. Die Bottom-Up Methode wird verwendet, um verschiedene existierende Datenbanken in eine zu entwickelnde Föderation zusammenzufassen, oder um eine weitere Datenbank in eine bereits bestehenden Föderation zu integrieren. Benötigt eine neue Anwendung Daten, die sie nicht aus dem vorhandenen Föderationsschema beziehen kann, so muß letzteres erweitert werden. Im Top-Down Verfahren wird dieser neue Datenbedarf auf die darunterliegenden Datenbankkomponenten weitergetragen. In der Praxis findet keine reine Anwendung nur eines Verfahrens statt, vielmehr verwendet man parallel Elemente aus beiden Methoden. Im weiteren Verlauf soll die Bottom-Up Methode näher betrachtet werden, da sich mit ihr die Entwicklungsschritte zur Erstellung einer Föderativen Datenbank gut studieren lassen. Anschließend sollen die darin enthaltenen Schritte und deren Problematik näher betrachtet werden.

4.1 Der Bottom-Up Entwicklungsprozeß

Das Bottom-Up Verfahren stellt sich zur Aufgabe, existierende unterschiedliche Datenbanken zu einer Föderation zusammenzufassen. Ausgangspunkt sind die einzelnen Datenbanken mit ihren unterschiedlichen lokalen Schemen. Zuerst muß man sich auf ein CDM (canonical or common data model), also ein allgemeingültiges Datenmodell für die gesamte Föderation, einigen. Alle Lokalen Schemen, die sich nicht von vornherein im CDM befinden, müssen ins CDM übersetzt werden. Zur Schemaübersetzung gehört einerseits eine Abbildung, die jedem Objekt aus dem Lokalen Schema ein Objekt des Komponenten Schema gegenüberstellt, sowie andererseits die Entwicklung eines Transformationsprozessors, der Kommandos für die Föderative Datenbank in entsprechende Kommandos, zugeschnitten auf die entsprechende Datenbankkomponente, übersetzt. Anschließend erfolgt die Schemadefinition, in die der Komponenten-Datenbankverwalter aus dem Komponenten Schema diejenigen Teile extrahiert und dem Export Schema zuführt, die er in die Föderation einbringen möchte. Parallel dazu muß der entsprechende Filterungsprozessor entwickelt werden, der insbesondere dafür Sorge trägt, daß unerlaubte Zugriffe unterbunden werden. Bei der nun folgenden Schemaintegration wird jeweils aus einer beliebigen Anzahl von Export Schemen ein Föderatives Schema generiert. Der dafür erforderliche Konstruktionsprozessor muß wiederum Kommandos, die ans Föderative Schema gerichtet sind, in

Kommandos aufspalten, die an die jeweiligen auch physisch verteilten Export Schemen adressiert werden. Falls erforderlich können abschließend noch Externe Schemen definiert werden, die je für eine bestimmte Benutzergruppe bestimmt sind. Entsprechend muß ein Filterungsprozessor und, falls die Externen Schemen ein vom CDM abweichendes Datenmodell unterstützen sollen, ein Transformationsprozessor definiert werden.

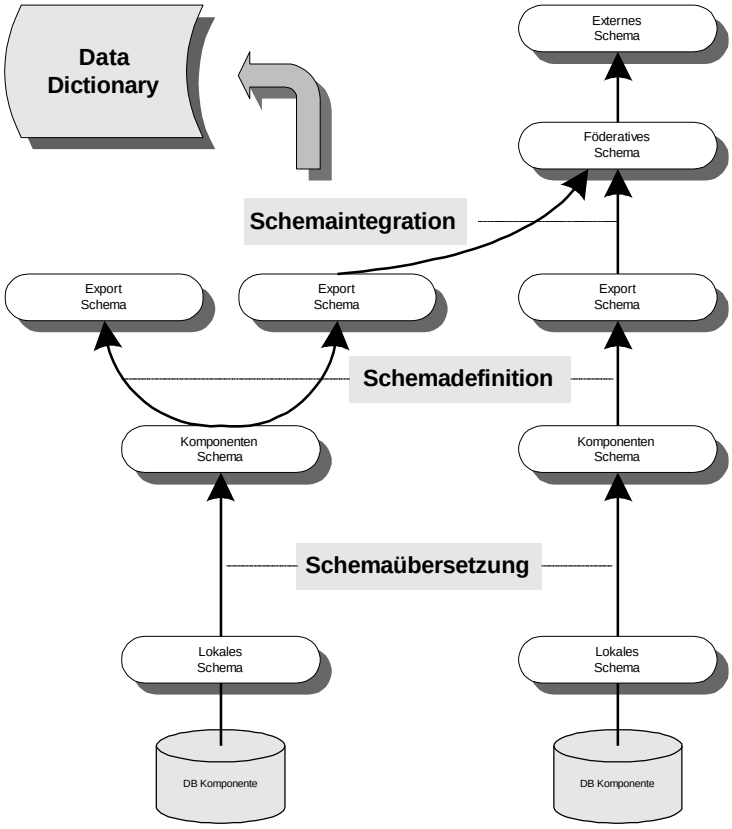


Abb. 4-1: Das Bottom-Up Verfahren

Ergebnis des Bottom-Up Verfahrens sind Prozessoren, Schemen sowie Abbildungen zwischen den Schemen (Mapping). Die beiden letztgenannten werden im Data Dictionary (Datenbankkatalog) gespeichert, der damit neben den Prozessoren den Kern einer Föderativen Datenbank bildet.

4.2 Schemaübersetzung

Eine Schemaübersetzung erfolgt bei der Transformation vom Lokalen Schema ins Komponenten Schema und eventuell auch bei der Transformation des Föderativen Schemas in ein Externes Schema. Es besteht einmal die Möglichkeit, daß man allgemeingültige Abbildungsregeln

formuliert, die beschreiben, wie Objekte aus dem Quell-Schema ins Ziel-Schema übersetzt werden. Ein Transformationsprozessor, der diese Regeln beherrscht, könnte für jede beliebige Übersetzung genutzt werden. Unterscheiden sich Quell- und Ziel-Schema dermaßen, daß man keine allgemeingültigen Regeln festlegen kann, so muß die Abbildung von einem Schema zum anderen explizit erfolgen. Das heißt, jeder einzelnen konkreten Abbildung einer bestimmten Quell-Ziel-Schema-Konstellation muß manuell eine Abbildungsregel zugewiesen werden. Diese explizite Transformation ermöglicht in vielen Fällen aber erst die ausreichende Berücksichtigung von semantischen Aspekten. Mangelt es den beteiligten Schemen an semantischer Beschreibungskraft, so wäre beispielsweise ein Automat bei der Übersetzung von Längenmaßen, deren Einheit er nicht kennt, überfordert.

In beiden Fällen besteht jedoch die Gefahr, daß bestimmte Übersetzungen nicht invers sind. Diese sind dadurch gekennzeichnet, daß zwar eine Transformation vom Quell-Schema zum Ziel-Schema möglich ist, jedoch bereitet die inverse Transformation vom Ziel-Schema zurück zum Quell-Schema Schwierigkeiten. Beispielsweise kann man aus dem Netto- und dem Tarabetrag den zugehörigen Bruttobetrag berechnen. Die dazu inverse Aufgabe, den Bruttobetrag in seine Netto- und Tarabestandteile aufzuspalten, ist ohne weitere Hilfsmittel, wie beispielsweise die Angabe das prozentualen Verhältnisses zwischen Brutto und Tara, nicht möglich. Für diese Probleme sind Lösungen zu finden.

4.3 Schema- und Datenheterogenität

Bei der Bildung eines Föderativen Schemas treten Probleme bezüglich der Heterogenität der zu integrierenden Export Schemen auf. Diese Probleme sind zweierlei Natur:

?? Unterschiede in der Datenstruktur

?? Unterschiede in der Semantik der Datenwerte

Obwohl die Schemaübersetzung für ein einheitliches Datenmodell sorgen soll, garantiert auch sie nicht für homogene Komponenten Schemata. Ein CMD (common data model) mit großer semantischer Beschreibungskraft kann viele Probleme bereits bei der Schemaübersetzung lösen. Jedoch wird die Schemaintegration nicht davon verschont, sich gleichfalls mit dem Thema zu

beschäftigen. Als vorweggenommenes Resümee kann man bereits jetzt feststellen, daß die Problematik dermaßen komplex ist, daß eine Automation der Schemaintegration in den allermeisten Fällen scheitern würde.

Won Kim und Jungyun Seo versuchen, in einem Beitrag der Zeitschrift "COMPUTER" [WJ91], eine umfassende Aufzählung und systematische Klassifikation aller möglichen Schema- und Datenkonflikte zusammenzutragen. Sie wählen als CDM das Relationale Modell, wobei sich ihre Ergebnisse aber auch auf andere Datenmodelle übertragen lassen. Die folgende Zusammenfassung gibt einen Überblick ihrer Erkenntnisse, die nachfolgend näher betrachtet werden sollen.

Schema Conflicts	Data Conflicts
I. Table-versus-table conflicts A. One-to-one table conflicts 1. Table name conflicts a) Different names for equivalent tables b) Same name for different tables 2. Table structure conflicts a) Missing attributes b) Missing but implicit attributes 3. Table constraint conflicts B. Many-to-many table conflicts II. Attribute-versus-attribute conflicts A. One-to-one attribute conflicts 1. Attribute name conflicts a) Different names for equivalent attributes b) Same name for different attributes 2. Default value conflicts 3. Attribute constraint conflicts a) Data type conflicts	I. Wrong data A. Incorrect-entry data B. Obsolete data II. Different representation for the same data (Same representation for different data) A. Different expressions B. Different units C. Different precisions

b) Attribute integrity-constraint conflicts B. Many-to-many attribute conflicts III. Table-versus-attribute conflicts	

Tab. 4-1: Schema- und Datenkonflikte

4.3.1 Schemakonflikte

Schemakonflikte sollen in diesem Rahmen nur kurz angesprochen werden. Es gibt zwei Ursachen für Schemakonflikte. Auf der einen Seite werden gleiche Informationen in unterschiedlichen Datenstrukturen, bestehend aus Tabellen und Attributen, gehalten. Beispielsweise kann eine Adresse bei der einen Datenbankkomponente als Attribut gespeichert werden. Die andere Komponente speichert die Adresse aufgeschlüsselt in einer gesonderten Tabelle. Zweitens kann dieselbe Struktur unterschiedlich spezifiziert werden. Das heißt, semantisch identische Tabellen oder Attribute haben unterschiedliche Namen, nutzen unterschiedliche Datentypen und befolgen uneinheitliche Integritätsregeln. Die Autoren kategorisieren in diesem Zusammenhang die Probleme in Tabelle-gegen-Tabelle-Konflikte, Attribute-gegen-Attribute-Konflikte und Tabelle-gegen-Attribute-Konflikte, wobei weiterhin zwischen der einfachen Konstellation einer 1 zu 1 Beziehung und der komplexeren N zu M Beziehung unterschieden wird.

4.3.2 Datenkonflikte

Auch bei strukturell gleichen, beziehungsweise strukturell angepaßten Schemen, können Konflikte auftreten, die ihre Ursache in inkonsistenten Datenwerten haben. Diese Datenkonflikte lassen sich in die Bereiche "unkorrekte Daten" (wrong data) und "unterschiedliche Repräsentation" (different representations for the same data) trennen.

4.3.2.1 Unkorrekte Daten

Mit falschen Daten bezeichnet man Datenwerte aus verschiedenen Datenbankkomponenten, die den gleichen Sachverhalt beschreiben und somit eigentlich identisch sein müßten, deren Datenwerte sich aber aus bestimmten Gründen unterscheiden. Der Grund für diese Diskrepanz

könnte zum einen darin liegen, daß Eingabe- beziehungsweise Erfassungsfehler gemacht wurden, oder daß eine Datenbank die aktuellen Änderungsdaten noch nicht berücksichtigt hat. Differieren beispielsweise die gespeicherte Gehaltshöhe eines Angestellten auf zwei Datenbankkomponenten, so kann entweder ein klassischer Tipfehler bei der letzten Eingabe der Gehaltserhöhung vorgekommen sein, oder eine Datenbank hat die Gehaltserhöhung aus organisatorischen Gründen noch nicht verbucht.

4.3.2.2 Unterschiedliche Repräsentation

Weitaus problematischer sind Fälle, in denen logisch gleiche Daten unterschiedlich repräsentiert werden. Zur genaueren Betrachtung unterscheiden wir hier die Punkte "unterschiedliche Ausdrucksweise" (different expressions), "unterschiedliche Einheiten" (different units) und "unterschiedliche Genauigkeit" (different precisions).

Unterschiedliche Ausdrucksweise

Hier wird der gleiche Sachverhalt durch eine unterschiedliche Datenrepräsentation dargestellt. So kann beispielsweise das Wort "Nordrheinwestfalen" auch durch die Buchstaben "NRW" abgekürzt werden. Die Zeichenketten "9390 Research Blvd. Ste. 220, Austin, TX" und "9390 Research Boulevard, Suite #220, Austin, Texas" geben zwar die gleiche Adresse an, sind aber in ihrer Form unterschiedlich. Ebenso kann die Schulnote "sehr gut" auch alternativ durch die Ziffer "1" substituiert werden. Nicht zuletzt können unterschiedliche Datentypen zur Repräsentation herangezogen werden. So kann die obengenannte Schulnote "1" entweder als numerischer Wert oder als Zeichen abgespeichert werden.

Unterschiedliche Einheiten

Numerische Datenwerte sind nur in Kombination mit ihren Einheiten zu verstehen. Obwohl man bestimmten Werten allgemein übliche Einheiten zuordnen kann, können Konfusionen auftreten, wenn die Einheit nicht mitgespeichert wird. Dieses tritt verstärkt auf, wenn kulturelle Grenzen überschritten werden. So kann beispielsweise eine Temperaturangabe von 10 Einheiten unterschiedliche Skalen zu Grunde liegen. So dürfen 10 Grad Celsius sicherlich nicht mit 10 Grad

Fahrenheit verwechselt werden. Vergleicht man Geldmengen unterschiedlicher Währung bei schwankenden Wechselkursen, so erhöht sich die Komplexität dieses Problems abermals.

Unterschiedliche Genauigkeit

Ein weiteres Problem tritt auf, wenn logisch gleiche Werte aus unterschiedlichen Wertebereichen gewählt werden und so in unterschiedlicher Genauigkeit vorliegen. So kann es beispielsweise zu Mißverständnissen führen, wenn eine Zeitangabe auf der einen Seite auf die Minute genau angegeben wird, beim anderen Wert, der den gleichen Sachverhalt beschreibt, aber jeweils auf eine volle viertel Stunde aufgerundet wird. In einem anderen Beispiel wird das Gewicht von Schiffen bei der einen Datenbank in die Gewichtsklassen „schwer“, „mittel“ und „leicht“ eingruppiert, die andere Datenbank verwendet die Klassen „superschwer“, „schwer“, „mittelschwer“, „mittel“, „mittelleicht“ und „leicht“.

4.4 Schemaintegration

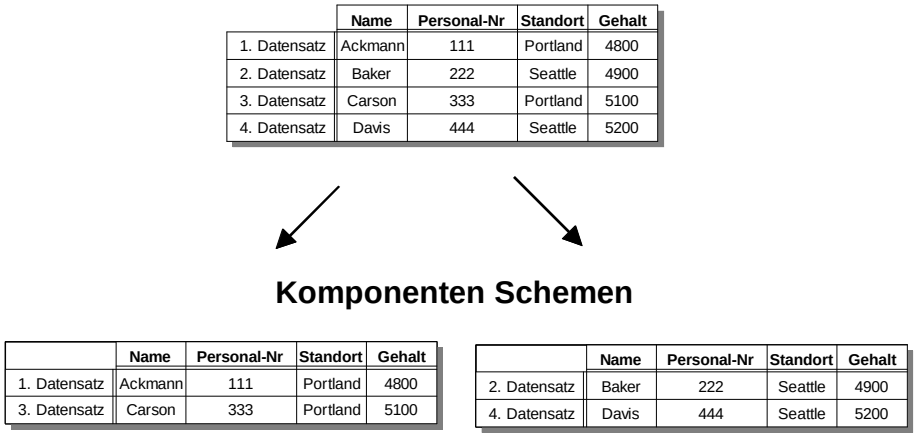
Die Schemaintegration ist eine komplexe Aufgabe, die vom Bearbeiter verlangt, daß er über Aufbau und Inhalt der zu integrierenden Export Schemen ausreichend informiert ist. Hilfreich ist dabei die Theorie über Fragmentierung und Replikation in verteilten Datenbanken. Nach deren Erläuterung soll ein Leitfaden für die Schemaintegration gegeben werden. Abschließend soll ein Modell vorgestellt werden, bei dem Datenkonflikte in bestimmten Spielräumen toleriert werden.

4.4.1 Fragmentierung und Replikation

Die Fragmentierung und Replikation[i] beschäftigt sich mit dem Problem der Aufteilung einer zentralen Datenbank auf verschiedene verteilte Datenbanken. Übertragen auf das Fünf-Schichten-Modell bedeutet dieses die Zerlegung des Föderativen Schemas in einzelne Komponenten Schemen (beziehungsweise Externe Schemen) der jeweiligen Datenbankkomponenten. Im eigentlichen Sinne ist dieses ein Problem des Top-Down-Entwurfs, das jedoch wegen seiner großen theoretischen Bedeutung auch hier betrachtet werden soll.

Horizontale Fragmentierung

[i]Bei der Horizontalen Fragmentierung werden ganze Datensätze (Tuples) des globalen Schemas, aufgrund bestimmter Regeln, den einzelnen Fragmenten zugewiesen. Im nachfolgenden Beispiel sollen die Mitarbeiter eines Unternehmens, je nach Standort ihres Arbeitsplatzes, unterschiedlichen Datenbankkomponenten zugeteilt werden. So enthält das eine Fragment nur Mitarbeiter aus Portland, das andere enthält nur Mitarbeiter aus Seattle. Geht man weiterhin davon aus, daß die Daten über Seattle-Mitarbeiter zum überwiegenden Teil nur in Seattle benötigt werden, und daß die Zweigstelle Portland hauptsächlich auf die Daten ihrer Mitarbeiter zugreift, so wird im Sinne der Datenverteilung das eine Fragment auf einer Datenbank in Portland und das andere Fragment auf einer Datenbank in Seattle gespeichert. Diese Zuweisung der Fragmente auf eine bestimmte Datenbankkomponente wird in der Literatur als Allocation (Zuteilung) bezeichnet.



	Name	Personal-Nr	Standort	Gehalt
1. Datensatz	Ackmann	111	Portland	4800
2. Datensatz	Baker	222	Seattle	4900
3. Datensatz	Carson	333	Portland	5100
4. Datensatz	Davis	444	Seattle	5200

	Name	Personal-Nr	Standort	Gehalt
1. Datensatz	Ackmann	111	Portland	4800
3. Datensatz	Carson	333	Portland	5100

	Name	Personal-Nr	Standort	Gehalt
2. Datensatz	Baker	222	Seattle	4900
4. Datensatz	Davis	444	Seattle	5200

Abb. 4-2: Horizontale Fragmentierung

Solange jeder Datensatz mindestens einem Fragment zugewiesen wurde, kann man mittels Vereinigungsoperation (UNION) das globale Schema aus den beteiligen Fragmenten rekonstruieren. Erlaubt man, daß ein Datensatz auch mehreren Fragmenten zugewiesen werden darf, so liegt eine Teilreplikation von Daten vor. Teilreplikation besagt, daß die gleichen Daten mehrfach in verschiedenen Fragmenten gespeichert sind. Im Extremfall wird jeder Datensatz in jedem Fragment gespeichert. Nun spricht man von Vollreplikation und ersetzt den Begriff

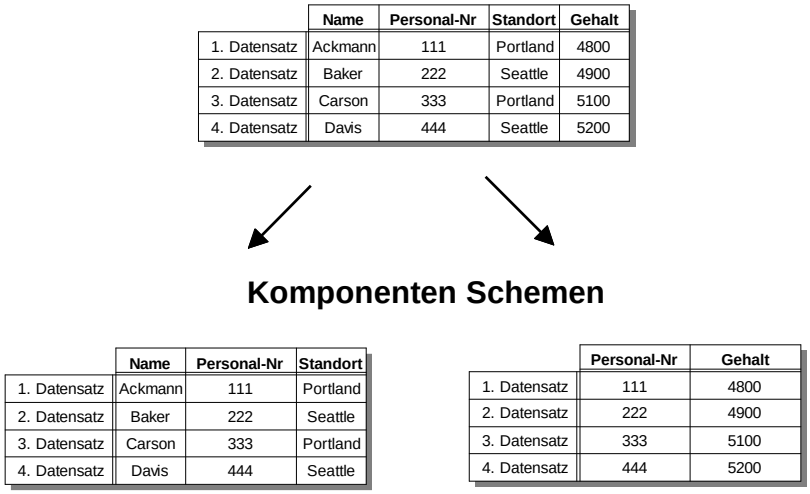
[i] Siehe [LJ95] ab Seite 82 - 4.5 DBAs Partition Files and Allocate the Resulting Fragments
Siehe [EN94] ab Seite 709 - 23.3 Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design

Fragment durch den Begriff Replikat[ii]. In unserem Beispiel würden beide Niederlassung direkten Zugriff auf alle Mitarbeiterdaten des Unternehmens, ohne Umweg auf die entfernte Datenbank des jeweils anderen, haben. Leseoperationen würden dadurch beschleunigt ausgeführt, da alle Daten lokal auf der eigenen Datenbankkomponente zur Verfügung stehen. Möchte man jede Art von Dateninkonsistenz verhindern, so müßte jede Schreiboperation auch auf dem Replikat der jeweils anderen Niederlassung durchgeführt werden, was zwangsläufig zu Leistungseinbußen führen würde. Bei anderen Ansätzen von verteilten Replikationen werden auch die problematischen Schreiboperationen nur lokal durchgeführt. In zeitlich bestimmten Abständen werden dann die jeweiligen Operationen auch beim jeweils anderen Replikat nachgeholt (Deferred Update). Dieser auch als Replikation bezeichnete Vorgang wird in Zeiten geringer Systemauslastung durchgeführt. In unserem Beispiel könnten nach Geschäftsschluß die Datenbanken von Seatte und Portland online geschaltet und repliziert werden. Wurde der gleiche Datensatz sowohl auf der einen als auch auf der anderen Replik geändert, so liegt ein sogenannter Replikationskonflikt vor. Dieser muß manuell oder über vordefinierte Strategien gelöst werden. Ebenso kann es gerade in einer heterogenen Föderation sinnvoll sein, daß man bestimmte Divergenzen beziehungsweise erlaubte Inkonsistenz zwischen einzelnen replizierten Daten zuläßt.

Vertikale Fragmentierung

Neben der Horizontalen Fragmentierung, die einer Projektions-Operation entspricht, gibt es die Vertikale Fragmentierung, die mit der Selektions-Operation vergleichbar ist. Hier wird ein globales Relationales Schema nicht zeilenweise sondern spaltenweise aufgeteilt. Folglich erhält jedes Fragment nur bestimmte Attribute, wobei ein Primärschlüssel-Attribut in mehreren Fragmenten vorhanden sein muß, damit man aus den Fragmenten per Join-Operation das Ausgangsschema wieder herleiten kann.

[i] Siehe [KH92] Seite 30 - 2.3 Arten der Datenfragmentierung
[ii] Siehe [KH92] Seite 106 bis 112 - 4.10 Update-Probleme bei replizierten Datenbanken



	Name	Personal-Nr	Standort	Gehalt
1. Datensatz	Ackmann	111	Portland	4800
2. Datensatz	Baker	222	Seattle	4900
3. Datensatz	Carson	333	Portland	5100
4. Datensatz	Davis	444	Seattle	5200

	Name	Personal-Nr	Standort
1. Datensatz	Ackmann	111	Portland
2. Datensatz	Baker	222	Seattle
3. Datensatz	Carson	333	Portland
4. Datensatz	Davis	444	Seattle

	Personal-Nr	Gehalt
1. Datensatz	111	4800
2. Datensatz	222	4900
3. Datensatz	333	5100
4. Datensatz	444	5200

Abb. 4-3: Vertikale Fragmentierung

In unserem Beispielunternehmen soll das Mitarbeiterschema nun nicht mehr nach Standorten der arbeitenden Angestellten zerlegt werden, sondern die sensiblen Gehaltszahlen sollen in einem gesonderten Schema, getrennt von den anderen Daten, gespeichert werden. Über das Primärschlüssel-Attribut Personalnummer kann das original Schema rekonstruiert werden.

Abgeleitete Fragmentierung

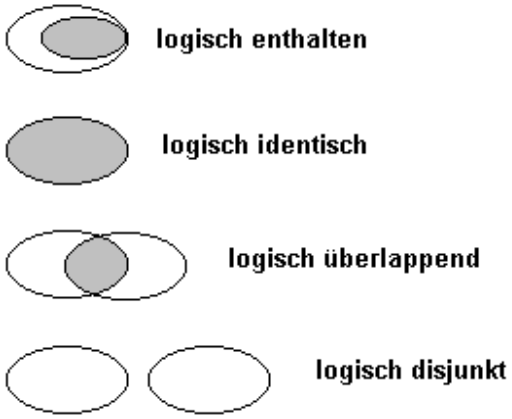
In der Praxis werden beide Fragmentierungsarten parallel angewandt. Somit spricht man von Abgeleiteter Fragmentierung (Derived Fragmentation, Mixed Fragmentation), wenn man beide Fragmentierungsarten nacheinander anwendet. Die Reihenfolge ist dabei irrelevant, da bei korrekter Anwendung jeglicher Kombination das Ausgangsschema wiederhergestellt werden kann.

4.4.2 Konstruktion eines Föderativen Schemas

Die Fragmentierung ist eine Technik, mit der man, ausgehend von einem Föderativen Schema, die Komponenten Schemen (bzw. Externe Schemen) der einzelnen Datenbankkomponenten konstruieren kann. Zu diesem Werkzeug des Top-Down-Verfahrens muß ein entsprechendes Pendant für das Bottom-Up-Verfahren gefunden werden. Ausgangspunkt sind nun die Komponenten Schemen beziehungsweise Externen Schemen der betreffenden Datenbankkomponenten, die zu einem Föderativen Schema zusammengefaßt werden müssen.

Nachfolgend sollen sechs Integrationsschritte beschrieben werden[i]. Auch hier soll als CDM der Einfachheit halber das Relationenmodell fungieren.

?? **Identifizierung von zusammengehörenden Tabellen, die Kandidaten für die Integration sind.**
Bei der Kandidatenkür können alle Kombination durchprobiert werden, was bei einer großen Anzahl von Tabellen sehr langwierig sein kann. Praxisrelevanter ist ein Vergleich von Relationen, die einen gleichen oder ähnlichen Namen besitzen, beziehungsweise deren Namen Synonyme sind. Danach sind entsprechende Tabellen zu begutachten, bei denen die Namen der Attribute eine Zusammenlegung favorisieren.



?? **Bestimmung der Datensatzbeziehung zwischen den gefundenen Tabellenpaaren.**
Hat man Relationen von verschiedenen Datenbankkomponenten gefunden, die logisch zueinander gehören, so ist die Dimension der Beziehung, die zwischen den Datensätzen beider Tabellen herrscht, zu ermitteln. Dabei kann einmal jede Zeile der einen Tabelle einer bestimmten Zeile der anderen Tabelle zugeordnet werden. Folglich ist die erste Tabelle logisch in der zweiten Tabelle enthalten (logical containment). Gilt dieses auch im umgekehrten Fall für die zweite Tabelle bezüglich der ersten Tabelle, so sind beide Tabellen logisch identisch (logical equivalence). Man bezeichnet eine Beziehung als logisch überlappend (logical overlap), wenn nur einige Datensätze aus der einen Tabelle in der anderen Tabelle wiederzufinden sind und umgekehrt. Falls kein Tupel sein logisches Pendant in der jeweils anderen Tabelle hat, so sind beide Relationen logisch disjunkt (logical disjoint). Nebenstehende Mengendarstellung zeigt alle vier Fälle im Überblick. Man beachte, daß die dunklen Flächen Datensätze symbolisieren, die logisch gleich in mindestens zwei Relationen verschiedener Datenbankkomponenten sind. Diese redundanten Daten werden durch die Integration zu Teilrepliken der Föderation.

?? **Festlegung welche Tabellenpaare zusammengefaßt werden.**
Alle Kandidatenpaare, deren Datensatzbeziehungen zu logisch redundanten Tupeln führen, werden zusammengefaßt. Nur bei Tabellen, deren Tupel logisch disjunkt sind, muß der Datenbankadministrator bestimmen, ob sie das gleiche Entity repräsentieren. Falls ja, so ist auch dieses Paar zusammenzufassen.

?? **Für jede zusammenzufassenden Tabellen ist zu bestimmen, wieviele neue Tabellen beim Integrationsprozeß zu bilden sind.**
Beim Single-table-approach werden die Tabellen durch eine Vereinigungsoperation (Union) zu einer einzigen Tabelle zusammengefaßt. Dazu sind die Attributnamen entsprechend

[i] Vergleiche [LJ95] Seite 75 ff. - 4.4 DBAs Integrate Schemas

abzustimmen sowie weitere Anpassungen durchzuführen. Der Multiple-table-approach erzeugt bei der Integration logisch gleicher Tabellen mehrere Tabellen im Föderativen Schema.

?? **Verschmelzung von Spalten mit redundanten Informationen.**
Enthält die neue Tabelle Attribute die zueinander redundant sind, wie beispielsweise das Geburtsjahr und das Alter von Angestellten, so kann man die Redundanz durch Löschen der überflüssigen Spalten beseitigen.

?? **Lösen von Datenkonflikten.**
Sind die Werte der logisch gleichen Datensätze nicht gleich, wie es eigentlich zu erwarten wäre, so liegt ein bereits beschriebener Datenkonflikt vor (*Siehe 4.3.2 Datenkonflikte*). Ähnlich wie beim verwandten Replikationskonflikt muß auch hier korrektiv eingegriffen werden.

4.4.3 Modell zur Spezifizierung von Datenbankbeziehungen

Kann der Datenbankentwickler beim Top-Down-Verfahren noch selber bestimmen, ob er bestimmte Daten mehrfach als Replikat hält, so muß er beim Button-Up-Verfahren die vorhandenen Gegebenheiten bei den einzubeziehenden Datenbankkomponenten akzeptieren. Diese sind nicht selten dadurch gekennzeichnet, daß logisch gleiche Daten auf unterschiedlichen Datenbankkomponenten doppelt gespeichert sind. Würde man die Redundanz dadurch beseitigen, daß in Zukunft nur eine Komponente mit der Datenhaltung betraut wird, so würde dieses gegen den Grundsatz von Föderativen Datenbanken verstoßen, die besagt, daß die beteiligten Datenbankkomponenten auch nach der Einbindung in eine Föderation ihre Selbständigkeit behalten. Diese Autonomität der einzelnen Datenbankkomponenten erschwert aber gleichzeitig die Formulierung von Integritätsbedingungen zwischen den Komponenten, die ihrerseits verhindern sollen, daß logisch gleiche Daten unterschiedliche Werte besitzen. Die daraus resultierenden Datenkonflikte sind zu lösen. Beispielsweise sind die Gewichte von Einbauteilen auf zwei Datenbankkomponente doppelt gespeichert. Komponente A benennt das Gewicht eines bestimmten Teiles mit 45 kg; Komponente B hat als Gewichtsangabe für das gleiche Teil nur 40 kg gespeichert. Zu bestimmen ist, ob im übergeordneten Föderativen Schema der Wert von Komponente A oder der Wert von Komponente B übernommen wird, oder ob man beispielsweise den Durchschnitt beider Werte in Höhe von 42,5 kg wählt. Hat sich der Datenbankentwickler für einen Wert entschieden, so stellt sich die Frage, ob dieser Wert auch von den entsprechenden Datenbankkomponenten übernommen werden soll, oder ob diese ihre ursprünglichen Werte behalten. Falls letzteres der Fall ist, so muß die Föderative Datenbank definieren, wie zu verfahren

ist, wenn der Wert modifiziert wird. Gerade in heterogenen Umgebungen ist nicht zu erwarten, daß logisch gleiche Werte die exakt gleiche Beschreibung haben. Vielmehr muß man hier viel weichere Kriterien anlegen, die bestimmen, wann zwei Werte dermaßen voneinander abweichen, daß man von einem Datenkonflikt sprechen kann.

Marek Rusinkiewicz, Amit Sheth und George Karabatis beschreiben ein Modell [RS91], mit dem weiche Konsistenzbedingungen zwischen logisch gleichen Daten unterschiedlicher Datenbanken formuliert werden können. Weiche Konsistenzbedingungen bedeutet in diesem Zusammenhang, daß man nicht exakte Identität verlangt, sondern Divergenzen in bestimmten Bandbreiten zuläßt. Mit anderen Worten wird Inkonsistenz zwischen gleichen Daten in genau umrissenen Grenzen gebilligt. Kann zwischen zwei Werten aber dennoch die erlaubte Inkonsistenz nicht eingehalten werden, so erlaubt das Modell die Spezifizierung von Konsistenz-Wiederherstellungsprozeduren. Diese spezifizieren wie man das Datenpaar wieder in einen konsistenten Zustand bringt.

Jede Datenpaarbeziehung wird formell durch ein Datenabhängigkeits-Bezeichner beschrieben (D data dependency descriptor). Der Bezeichner ist ein fünfelementiges Tupel, indem neben dem Quelldatenobjekt (S source data object) und dem Zieldatenobjekt (U target data object) die Spezifikation des Abhänigigkeitsprädikats (P Specification of the dependency predicate), die Spezifikation der gegenseitigen Konsistenz (C Specification of mutual consistency) sowie die Spezifikation der Konsistenzwiederherstellungs-Prozedur (A Specification of consistency restoration procedures) enthalten ist.

$$\mathbf{D = < S , U , P , C , A >}$$

Diese Beschreibung einer Datenbeziehung mittels eines Datenabhängigkeits-Bezeichners ist unidirektional. Das heißt, ein neuer Tupel muß konstruiert werden, wenn Quelle und Ziel vertauscht werden.

4.4.3.1 Spezifizierung des Abhängigkeitsprädikats

(**P** DEPENDENCY PREDICATE)

Die Spezifizierung des Abhängigkeitsprädikats soll hier nur kurz angesprochen werden. Das dafür zuständige Prädikat P ist ein boolscher Ausdruck, der die Beziehungen zwischen den

Datenobjekten der Quelle (S) und des Zieles (U) festlegt. Das Prädikat wird dabei mit Operatoren der relationalen Algebra sowie der erweiterten relationalen Algebra gebildet. Nachfolgendes Beispiel zeigt ein Prädikat, das die Abhängigkeit einer Datenbank (EMP), die sämtliche Einkommen der Mitarbeiter eines Unternehmens enthält, mit einer Datenbank (DEPT_SAL), die das durchschnittliche Einkommen pro Abteilung verwaltet, beschreibt.

$$\mathbf{P:\quad DEPT\_SAL = ?_{D\#,\ AVG(Sal)}\ (EMP)}$$

Nach der Gruppierung der Mitarbeiterdatenbank EMP bezüglich der Abteilungsnummer D#, wird das Durchschnittsgehalt AVG(Sal) berechnet und der anderen Datenbank gegenübergestellt. Auf eine genaue Beschreibung der Syntaxs soll im Rahmen dieser Arbeit verzichtet werden.

4.4.3.2 Spezifizierung der Konsistenzbedingungen

(**C** MUTUAL CONSISTENCY REQUIREMENTS)

Entscheidender für unsere Belange ist die Spezifizierung der Konsistenzbedingungen, also der exakten Beschreibung, wann bestimmte Datenpaare als Konsistenz oder Inkonsistenz gelten sollen. Logische Operatoren ? ? ? verknüpfen einzelne Konsistenzbedingungen c_i, die man in zeitliche Konsistenzbedingungen (temporal consistency terms) und der Datenintegritätbedingungen (data state consistency terms) unterteilen kann, zum Prädikat C. Zeitliche Konsistenzbedingungen verlangen in einer fest definierten Zeitspanne die Einhaltung von C. Anders ausgedrückt erlaubt man an allen anderen Zeitpunkten, daß sich die betroffenen Datenwerte beliebig voneinander unterscheiden dürfen. Dieses hat praktische Bedeutung, wenn beispielsweise die eine Datenbank später aktualisiert wird als die andere. Die Datenintegrität widmet sich dem eigentlichen Problem: den Datenkonflikten. Wiederum ist eine Unterscheidung in Konsistenzbedingungen, die sich mit den eigentlichen Ausprägungen der Datenwerte beschäftigen, und Bedingungen, die nach bestimmten Benutzer- oder Systemoperatioen einzuhalten sind, möglich. Als Benutzer- oder Systemoperationen kommen dabei Hinzufügen (insert), Löschen (delete) und Modifizieren (modify) in Frage. Zur genaueren Erläuterung werden nachfolgend diverse Beispiele für genannte Kategorien von Konsistenzbedingungen genannt. Auf eine Beschreibung der dazu verwendeten Syntaxs wird verzichtet, da sie sich aus den Beispielen ergibt.

Zeitliche Konsistenzbedingungen

?? Konsistenz muß zu einem bestimmten Tagesdatum beziehungsweise zu einem bestimmten Zeitpunkt gewährleistet sein.

@ 9:00	Konsistenz muß während des Tages um 9:00 Uhr gewährleistet sein.
on May 27, 1991	Konsistenz muß während des ganzen 27 Mai 1991 gewährleistet sein.

?? Konsistenz muß vor oder nach einem bestimmten Zeitpunkt gewährleistet sein.

! 8:00	Konsistenz muß vor 8:00 Uhr gewährleistet sein.
25-Aug-1991 !	Konsistenz muß nach dem 25 August 1991 gewährleistet sein.

?? Konsistenz muß innerhalb eines definierten Zeitintervalls gewährleistet sein.

? (9:00 ? 17:00)	Konsistenz muß zwischen 9:00 Uhr und 17:00 Uhr gewährleistet sein.
? (10-Jun-1991 @ 17:00 ? 11-Jun-1991 @ 8:00)	Konsistenz muß zwischen dem 10 Juni 1991 17:00 Uhr und dem 11 Juni 1991 8:00 Uhr gewährleistet sein.

?? Konsistenz muß periodisch zu einem bestimmten Zeitpunkt wieder hergestellt sein.

? (day @ 12:00)	Jeden Tag um 12:00 Uhr muß die Konsistenz gewährleistet sein.
? (month on 15)	Am 15 jeden Monats muß die Konsistenz gewährleistet sein.

?? Konsistenz muß nach einer bestimmten Dauer der Konsistenzverletzung wieder hergestellt werden. Folglich wird hiermit die Dauer von Inkonsistenz begrenzt.

?* (8 hour)	Maximal nach 8 Stunden Inkonsistenz muß diese wieder beseitigt werden.

Konsistenzbedingung bezüglich konkreter Datenwerte

?? Eine bestimmte Anzahl von Datenwerten dürfen Inkonsistent sein, ohne daß die Gesamtkonsistenz verletzt wird.

10% (DB1.EMP)	Stimmen über 10% der Gehaltsdaten aus DB1.EMP nicht mit den Gehaltsdaten der abhängigen Datenbasis überein, so ist die gewährte Inkonsistenz verletzt.

?? Überschreitet eine Änderungsoperation ein bestimmtes Limit, so ist eine eventuelle Inkonsistenz zu beseitigen.

? EMP.Salary > 500	Überschreitet eine verbuchte Einkommenserhöhung den Wert 500, so muß eventuelle Inkonsistenz beseitigt werden.

?? Eine Konsistenzbedingung können über aggregierte Datenwerte formuliert werden.

? (AVG (EMP.Sal)) > 50	Überschreitet das durchschnittliche Einkommen aller Mitarbeiter den Wert 50, so muß eventuelle Inkonsistenz beseitigt werden.

Konsistenzbedingung bei bestimmten Benutzer- oder Systemoperationen

?? Nach einer bestimmten Anzahl von Operationen auf die Quelldaten, die auch näher bestimmt werden können, muß erneut Konsistenz gewährleistet werden.

10 updates on R R? S	Nach 10 Operationen auf die Quelldaten muß erneut Konsistenz hergestellt werden..
2 delete on R R? S	Nach 2 Löschoperationen auf die Quelldaten muß erneut Konsistenz hergestellt werden.

?? Vor einer bestimmten Operation, muß die Konsistenz wiederhergestellt werden.

Read on U	Bevor ein bestimmtes Datum ausgelesen wird, ist die eventuelle Inkonsistenz zu beseitigen.

?? Die Anzahl der Transaktionen, die ohne nachfolgender Konsistenzwiederherstellung ausgeführt werden dürfen, ist begrenzt.

10 sales	Es dürfen 10 Transaktionen mit dem Namen „sales“ ausgeführt werden, ohne daß die Erlaubte Inkonsistenz verletzt wird.

?? Es können Transaktionen bestimmt werden, denen eine Konsistenzwiederherstellung vorausgehen oder folgen muß.

!calculate_payroll_checks	Bevor die Transaktion mit dem Namen „calculate_payroll_checks“ aufgerufen wird, muß Konsistenz gewährleistet sein..
take_inventory!	Nach Ausführung der Transaktion mit dem Namen „take_inventory“ muß die Konsistenz wiederhergestellt werden.

4.4.3.3 Spezifizierung der Konsistenzwiederherstellungsprozedur

(**A** CONSISTENCY RESTORATION PROCEDURES)

Ist die Konsistenzbedingung C nicht erfüllt, so wird der letzte Punkt des Datenabhängigkeits-Bezeichners ausgeführt. Hinter A verbirgt sich eine oder auch mehrere Konsistenzwiederherstellungsprozeduren, mit der die in Beziehung stehenden Daten zurück in konsistenter Form gebracht werden können. Welche Prozedur eingesetzt wird, bestimmt die Wiederherstellungsbedingung C_R (restoration condition), die wie das Konsistenzprädikat C durch logische Verknüpfung (? ? ?) gebildet wird, ohne daß beide zwingend identisch sein müssen.

```
when C_R1 do prodedure name [as execution mode ]
...
when C_Rn do procedure name [as execution mode]
otherwise  default procedure name [as execution mode ]
```

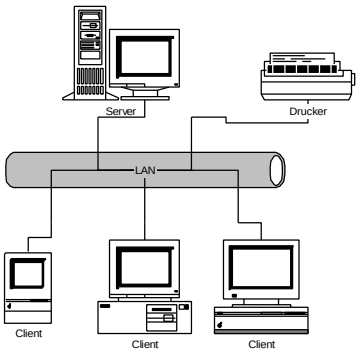
So kann beispielsweise definiert werden, daß, wenn der Datenwert X um 10% größer als der Datenwert Y ist, die Wiederherstellungsprozedur U ausgeführt wird. Falls jedoch der Datenwert X um 5 Einheiten kleiner als der Datenwert Y ist, wird die Wiederherstellungsprozedur V angestoßen. Jeder Prozedur kann zusätzlich ein Ausführungsmodus (execution mode) zugeordnet

werden, der Eigenschaften des Ablaufs detaillierter festlegt. Im Rahmen dieser Arbeit soll hierauf jedoch nicht näher eingegangen werden.

5 Client-Server Architekturen

Die Client-Server-Architekturen[i] stellen eine der wichtigsten Basiskomponenten für die Verteilung von Datenbanken dar. Ein Netzwerk mit dieser Architektur besteht aus einem oder mehreren leistungsfähigen Rechnern als Server und dezentralen Arbeitsplatzrechnern, den Clients, die mit lokaler Eigenintelligenz ausgestattet sind. Die Kopplung aller Rechner übernimmt ein LAN. Das dadurch entstandene Netzwerk kann schnell und flexibel um einige weitere Rechnerkomponenten erweitert werden. Der große Vorteil gegenüber der Mainframe-Terminal Architektur liegt in der Eigenintelligenz der Clients. Auf ihnen können Applikationen laufen, die über eine hochauflösende graphische Benutzerschnittstelle verfügen. Ein Charakteristikum der Client-Server-Architektur ist, daß der Bedarf an Rechenzeit und Speicherplatz zwischen Client und Server aufgeteilt wird (Ressourcensharing). So übernimmt bei einem Datenbankprogramm als Frontendapplikation der Client nur die Darstellung der Benutzeroberfläche mit den zugehörigen Bildschirm-Ein-/Ausgaben. Auf dem Backend laufen alle rechenzeitintensiven Aufgaben, wie beispielsweise das Suchen und Sortieren von Datensätzen. Die Frontendapplikation sendet den Befehl für die auszuführende Operation an den Datenbankserver. Das Ergebnis dieser Operation wird über das Netz an den entsprechenden Client zurückgeschickt. Der Datenbankserver vereint also hohe Sicherheit und Integrität der Daten sowie zentrale Kontrolle und gemeinsame Nutzung leistungsfähiger Ressourcen, z.B. CPU und Plattenspeicher, mit der Flexibilität und Benutzerfreundlichkeit von PC-Arbeitsplätzen oder Workstations.

Die grobe Arbeitsweise eines Client-Server-Systems kann man dadurch beschreiben, daß ein Client Anfragen an den Server stellt und dieser daraufhin Resultate zurückgibt. Im Falle eines SQL-Datenbank-Servers besteht die Anfrage aus SQL-Befehlen und das Resultat aus der Ergebnismenge zusammen mit Status- und Fehlercode. Bei der Kommunikationsverbindung ist eine Server-Schnittstelle, eine Client-Schnittstelle und ein Protokoll involviert.



[i] Vergleiche [KH92] Seite 137 - 5.5.2 Client-Server-Architektur

Eine Client-Applikation greift über ihre Clientschnittstelle auf die Serverschnittstelle des Servers zu. Der Daten- und Befehlstransport läuft über das dazwischenliegende Protokoll. Als Clientschnittstelle bezeichnet man die API, die sich aus Aufrufkonventionen, Prozedurdefinitionen und Syntax und Semantik der Sprache zusammensetzt. Das Protokoll besteht aus Kommandos und Argumenten, die als Bits über das Kommunikationsmedium von einer Schnittstelle zur anderen Schnittstelle fließen. Die Server-Schnittstelle ist das Gegenstück zur Client-Schnittstelle. Anstelle Prozeduraufrufe der Client-Anwendung entgegenzunehmen, muß sie die ankommenden Kommandos aufnehmen und zur Server-Applikation weiterleiten. Früher versteckten Datenbankverwaltungssysteme ihre Server-Schnittstelle, so daß ein freier Zugriff nicht möglich war. Später wurde sie als seperates Produkt angeboten, so daß Kunden sie erwerben und eigene Client-Server-Anwendungen entwickeln konnten. Diese propritären Schnittstellen sind sehr leistungsfähig, da sie alle Besonderheiten des Datenbanksystems kennen und berücksichtigen können. Dieses bereitet solange keine Probleme, wie nicht unterschiedliche Systeme von unterschiedlichen Herstellern im Einsatz sind. Beispielsweise soll eine Anwendung, die für eine Oracle Datenbank geschrieben wurde, nachträglich auch mit einer IBM Datenbank laufen. Der Anwendungsentwickler müßte beide Schnittstellen in sein Programm integrieren und die Logik des Programms soweit erweitern, daß je nach anzusprechender Datenbank die entsprechende Schnittstelle aufgerufen wird. Die Probleme und Kosten vergrößern sich für jede weitere zusätzliche Datenbank, die ebenfalls unterstützt werden soll.

Im nachfolgenden Unterkapitel sollen kurz Lösungsansätze für offene Verbindungen aufgezählt werden. Danach soll die Rolle von SQL in Client-Server Umgebungen näher untersucht werden. Abschließend wird Microsofts ODBC als Vertreter einer Call-Schnittstelle vorgestellt.

5.1 Methoden für offene Verbindungen

Man kennt drei Methoden, um eine offene Verbindung von Client und Server zu ermöglichen[i]. Die Einheitliche-Schnittstelle konzentriert sich auf eine allgemein standardisierte Client oder Server-Schnittstelle. Bei der Gateway Methode garantiert eine Datenbank mit Gatewayfunktion die offene Verbindung. Die dritte Methode vereinheitlicht, im Sinne eines Einheitlichen-Protokolls, die Kommandos des Protokolls, welches zwischen Client und Server eingesetzt wird. Konkrete

Lösungen sind jedoch nicht reine Vertreter einer Methode, sondern adaptieren verschiedene Ansätze.

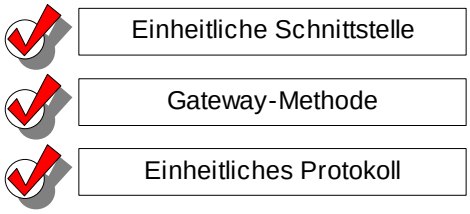


Abb. 5-1: Methoden für offene Verbindungen

5.2 SQL als offenes Verbindungsprotokoll

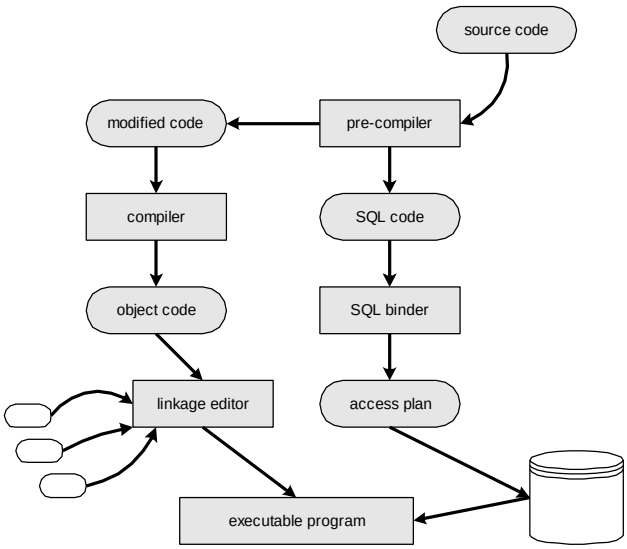
Die Structured Query Language (SQL) wurde ursprünglich von IBM entwickelt und stellt den De-facto-Marktstandard für die interaktive Arbeit mit einer Datenbank dar[ii]. Sie ist mengenorientiert und deskriptiv. Das heißt, es wird im Gegensatz zu prozeduralen Sprachen nicht beschrieben, wie ein Ergebnis zu erzielen ist, sondern WAS in der Lösungsmenge sein soll. Die aus dem Großrechnerbereich stammende Sprache umfaßt eine Datendefinitionsteil DDL, dessen Aufgabe in der Definition des Modellschemas liegt, und eine Datenmanipulationsteil DML, der sich mit der eigentlichen Manipulation der Daten befaßt. SQL wurde ursprünglich als einfache Sprache für interaktive Ad-hoc-Datenbankabfragen entwickelt. Ihre weite Verbreitung und weit fortgeschrittene Standardisierung erweckte Hoffnungen sie auch im Umfeld von Datenbankapplikationen einzusetzen. Es fehlen aber Variablenkonzepte und die elementaren Konstrukte der prozeduralen Programmierung, wie Wiederholungen (z.B. REPEAT ... UNTIL) und Bedingungen (IF ... THEN ... ELSE, CASE). Zweckmäßiger ist es, SQL in Programmiersprachen einzubinden. Ein Problem bei der Verwendung von SQL in Programmiersprachen der 3. Generation sind die Ergebnistabellen der SQL-Mengenoperationen (Select, Join, Union, etc.). Sie lassen sich nicht unmittelbar weiterverarbeiten. Zwar bietet SQL mit UPDATE, INSERT und DELETE die Möglichkeit, eine Menge von Datensätzen zu verändern.

[i] Siehe [HR93] Seite 147 bis 160 - 6.4 The three basic approaches
[ii] Siehe [HR93] Seite 38 bis 40 - 2.3 SQL as a connectivity language
Siehe [HH86] Seite 506 bis 510 - 3.2.2.4 Sprachen des Datenbanksystems
Siehe [EN94] Seite 185 ff. - Kapitel 6 SQL - A relational database language
Siehe [AA90]

Umgekehrt kann man aber auf die Ergebnistabellen von SQL-Mengenoperationen mit Befehlen von 3GL-Sprachen nicht zugreifen. Abhilfe schafft das Cursorkonzept von SQL. Dieses erlaubt, nach der Definition einer Ergebnismenge, mit Hilfe eines sogenannten Cursors satzorientiert durch die Menge zu navigieren und Werte zu modifizieren. Hierbei gibt es unterschiedliche Konzepte, mit denen SQL in 3GL-Sprachen verwendet werden kann.[i]

5.2.1 Embedded SQL

Embedded SQL ermöglicht die Verwendung von SQL-Statements in einer Wirtssprache (host language) und ist zur Zeit noch die wichtigste Form der Einbindung von SQL in die Anwendungsprogrammierung. So werden für SQL/92 Ada, C, Cobol, Fortran, Pascal und PL/1 als Wirtssprache unterstützt. SQL-Deklarationen und -Statements gehen direkt in den Quelltext der Wirtssprache ein. Embedded SQL wird in der Hostsprache durch ein vorangestelltes EXEC SQL identifiziert. Ein Precompiler setzt dann SQL-Statements in Funktionsaufrufe um.



Quelle: [HR93] Seite 42 Figure 2.5

Abb. 5-2: Embedded SQL

[i] Siehe HR93] ab Seite 40 - 2.4 Database programming interfaces

5.2.2 Dynamic SQL

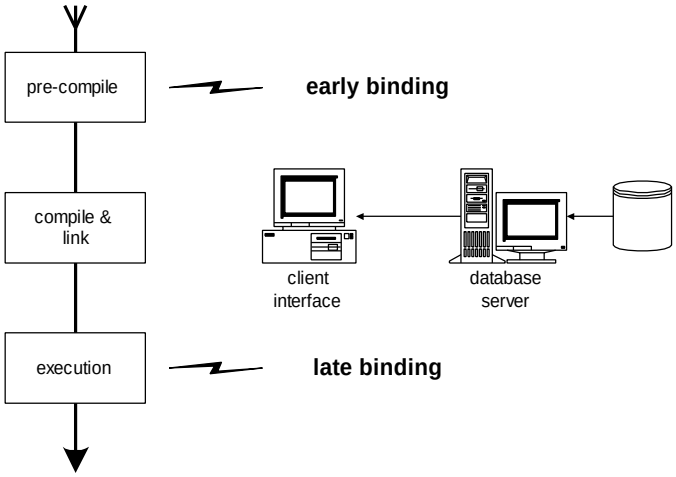
Dynamisches SQL ist lediglich eine konzeptionelle Erweiterung von Embedded SQL. SQL-Statements werden erst zur Laufzeit generiert und ausgeführt. Datenbankoperationen müssen deshalb nicht im voraus feststehen. Dadurch verschlechtert sich allerdings auch die Performance, da der Syntax-Check erst zur Laufzeit erfolgt. Dynamisches SQL ermöglicht aber eine wesentlich flexiblere Programmierung von Anwendungssystemen, was diesen Nachteil mehr als aufwiegt.

5.2.3 Call-Schnittstelle

Die modernste und zukunftsträchtigst Möglichkeit, SQL in 3GL-Sprachen zu verwenden, bieten Call-Schnittstellen. SQL-Statements werden in Funktionsaufrufe als String-Parameter übergeben und zur Laufzeit vom DBMS interpretiert. Diese spezielle Form des dynamischen SQL wird im Rahmen des SQL3-Standards als SQL/CLI (Call Level Interface) normiert. Die verbreitetste Implementierung einer Call-Schnittstelle ist wohl zur Zeit ODBC unter Microsoft Windows.

5.2.4 Gegenüberstellung der drei Konzepte

Man unterscheidet statisches SQL, bei dem der Code während der Programmerstellung compiliert und gebunden wird, und dynamisches SQL, bei dem der resultierenden Zugriffspfad erst zur Zeit der Programmlaufs ermittelt wird. Die Begriffe early und late binding werden auch in diesem Zusammenhang verwendet. Statisches SQL ist leistungsfähiger aber weniger flexibel, dynamisches SQL ist umgekehrt weniger leistungsfähig dafür aber flexibler. Beide Ablaufarten lassen sich sowohl mit Embedded SQL als auch mit Call-Level SQL realisieren, wobei Embedded SQL dem statisches SQL und Call-Level SQL dem dynamischen SQL entgegen kommt. So liegt es nah, generische Tools mittels einer Call-Schnittstelle zu realisieren.



Quelle: [HR93] Seite 44 Figure 2.7

Abb. 5-3: Early und late binding

Der wesentliche Vorteil der Call-Schnittstelle gegenüber Embedded SQL besteht darin, daß alle SQL-Befehle über C-Funktionsaufrufe realisiert werden. Damit funktioniert der Datenbankzugriff sofort in allen Programmiersprachen, die C-Funktionsaufrufe unterstützen. Im Gegensatz dazu ist man bei Embedded und Dynamic SQL auf einen Precompiler angewiesen, der für jede Programmiersprache anders ausfällt.

5.2.5 Die Call-Schnittstelle ODBC

Der bekannteste Vertreter einer Call-Schnittstelle ist ODBC (Open Database Connectivity) als zentraler Bestandteil von Microsofts WOSA[i] (Windows Open Services Architecture). Die ODBC-Schnittstelle ist eine Funktionsbibliothek, aus der sich ein Anwendungsprogramm bedienen kann. Zielsetzung ist der einheitliche Zugang zu Informationen, die in unterschiedlichen Datenbanken gespeichert sind. Eine Anwendung soll so über eine einzige Schnittstelle Zugriff auf ein breites Spektrum von Datenbanksystemen erhalten. Die ODBC-Spezifikation setzt sich aus folgenden Punkten zusammen:

?? Ein Bibliothek von ODBC Funktionen, die es einer Anwendung erlaubt eine Verbindung zu einem Datenbankmanagementsystem herzustellen, SQL-Statements auszuführen und Resultate zu empfangen.

[i] Vergleiche [MS95] SDKs/ODBC SDK 2.10/Programmer's Reference
Vergleiche[WB95]

?? Die SQL-Syntax

?? Ein standardisierte Menge von Fehlercodes.

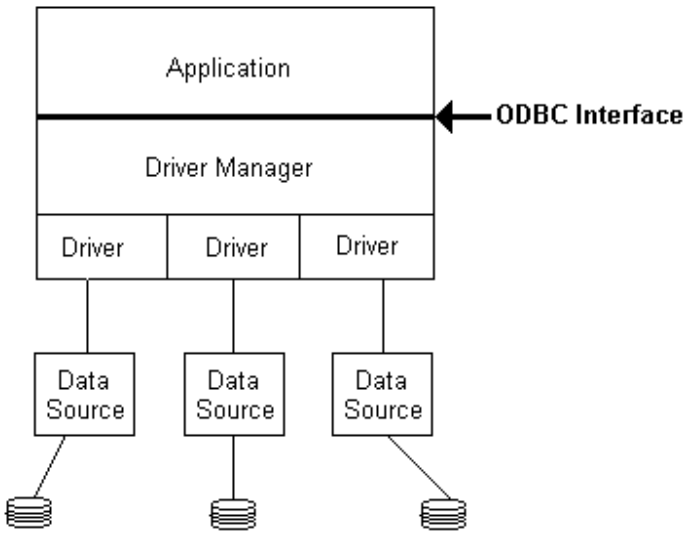
?? Verbindungs- und Anmeldungsroutinen für ein Datenbankmanagementsystem.

?? Datentyp-Repräsentanten

Mit ihnen schirmt die Schnittstelle das Anwenderprogramm und damit auch den Benutzer von den Unterschieden der Datenbank- und Netzwerksoftware ab.

5.2.5.1 Komponenten einer ODBC Verbindung

Bei einer ODBC-Verbindung sind mehrere Komponenten beziehungsweise Schichten eingebunden.



?? **Anwendung (Application)**
Ruft ODBC-Funktionen auf, die SQL-Befehle übersenden und Resultate empfangen.

?? **Treibermanager (Driver Manager)**
Lädt ODBC-Treiber auf Anweisung der Anwendung.

?? **Treiber (Driver)**
Leitet die SQL-Befehle an die Datenquellen weiter und gibt umgekehrt die Resultate an die Anwendung zurück. Falls notwendig, werden die weiterzuleitenden Befehle an die Eigenart der adressierten Datenquelle angepaßt. Auch müssen eventuell Datenformate und Fehlercodes transformiert werden.

?? **Datenquelle (Data Source)**
Als Datenquelle bezeichnet man die Kombination aus Datenbankmanagementsystem, Betriebssystem und Netzwerk.

5.2.5.2 Treiberrangstufen

Den Treibern können anhand der Anzahl der beteiligten Kommunikations-Instanzen Rangstufen (Tier) zugewiesen werden.

?? **Single-Tier Treiber**
Treiber dieses Typs arbeiten direkt auf dem Dateisystem und realisieren den Funktionalitätsumfang einer SQL-Engine. Das SQL-Statement wird interpretiert, in Datei-Operationen transformiert und diese dann ausgeführt. Beispiele sind Treiber für xBase-Dateien und Dateien in Spreadsheetformaten, welche im Filesystem des Clients liegen.

?? **Two-Tier Treiber**
Diese Art von Treibern kommuniziert direkt mit einem Datenbankserver. Für relationale Datenbankserver reduziert sich die Funktionalität des Treibers im wesentlichen auf die Abwicklung der Kommunikation zwischen Client und Server. Für nichtrelationale Datenbankserver muß zusätzlich eine SQL-Engine die SQL-Statements in das Zielformat des Datenbankservers konvertieren.

?? **Gateway (Three-Tier-Treiber)**
In dieser Architektur reicht der Treiber die SQL-Statements an einen Gateway-Prozeß weiter, welcher diese wiederum an einen Host (-Prozeß) weiterleitet.

5.2.5.3 API und SQL Konformitätsebenen

Ideal wäre es, wenn jeder Treiber die selben ODBC-Funktionen und jede Datenquelle die selben SQL-Anweisungen unterstützen würde. Da dieses nicht der Fall ist, definiert die ODBC-Schnittstelle Anpassungsstufen, die einen Treiber eine bestimmte Kategorie zuordnet. Ein Treiber muß alle Funktionalitäten einer API beziehungsweise SQL Anpassungsstufe unterstützen, damit er in dieser Stufe zugeordnet bleiben darf. Die Existenz von darüber hinausgehenden Funktionen, können beim Treiber über bestimmte ODBC -Aufrufe abgefragt werden.

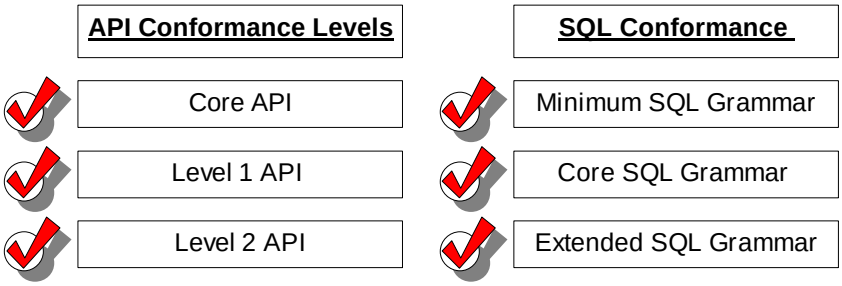
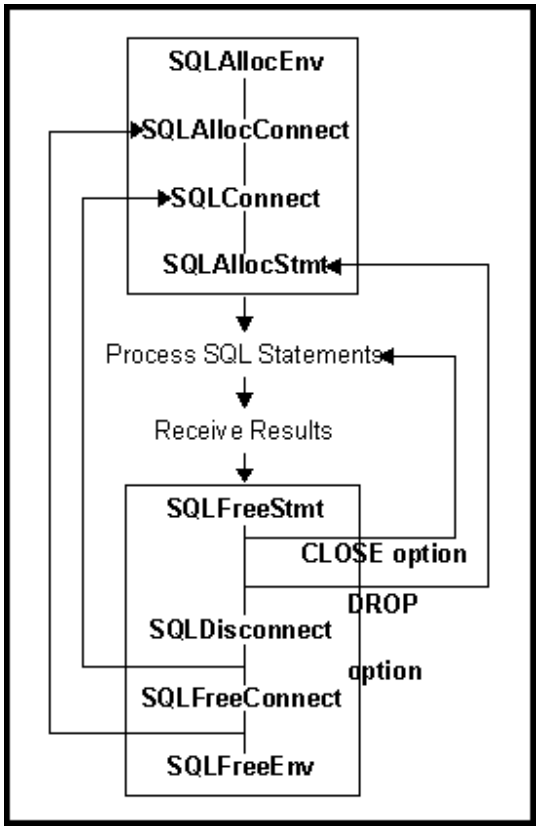


Abb. 5-4: ODBC Anpassungsstufen

Bei den API Anpassungsstufen unterscheidet man Core API, Level 1 API und Level 2 API. Wobei jede höhere Stufe die Funktionalität der niedrigeren Stufe mit einschließt und darüber hinaus erweiterte Funktionalitäten fordert. Das gleiche Prinzip verwenden die SQL Anpassungsstufen Minimum SQL Grammar, Core SQL Grammar und Extended SQL Grammar. In ihnen werden zusätzlich auch die zu unterstützenden Datentypen festgelegt. Diese Anpassungsstufen sind nicht mit der ODBC-Standardisierung zu verwechseln, die zur Zeit als Version 2.0 aktuell ist. Vielmehr sollen sie das auch in Zukunft vorherrschende unterschiedliche Funktionsangebot der diversen Datenquellen widerspiegeln und berücksichtigen.

5.2.5.4 Die Programmierung der ODBC-API

[i]Um überhaupt Befehle an einen SQL-Server schicken zu können, muß zunächst eine Verbindung zur betreffenden Datenbank über einen ODBC-Treiber aufgebaut werden. Dies geschieht dreistufig über ein Environment-Handle, Connection-Handles und Statement-Handles. Jede Applikation benötigt für den Datenbankzugriff über ODBC-Treiber ein Environment Handle, das die Verbindung zum jeweiligen Treiber verwaltet. Für jede Verbindung zu einer Datenquelle - sprich SQL-Server - benötigt man außerdem ein Connection Handle. Mit ODBC ist es problemlos möglich, aus einem Anwendungsprogramm auf mehrere SQL-Server zuzugreifen. Zur Ausführung eines SQL-Befehls muß zusätzlich ein Statement Handle alloziert werden. Das Statement Handle identifiziert einen abgesetzten SQL-Befehl und dient unter anderen dazu, einem SQL-Befehl dynamische Parameter zu übergeben beziehungsweise Speicherbereiche für diese bereitzustellen oder Cursornavigationsoperationen für ein bestimmtes Select-Statement auszuführen. Es ist möglich, mit einem Cursor nicht nur auf einzelne Datensätze, sondern auch blockweise auf mehrere Datensätze zuzugreifen. Beim Öffnen eines Cursors kann der Programmierer festlegen, wieviele Datensätze der Block enthalten soll. Relativ und absolut kann nicht nur der Cursor, sondern auch innerhalb der einzelnen Blöcke positioniert werden.



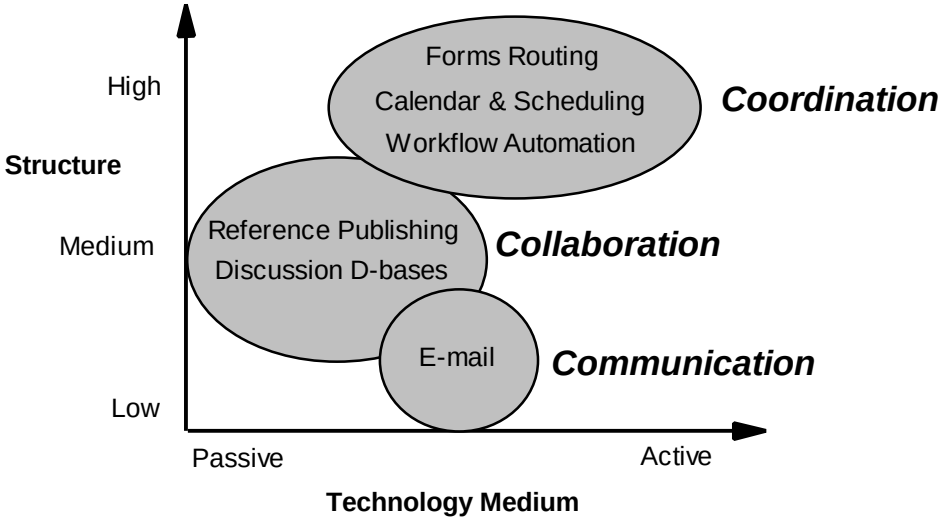
[i] C't magazin für computer technik - Datenlatein - Oktober 1994 - [teilweise zitiert]

6 Integration von Lotus Notes und Relationalen Datenbanken

In diesem Kapitel soll die Groupwarelösung Lotus Notes kurz vorgestellt werden, um danach die Unterschiede zu einer Relationalen Datenbank herauszuarbeiten. Darauf aufbauend sollen Werkzeuge und fertige Lösungen zur Integration beider Systeme betrachtet werden.

6.1 Die Groupwarelösung Lotus Notes

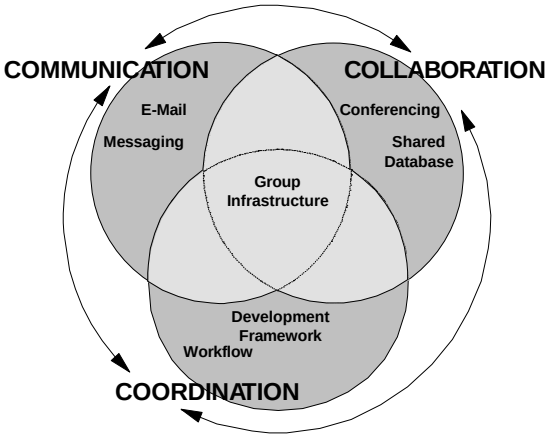
Bislang hat sich noch keine feste Definition von Groupware herauskristallisiert. Einvernehmen besteht jedoch darin, daß diese verhältnismäßig neue Softwaregeneration die Zusammenarbeit einer Arbeitsgruppe qualitativ verbessern soll. Ebenso unumstritten heißt der Markführer in dieser Produktgattung Notes von der Firma Lotus. Für Lotus sind die drei Standbeine von Workgroup Computing: Kummunikation, Zusammenarbeit und Koordination. Damit wird klar, daß Groupwaresysteme nicht auf isolierten Rechnern zu finden sind, sondern nur innerhalb eines vernetzten Rechnerverbundes ihre Berechtigung haben. Folglich gewinnen Systeme dieser Art erst heute in den 90er Jahren, wo genügend Rechnerleistung und sich eine angemessene Netzwerk-Infrastruktur etabliert hat, mehr und mehr an Bedeutung und finden ihren Weg in die Unternehmungen. Aus der Sicht von Lotus fundiert Groupware und somit natürlich auch ihr Produkt Notes auf einer Verteilten Datenbank sowie einer Sendetechnik. Auf diesem Fundament müssen Hilfsmittel angeboten werden, die mehr oder minder strukturierte Gruppenarbeit aktiv lenken oder passiv unterstützen.



Quelle: Lotus White Paper - Groupware

Abb. 6-1: Groupwarekategorien

Bei E-Mail können sich Gruppenmitglieder, basierend auf der Sendetechnik, Nachrichten zuschicken. Wollen aber eine größere Anzahl von Mitglieder miteinander Ideen und Informationen austauschen, so nutzt man Foren in der Verteile Datenbank, um virtuelle Versammlungen raum- und zeitlos durchzuführen. Die Sendetechnik beruht auf das sogenannte push model, da hier die Information den Adressaten direkt zugeschickt wird. Das Pendant ist das pull model, wobei jeder selber aktiv werden muß, um sich die Information aus der Verteilten Datenbank zu holen. Eine ausreichende Entwicklungsumgebung vorausgesetzt, kann mit Hilfe beider Modelle eine Arbeitsfluß-Automation (workflow automation) installiert werden, die aufgrund von spezifizierten Zuständen und Bedingungen den Arbeitfluß steuert.



Aber erst die Gesamtheit aller vorgestellten Spielarten von Informationsmanagement macht, im Sinne des Synergie-Effekts, Workgroup-Computing aus. Somit ist Workgroup mehr als die Summe ihrer Teilsysteme (E-Meil, Conferenzing usw). Daneben bietet Lotus Notes weitere für die Praxis unverzichtbare Systemelemente an. Auf Dokumente, als der Träger von diversen Datenobjekten, kann über ein ausgeklügeltes Ansichten-Konzept zugegriffen werden, wobei Versionierung und die Definition von Hierarchien möglich sind. Die Konsistenz der zugrundeliegenden Verteilten Datenbanken, als Container für die Dokumente, wird mittels frei initiierten Replikation-Prozessen erzielt. Dieser Datenabgleich kann auch, im Sinne von Mobile Office, für die Aktuallisierung der Daten reisender Mitarbeiter eingesetzt werden. Weiterhin steht eine vielschichtige Anwendungs-Entwicklung-Umgebung zur Verfügung, die dem einfachen Benutzer als auch den professionellen Entwickler dient. Ausgefeilte Sicherheitsmechanismen mit Authentizierung, Zugriffskontrolle und Verschlüsselung wehren Datenmißbrauch ab.

6.2 Abgrenzung von Notes zu operativen Datenbanken

Datenbanksysteme haben sich seit den 70er Jahren fest in den Unternehmungen etabliert. Vor dieser Zeit verwalteten die Programme ihre Daten mittels Dateien selber und legten ihre Datenstruktur eigenständig fest. Folglich konnte ein Programm nur seine eigenen Dateien lesen, was zur redundanten Datenhaltung und Inkonsistenz zwischen den Dateien führte. Operative Datenbanksysteme befreiten die Programme von der Datenspeicherung und gaben ihnen eine transparente Schnittstelle auf einen einheitlichen Datenspeicher. Dabei dient das jeweilige Datenbankmodell zur formalen Beschreibung aller in der Datenbank enthaltenen Daten und deren Beziehungen. Der heutige Stand kommerzieller operativer Datenbanksysteme verwendet das Relationale Datenbankmodell, das das hierarchische Datenmodell und das Netzwerkmodell abgelöst hat. Vergleicht man Relationale Datenbanken mit der Datenhaltungskomponente von

Lotus Notes, so entdeckt man diverse Unterschiede. Diese sind nicht nur in der Art der beteiligten Daten zu sehen, sondern auch in der Art der Datenspeicherung

6.2.1 Art der Zieldaten

Professionelle operative Applikationen für beispielsweise Personaldatenverwaltung, Lagerhaltung oder auch Finanzbuchhaltung können auf die Dienste eines zentralen oder auch verteilten Datenbanksystems nicht mehr verzichten. Dabei handelt es sich um sogenannte formatierte Massendaten, auch harte Daten genannt, die platzsparend, redundantfrei und schnellzugreifbar abgelegt werden. Harte Daten kann man dadurch charakterisieren, daß sie mittels ihrer Datentypfestlegung einem klar festgelegten Wertebereich unterliegen. Ein Beispiel hierfür ist die Gehaltssumme eines Mitarbeiters, die als ganze Zahl zwischen einem fest vorgegebenen Maximal- und Minimalwert, abgespeichert wird. Ein weiteres Beispiel ist der Name der Kostenstelle, der sich aus einer maximal 50-Zeichen umfassenden Zeichenkette zusammensetzen darf. Diese genaue Festlegung des Datentyps eines Datenwertes ermöglicht einerseits die schnelle Verarbeitung von Massendaten und die Definition von Integritätsbedingungen, andererseits bedeutet es für andere Aufgaben eine zu strikte Einschränkung.

Unter weichen Daten versteht man unter anderem strukturierte Texte, Rastergrafiken, Bitmaps, Ton, Video, Animationen, 3D-Grafiken der unterschiedlichsten Formate sowie auch OLE-Objekte (Object Linking and Embedding), die im Zeitalter von Multimedia immer verstärkter Verbreitung finden. Diese im Verhältnis oft großen Datenobjekte, die einen breiten Definitionsfreiraum beanspruchen, sind unter anderem Zielgebiete von objektorientierten Datenbanken. Obwohl auch moderne Relationale Datenbanken diese unstrukturierten Daten als BLOB (Binary Large Object) unterstützen, sind sie doch vornehmlich für die Speicherung von operativen Massendaten wie Stücklisten, Kundenstammsätze oder auch Fertigungsaufträge zuständig.

Lotus Notes bezieht eine Teil seiner Existenzberechtigung aus der Integration dieser unstrukturierten Datentypen in sein Dokumentmanagement. Dokumente, wie beispielsweise Protokolle, Expertisen, Reports oder Korrespondenz dürfen in sogenannten Rich Text Felder beliebige oben beschriebene weiche Daten integrieren. Damit wird ein Dokument zum Compound-Document oder Semi-Structured Document. Beispielsweise kann einem Dokument, das sich mit

der Mängelrüge eines Kunden beschäftigt, ein eingescanntes Bild der defekten Ware beiliegen. Oder ein Bericht über die Absatzprobleme in der Region X liegt eine Auflistung der letzen Absatzzahlen in Form eines OLE-Spreadsheet Objektes bei. Der Adressat kann durch Doppelklick auf die Tabelle die dazugehörige Tabellenkalkulation starten und eventuell einige Werte korrigieren. Diese Beispiele sollen zeigen, daß die Zieldaten von Groupwarelösungen wie Lotus Notes und operative Datenbanken verschieden sind. Notes beschäftigt sich mit weichen unstrukturierten Daten vornehmlich innerhalb des teamorientierten Officebereich und möchte das derzeit noch weitverbreitete Informationsmanagement auf Papierbasis ablösen. Operativ Datenbanken dienen als Massenspeicher für die klassische Datenverarbeitung. Die Datenbankkomponente von Notes ist somit keine echte Konkurrenz zu operativen Datenbanken. Beide Systeme laufen parallel nebeneinander und dienen ihrem speziellen Einsatzb ereich.

Trotzdem kann es Schnittpunkte zwischen beiden Systemen geben. Notes erlaubt selbstverständlich auch die Einbindung von harten strukturierten Daten, wie Text, Schlüsselwörter, Zeitangaben und Zahlen, da sie die komfortable Verwaltung der im Vordergrund stehenden weichen Daten erst ermöglichen und ein natürliches Dokument erst komplimentieren. Andererseits erlauben Relationale Datenbanken der jüngeren Generation die Speicherung von Binary Large Objects, hinter denen sich weiche Daten verstecken. So gibt es zahlreiche denkbare Situationen in denen logisch gleiche Daten auf beiden Systemen vorhanden sind. Beispielsweise erfaßt die Relationale Datenbank Adressen aller Kunden in den Kundenstammdaten. Ein Teil dieser Adressen sind redundant in Notes gespeichert, wo sie für Besuchsberichte der Vertriebsabteilung benötigt werden. In einem anderen Beispiel werden Kundenreklamationen vom Außendienstmitarbeiter erfaßt. Das Dokument erfaßt, neben den eingescannten Fotos der beschädigten Maschine, Angaben über Teileart, Teilenummer und Art der verwendeten Oberflächenglasur. Letztere Angaben finden sich auch ausschnittsweise im Teilestammsatz auf der operativen Datenbank wieder. Hier wären nahtlose Schnittstellen zwischen beiden Systemen wünschenswert.

6.2.2 Art der Datenspeicherung

Datenbanken sollen reale und logische Objekte und deren Beziehungen anhand ausgewählter Attribute speichern. Bei Relationalen Datenbanken erfolgt die logische Datenspeicherung mit Hilfe

des Relationenmodells. Eine Datenbank besteht hier aus einer Sammlung von Relationen, die man sich als Tabellen vorstellen kann, in der die Datenwerte abgelegt werden. Ein Objekt belegt eine Tabellenzeile, in der die Attributwerte in Spalten geordnet aufgelistet sind. Die Eindeutigkeit dieses Tupels wird über dessen Primärschlüssel erzielt. Ein Primärschlüssel setzt sich aus einem oder mehreren Attributen zusammen, die niemals die gleichen Werte haben dürfen und somit das Tupel eindeutig identifizieren. Alle Attribute, die als Primärschlüssel fungieren können, werden Schlüsselkandidaten genannt. Kann ein Objekt in logische Teilobjekte aufgespalten werden, so kann man dieses auch beim Design berücksichtigen. Der gesamte Datensatz des Objektes (alle Attribute) wird in einzelne Tupel aufgeteilt, die jeweils einer separaten Tabelle zugeordnet werden. Diese Aufteilung sollte aufgrund von Normalisierungsregeln erfolgen, die eine platzsparende, redundanzarme Datenhaltung garantieren. Soll nun aus den einzelnen Tupeln wieder der gesamte Datensatz des Objektes konstruiert werden, so nutzt man Fremdschlüssel. Ein Fremdschlüssel ist ein Attribut oder eine Attributkombination einer Tabelle, deren Werte identisch mit den Werten eines Schlüsselkandidaten einer anderen Tabelle sind und somit eine logische Verbindung zwischen beiden Tabellen schaffen. Die Referentielle Integrität garantiert, daß diese Beziehung durch unbedachtsame Operationen niemals verloren geht. Daneben gibt es die Semantische Integrität, die die Korrektheit der Daten bezüglich der abgebildeten Realität garantieren soll. Im Mehrbenutzerbetrieb bei konkurrierenden Zugriffen kennt man zusätzlich die Operationale Integrität, die für permanente Konsistenz in diesem Umfeld zuständig ist. Ein zentraler Datenkatalog, ein sogenanntes Data Dictionary, hält alle Verwaltungsinformationen, wie beispielsweise Informationen über die Tabellen, die Tabellenspalten, Zugriffsberechtigungen usw., in speziellen Systemtabellen, auf die wie Nutzdaten zugegriffen werden kann. Mit Hilfe dieser Informationen kann man das Datenbankschema einer konkreten Relationalen Datenbank ermitteln.

Hier ist auch der größte Unterschied zwischen der Datenbankkomponente von Lotus Notes und dem relationalen Datenbankmodell zu finden. Notes besitzt kein Datenbankkatalog, in dem man zentral abgelegte Strukturinformationen abfragen könnte. Notes setzt sich aus einer Sammlung von Dokumenten zusammen, die beliebige Datenobjekte einkapseln. Versucht man Assoziationen mit dem relationalen Modell herzustellen, so kann man Dokumente (notes) mit Datensätze gleichsetzen. Normalisierung findet in Notes nicht statt. Alle zusammengehörigen Informationen werden deshalb in einem Dokument vereint. Sieht man einmal von Techniken wie Hyperlinks oder

der Definition eines Dokuments als Anwort-Dokument ab, stehen Dokumente nicht in Beziehung zu anderen Dokumenten. Felder (items) entsprechen den Attributen aus der relationalen Welt. Jedes Dokument kann eine beliebige Anzahl von Feldern beinhalten, ohne sich an irgendwelche Schemavorgaben zu richten. Jedes Feld ist gekennzeichnet durch den Feldnamen, Feldtyp, Datenlänge, Datenwert und einem Flag. Notes unterstützt die Datentypen Text, Nummer, Zeit/Datum, Textliste, Nummerliste und Zeit/Datumliste für formatierte harte Daten. Für unstrukturierte weiche Daten steht der Datentyp Rich Text zur Verfügung, der mehrere Megabytes Daten umfassen kann. Die interne Datenstruktur für Felder wird item genannt. Ein Rich Text Feld kann sich aus einem oder auch mehreren items mit gleichem Namen zusammensetzen. Die Daten eines solchen Rich Text-items bestehen aus einer beliebigen Menge von CD records (Compound Document oder Composite Date). Es gibt 36 unterschiedliche Typen von CD Records, die jeweils für einen bestimmten Typ von Daten ausgerichtet sind.

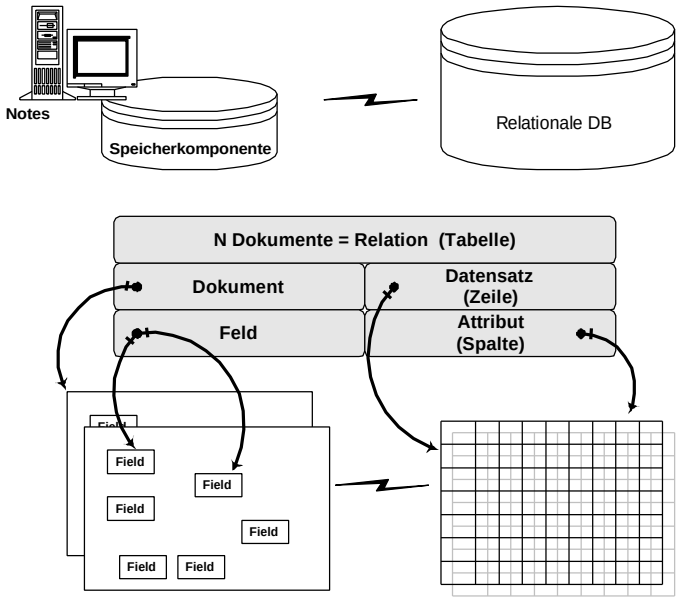


Abb. 6-2: Gegenüberstellung: Notes Speichermodell - relationales Modell

Notes weist jedem Dokument automatisch eine ID zu. Diese identifiziert, im Sinne eines Primärschlüssels, ein Dokument eindeutig. Diese ID setzt sich aus mehreren verwebten Teil-IDs zusammen (Universal Note ID, Originator ID, Global Note ID, Note ID, Instance ID, Global Instance ID), die jeweils für unterschiedliche Aufgaben benötigt werden. So identifiziert die Universal Note ID alle Kopien eines Dokuments unabhängig davon, wann sie das letzte Mal modifiziert wurde oder auf welchem Rechnerknoten im Netz sie sich befindet. Folglich müssen

keine Feldwerte zum Primärschlüssel gekürt werden, so wie es im übertragenden Sinne das relationale Modell verlangen würde. Da Dokumente, anders als Tupel, keiner Relation zugeordnet sind, fehlt eine Zuordnung der einzelnen Dokumente und deren Feldwerte zu einer einordnenden Objektklasse. Abhilfe schaffen hier Masken, mit denen Dokumente erzeugt werden und denen sie auch danach zur Bildschirmanzeige der Feldwerte zugeordnet bleiben. Technisch gesehen hat jedes Dokument ein Feld mit den Namen „Form“, in dem der Name der zugeordneten Maske vermerkt wird. Aktiviert man das Dokument in einer Ansicht, so werden die Datenwerte des Dokuments in der entsprechenden Maske angezeigt. Beispielsweise wird ein Dokument mit Adreßinformationen, das im Feld Form den Wert „Privatadresse“ gespeichert hat, mit der Maske „Privatadresse“ aufgerufen. Ein zweites Dokument, das im Feld Form „Geschäftsadresse“ vermerkt hat, wird mit der Maske „Geschäftsadresse“ aufgerufen. Damit ist jedem Dokument eine bestimmte Maske zugeordnet, die dadurch wiederum implizit alle ihr zugeordneten Dokumente einer bestimmten Klasse zuordnet. So werden alle Dokumente im vorausgegangenen Beispiel über ihre Maske in Geschätsadressen und Privatadressen aufgeteilt.

Macht man sich die Tatsache zu Nutze, daß in der Maske anzuzeigende beziehungsweise zu erzeugende Felder eines Dokuments mit Name und Datentyp festgelegt sind, könnte man diese Metainformation nutzen, um daraus das Datenbankschema der zugehörigen Notes Datenbank herzuleiten. So wie eine Relation alle Attribute und deren Datentypen festlegt, so würde eine Maske alle Felder und deren Datentypen des zugeordneten Dokuments beinhalten. Die Metainformation aus allen Masken einer Notes Datenbank würde dessen Datenbankschema ergeben. Das Konjunktiv macht deutlich, daß hier einigen Einschränkungen zu machen sind. Erstens können Masken Felder besitzen, die nur für die Anzeige berechnet werden. Das heißt, der Feldwert wird bei der Anzeige eines Dokuments mit der entsprechenden Maske mittels @Funktionen berechnet. Nach Beenden der Anzeige wird dieser berechnete Feldwert wieder verworfen, das heißt er wird nicht im Dokument gespeichert. Folglich dürfen diese Felder mit dem Feldtyp „Berechnet zur Anzeige“ für die Herleitung des Schemas nicht berücksichtigt werden. Weiterhin können mit @Funktionen und LotusScript Prozeduren Dokumente manipuliert werden, was mit Hilfe der Metainformation aus den Masken nicht zu ersehen ist. Beispielsweise kann so ein neues Feld in ein Dokument gesetzt werden oder ein vorhandenes entfernt werden. So kann man sich nicht voll darauf verlassen, daß auch alle Felder der Maske im damit verbundenen

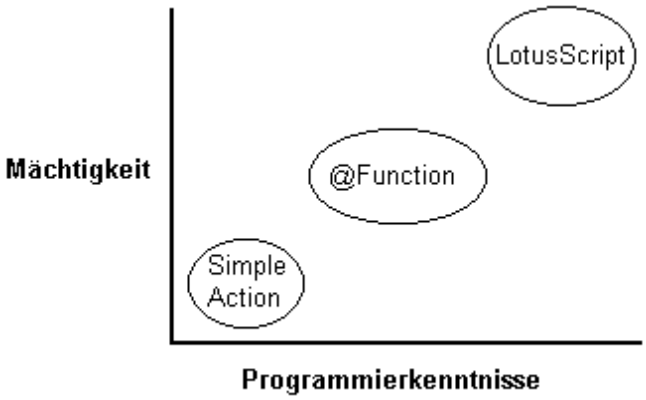
Dokument vorhanden sind und daß nicht noch weitere mit der Maske nicht direkt sichtbare Felder dem Dokument innewohnen. Drittens bereitet die Existenz von Teilmasken Schwierigkeiten. Teilmasken beinhalten wie normale Masken Felddefinitionen plus statischen Text. Masken können Teilmasken importieren, so daß Teile, die in vielen Masken zu finden sind, nur einmal innerhalb einer Teilmaske definiert werden müssen. Importieren heißt hier genauer, daß eine Teilmaske entweder in eine Maske eingesetzt wird oder über eine Formel berechnet werden kann. Beim Einsetzen übernimmt die Maske alle Metainformationen der Teilmaske, was für die Herleitung des Schemas zu keinen Problemen führt. Werden die Teilmasken aber aufgrund einer Formel in die Maske integriert, so können hier aktuelle Aspekte zur Zeit des Maskenaufrufs berücksichtigt werden. Hierdurch erhalten Masken eine Dynamik, die mit einem statischen Schema nicht zu vereinbaren ist. Als vierten und letzten Punkt sind die Dialogmasken zu erwähnen. @Funktionen und LotusScript Prozeduren in Aktionen oder Agenten können mit dem Benutzer in Dialog treten. Dazu werden unter anderem normale Masken verwendet, die beispielsweise als Dialogboxen fungieren. Diese sogenannten Dialogmasken unterscheiden sich vom Aufbau her von normalen Masken nicht. Sie sind für die Herleitung des Schemas jedoch nicht relevant und können somit für diese Zwecke unberücksichtigt bleiben.

6.3 Technische Realisierung

Es gibt die unterschiedlichsten Ansatzpunkte, um eine wie auch immer geartete Verbindung zwischen Lotus Notes und einem Relationalen Datenbanksystem herzustellen. So ist zu entscheiden, ob eine vorhandene auf dem Markt befindliche Lösungen genutzt wird, oder, falls kein zufriedenstellendes Produkt bereitsteht, man selber programmiert. Die Eigenproduktion erfordert neben der konzeptionellen Lösung die Wahl des geeigneten Programmierwerkzeugs. So könnte man im konkreten Fall einer Notes-RDB Kopplung, entweder mit Hilfe der in Notes vorhandenen Programmiermöglichkeiten auf eine Relationale Datenbank zugreifen, oder man setzt zwischen Notes und der Datenbank eine weitere Ebenen, die, im Sinne von Middleware, eine Brücke zwischen beiden Systemen bildet. Nach der Betrachtung aller Programmiermöglichkeiten im Notes Umfeld soll die Integrationslösung NotesPump, die auch von Lotus vertrieben wird, vorgestellt werden. Auf weitere Produkte wie beispielsweise Casahl Technology Replic-Action,

Percussion Software Notrix Composer oder Platinum Technology InfoPump soll innerhalb dieser Arbeit nicht weiter eingegangen werden.

6.3.1 Anwendungsentwicklung mit Lotus Notes



Notes bietet eine breite Spanne von Werkzeugen für die Anwendungsentwicklung an. Für den einfachen Benutzer ohne jegliche Programmierkenntnisse stehen sogenannte „Simple Actions“ zur Verfügung, mit denen durch einfache Menüauswahl eine Reihenfolge von auszuführenden Aktionen definiert werden kann. @Function verlangen vom Benutzer eine längere Einarbeitungszeit, ermöglich gleichzeitig aber auch das Lösen von komplexeren Aufgaben, was den Zugriff auf ODBC-fähigen Datenbanken mit einschließt. Die mit Parametern versehenen Funktionsaufrufe werden vornehmlich für die Programmierung von Eigenschaften und Methoden der Notes-Objekte eingesetzt, wie beispielsweise bei der Bestimmung von Defaultwerten oder Schlüsselwerten bei Feldern. LotusScript hingegen ist ein voll objektorientierte Programmiersprache, die wegen ihrer uneingeschränkten Steuerungslogik die umfangreichsten Möglichkeiten bei der Programmierung von komplexen Aufgaben in Agenten, Actions oder Buttons bietet. Lotus Notes verlangt entweder die Programmierung mit @Functions oder LotusScript, oder läßt auch die alternative Wahl zwischen beiden Werkzeugen zu. Neben der Anwendungsentwicklung innerhalb der Notes-Umgebung kann auch von außerhalb auf Notes zugegriffen werde. Zugriffsschnittstellen bieten einmal die diversen Notes-APIs sowie der Notes ODBC-Treiber. Alle Werkzeuge sollen nun nacheinander, im Hinblick auf eine Kopplung von Notes mit Relationalen Datenbanken, betrachtet werden.

6.3.1.1 @Function

Mit den Funktionen @DBColumn und @DBLookup kann neben Notes Datenbanken auch auf anderen ODBC-fähige Datenbanken zugegriffen werden, um beliebige Daten auszulesen[i]. @DBColumn greift über den ODBC-Treibers auf eine spezifizierte Relation einer SQL-Datenbank zu und gibt alle Werte einer Spalte zurück. Dabei kann die Sortierreihenfolge und die Behandlung von Null-Werten vorbestimmt werden. Auch kann man doppelte Werte ignorieren lassen. Ein Hinzufügen, Löschen oder Modifizieren der externen Daten ist nicht möglich. @DBLookup erweitert @DBColumn, indem zusätzlich eine Selektion über Spaltenwerte erfolgen kann. Mit @DBCommand, das nur mit externen Datenbanken zusammenarbeitet, können beliebige SQL-Befehle oder Stored Procedure aufgerufen werden.

6.3.1.2 Lotus Script

LotusScript ist eine vollwertige objektorientierte Programmierspache, mit der man strukturierten Code erzeugen kann[ii]. Die Visual Basic ähnliche Sprache ist in Lotus Notes eingebunden, das heißt die Programmierung erfolgt mit Hilfe der integrierten Entwicklungsumgebung (IDE) als komfortables Werkzeug zum Editieren und Debugen. Der erzeugte Code läuft innerhalb des Notes Clients oder Servers ab, was ihm wiederum unabhänigig vom Betriebssystem macht. Bestimmte Ereignisse (event) rufen den in Notes-Objekten liegenden Code auf. Eine Lotus-Script Klasse definiert Eigenschaften und aufzurufende Methoden eines Lotus-Script Objektes. So hat beispielsweise das Datenbank-Objekt die Eigenschaft Created, die angibt, wann die Datenbank erstellt wurde und die Methode CreateCopy, mit der eine Kopie der Datenbank erzeugt werden kann.

[i] Siehe [DI96] ab Seite 11 - Native Notes Access to DBMSs: @DB Functions
Siehe [LN96]
[ii] Siehe [LI96] ab Seite 131 - 7 Using LotusScript
Siehe [LN96]

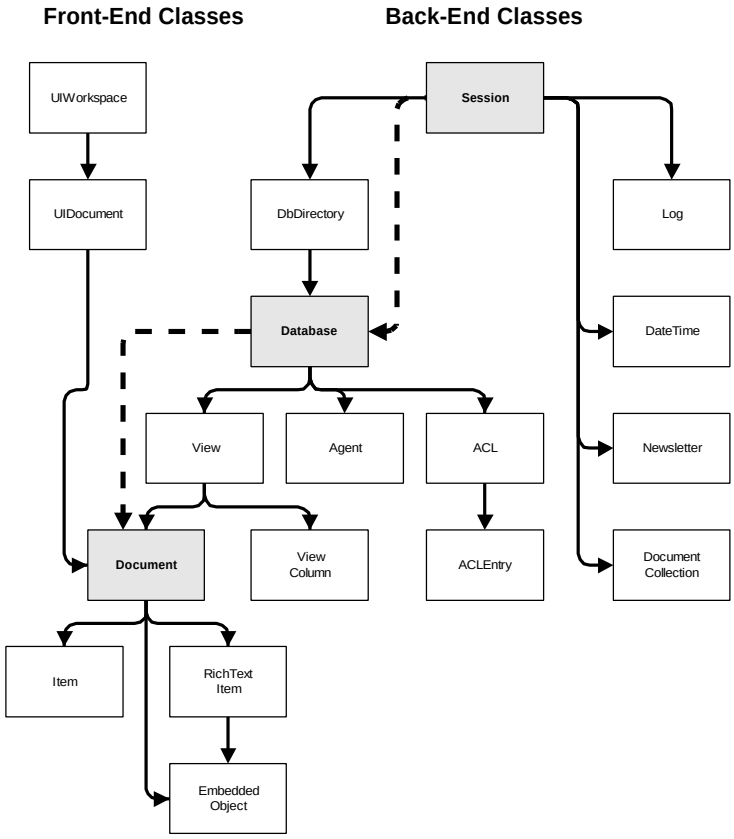


Abb. 6-3: LotusScript Klassenhierarchie

Die Klassenhierarchie legt fest, welche Klasse welche Unterklassen umfaßt, beziehungsweise in welchen Oberklassen diese Klasse eingebettet ist. So ist das oben beschriebene Datenbank-Objekt in eine Notes-Session eingebunden und beinhaltet selbst Objekte der Unterklasse Notes-Dokument. Über diese Hierarchie kann auf einzelne Objekte zugegriffen werden, um deren Eigenschaften zu lesen, zu setzen oder deren Methoden aufzurufen. Man kann zwischen Hintergrund und Benutzerschnittstellen-Klassen unterscheiden, wobei letztere die Aktionen eines Benutzers mimt. Neben diesen vorgegebenen Klassen kann der Programmierer auch eigene Klassen definieren. Weiterhin umfaßt die Syntax neben den statischen Datentypen (Ganzzahl, Festkommazahl, Fließkommazahl, Zeichenkette, Variant, Array, Liste, Objektreferenz) auch zusammengesetzte Datentypen. Ein spezieller Datentyp Variant kann unterschiedliche Datentypen umfassen und kann somit als Äquivalent des in Sprachen der 3. Generation vorhandenen Pointers betrachtet werden.

Die vorausgegangene Beschreibung zeigt, daß LotusScript keine Schnittstelle zu externen Datenbank besitzt. LotusScript erlaubt aber zum einen die Aufrufe von C-Funktionen und zum anderen die Erweiterung der LotusScript Sprache mit sogenannten LotusScriptEXtension Modulen. Mit beiden Möglichkeiten kann man den LotusScript Sprachschatz um die gewünschte Schnittstelle zu einer Relationalen Datenbank erweitern.

6.3.1.2.1 Aufruf von C-Funktionen

Die C-Funktionen können im LotusScript-Code wie ganz normale Script-Funktionen aufgerufen werden[i]. Allerdings müssen sie gesondert deklariert werden und bei der Parameterübergabe müssen bestimmte Regeln eingehalten werden. Bei allen Windwos (16Bit/95/NT) und beim OS/2 Betriebssystem werden die C-Funktionen in Dynamic Link Libraries (DLLs) verwahrt. Auf anderen Plattformen werden äquivalente Shared Libraries verwendet. DLLs wie auch Shared Libraries sind Binärcode-Dateien, deren Funktionen von beliebigen Programmen aufgerufen werden dürfen. Da der Binärcode nur speziell für ein bestimmtes Betriebssystem kompiliert werden kann, verliert der aufrufende LotusScript-Code seine Systemunabhängigkeit. Möglich ist nun, daß über die C-Funktionen auf die ODBC-API zugegriffen wird und man somit eine Schnittstelle zu einer beliebigen ODBC-fähigen Datenbank schafft. (*Siehe 5 Client-Server Architekturen*)

6.3.1.2.2 LotusScriptEXtension (LSX)

Mit dem LSX-Toolkit kann man den LotusScript-Sprachumfang um weitere Klassen erweitern[i]. Diese zusätzlichen Klassen kann der LotusScript-Programmierer, wie die bereits vorhandenen Klassen, innerhalb seines Codes verwenden. Technisch gesehen bestehen die LSX-Klassen aus C++-Code, der über die bereits erwähnten DLLs oder Shared Libraries zur Verfügung gestellt wird. Der Unterschied zu einfachen C-Funktionsaufrufen ist, daß sich die LotusScriptExtension nahtloser in den Script-Code integriert. Die zusätzlichen Klassen kann der Script-Programmierer wie die gewohnten verwenden, bis auf die einmalige Bekanntmachung der entsprechenden Bibliothek durch den Befehl „USELSX“. Mit dieser Technik könnte man beispielsweise komplizierte Algorithmen in LSX-Klassen auslagern. Gründe hierfür sind einerseits die

[i] Siehe [LN96]

Erweiterung des auf 64Kbyte begrenzten Speicherplatzes für Code und Daten in LotusScript Modulen, andererseits kann so der Quellcode gegenüber dritten geschützt werden.

LotusScript Data Object (LS:DO)

Lotus selber hat die ODBC-API in LSX-Klassen umhüllt, so daß ODBC-Aufrufe auch über LotusScript möglich sind[ii]. Dieses LSX-Klassen werden LotusScript Data Object (LS:DO) genannt und liegen jeder Lotus Notes Installation für Windows 3.X, Windows 95, Windows NT und OS/2 bei. Ab der Version 4.5 sollen auch die Plattformen Solaris SPARC, Solaris x86, HP-UX und AIX mit einer entsprechenden LSX-Bibliothek unterstützt werden. Die LS:DO besteht aus drei Klassen:

?? ODBCConnection

?? ODBCQuery

?? ODBCResultSet

Ein LotusScript Modul muß den Befehl UseLSX „*LSXODBC“ aufrufen, damit die drei Klassen zur Verfügung stehen. Hiermit wird eine Verbindung zur DLL „nlsxodbc.dll“[iii] hergestellt, in der der LSX-Binärcode liegt. Die Klasse ODBCConnection bietet unter anderem die Methode „ConnectTo“ an, mit der eine Verbindung zu einer ODBC-fähigen Datenbank hergestellt werden kann. Der SQL-Befehl für einen Aufruf wird über die Klasse ODBC-Query formuliert, um ihn mit der Methode „Execute“ der Klasse ODBCResultSet an den ODBC-Treiber zu schicken. Vorher kann das Verhalten der Abfrage mit dem Setzen bestimmter Eigenschaften beeinflußt werden. Nach erfolgreicher Ausführung erlauben andere Methoden nach dem Curserprinzip über die Ergebnismenge zu navigieren und Operationen darauf auszuführen. Danach kann die modifizierte Ergebnismenge in die Datenbank zurückgeschrieben werden. Auszulesende Daten werden automatisch in LotusScript-Datentypen transformiert.

[i] Siehe [LS96]
[ii] Siehe [DI96] ab Seite 6
Siehe [LI96] ab Seite 213 - LS:DO Data Access Object Model
[iii] Prefix n bei Windows 95

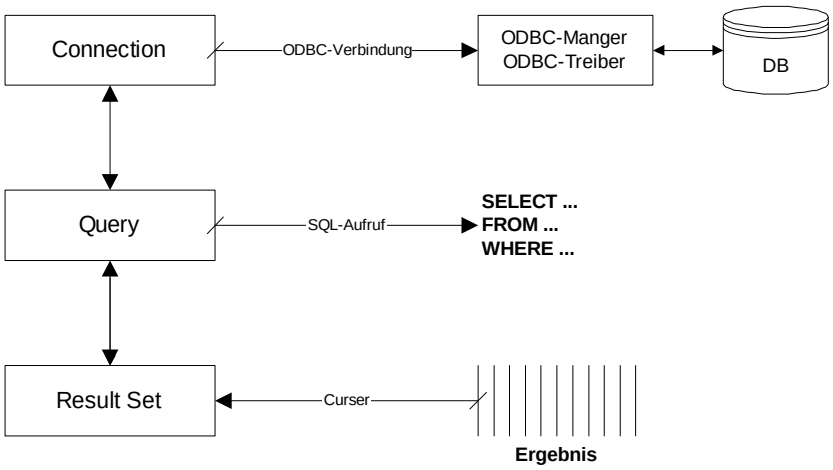


Abb. 6-4: Zugriff auf eine Datenbank mit LotusScript Data Object

Leider sind derzeit noch erhebliche Fehler in den LS:DO zu finden, beziehungsweise stehen noch nicht alle versprochenen Eigenschaften und Methoden zur Verfügung. So ist beispielsweise keine Autoregistrierung der ODBC-Quellen mit Hilfe einer entsprechenden Methode möglich. Für fest installierte Datenbanken ist dieses kein großes Problem, da das Anlegen der ODBC-Quelle mit Hilfe des ODBC-Managers ein einmaliger Akt ist. Bei ODBC-Verbindungen der ersten Rangstufe (Single-Tier Treiber) besteht jedoch das Problem, daß die entsprechenden Dateien (Beispiel Dbase oder Excel) nur temporär vorhanden sind oder häufiger einen neuen Pfad zugewiesen bekommen. Ebenso fehlt eine Unterstützung von Rich Text Daten.

MQ Series Link

MQSeries (Messaging Queue Interface) ist IBM's propritäre Schnittstelle zu transaktionsbasierten Datenbanken, die unterschiedliche Netzwerke und unterschiedliche Betriebssysteme verschleiert[i]. Die MQSeries API wird ähnlich der ODBC-API in LSX-Klassen eingebettet und kann somit über LotusScript aufgerufen werden. Damit kann der Benutzer in der Notes-Umgebung Prozesse, die von der IBM-Middleware unterstützt werden, steuern bzw. verändern. Anders als bei ODBC wird hier jedoch nicht direkt auf die Daten zugegriffen, sondern man führte sogenannte Transaktionen aus. Bei diesen Transaktionen wird neben der sowieso unabdingbaren Forderung nach Konsistenz

[i] Siehe [DI96] ab Seite 17

und Dauerhaftigkeit auch strikte Isolation sowie Atomizität mit Rücksetzen auf den Anfangszustand im Störungsfall verlangt[i].

Oracle LSX

Ähnlich den LotusScript Data Object (LS:DO) funktioniert auch die Oracle LSX vom Datenbankhersteller Oracle. Hier wird jedoch nicht auf die offene herstellerunabhängige ODBC-API zurückgegriffen, sondern man nutzt die hauseigene Schnittstelle.

6.3.1.3 Notes API

Professionelle Notes-Entwickler stoßen bei komplexen Aufgaben mit den Notes Entwicklungswerkzeugen oft an deren Grenze des Machbaren[ii]. Für solche Fälle bietet sich die Programmierung mittels der Notes-API an. Das API ist eine in die Programmiersprache C eingebettete Programmierschnittstelle, die Zugriff auf die internen Daten- und Prozeßstrukturen von Notes bietet. Zur Zeit gibt es drei Arten von APIs, die C API, die HiTest C API und die C++ API. Die beiden letzten bauen jedoch auf die C API auf, indem sie ein höhere Abstraktionsstufe anbieten und beide objektorientierte Programmierung ermöglichen, wobei die HiTest C API nicht vollständig objektorientiert ist.

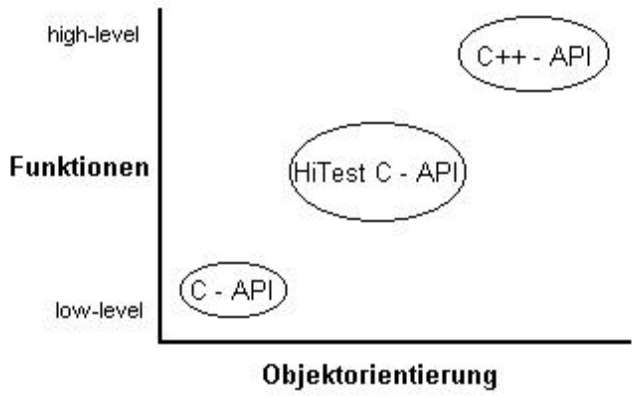


Abb. 6-5: Notes API's

[i] auch ACID-Eigenschaft genannt: atomicity, consistency, isolation, durability [LL95] Seite 623
[ii] Siehe [LA95]

Das API-Toolkit besteht aus einer Reihe von Headerfiles, die charakteristische interne Strukturen beschreiben, und Funktionen, durch die der Zugriff auf die Notes-Daten realisiert wird. Durch Einbinden in eine Programmiersprache, kann diese folgende Operationen auf den Notes-Kern ausführen:

?? Erstellen, Löschen und Kopieren von Notes Datenbanken

?? Lesen, Schreiben und Modifizieren von Notes-Dokumenten

?? Suchen, Sortieren und Indizieren von Dokumenten einer Notes Datenbank

?? E-Mail und Dokumente versenden

?? Lesen, Schreiben und Modifizieren von Designelementen

?? Lesen, Schreiben und Überprüfen von Formeln

?? System verwalten

Jedoch besteht keine Möglichkeit, über die API die Benutzeroberfläche zu steuern oder auf den Notes-Arbeitsbereich zuzugreifen. Damit ist es möglich, folgende API-Applikationen zu realisieren:

?? Stand-alone-Programme
Stand-alone-Programme laufen außerhalb der Notes-Umgebung und greifen von außen auf Notes zu.

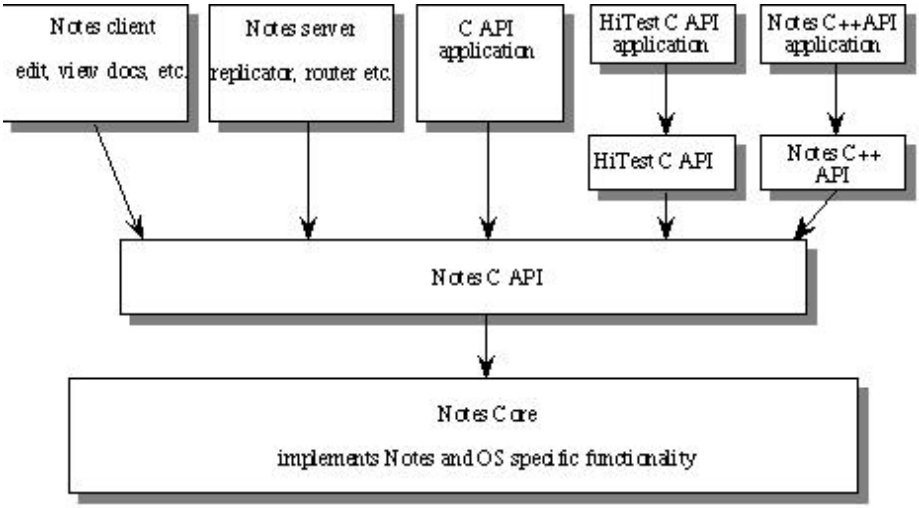
?? Server-Tasks
Diese Programme laufen als Teilprozesse des Servers, um zeit- oder ereignisgesteuert gewisse Verarbeitungsfunktionen auszuführen.

?? Import- und Exportfilter
Sie dienen dazu nicht unterstützte Dateiformate zu importieren oder exportieren.

?? Menüfunktionen
Erweitern den Workstation-Desktop um zusätzliche Menüpunkte.

?? Datenbank-Hook-Treiber
Werden bei bestimmten Ereignissen angestoßen und ausgeführt.

?? Treiber für externe Datenbanken



Quelle: Lotus Notes APIs - 12/95 Speech

Abb. 6-6: Architektur einer Notesumgebung

Da die C API nicht nur die Grundlage der HiTest API und C++ API ist, sondern auch vom Notes Client und Server genutzt wird, findet man alle bestehenden Notes-Sicherheitsmaßnahem auch in API-Applikationen wieder. Folglich ist eine Umgehung der Sicherheitsvorkehrungen auf Server-, Ansicht-, Dokument- und Feldebene über die API-Programmierung ausgeschlossen.

Möchte man Daten zwischen externen Datenbanken und Lotus Notes austauschen, so kann der Zugriff auf die Notes-Daten über eine der drei Notes-APIs erfolgen. Der Zugriff auf externe Daten kann mittels Embedded SQL, Dynamic SQL oder einer Call-Schnittstelle wie ODBC vorgenommen werden. Der Entwickler hat zu entscheiden, welche API und welche Schnittstelle zur externen Datenbank er nutzen will. Weiterhin ist zu bedenken, daß der API-Binärcode jeweils nur von einem bestimmten Betriebssystem lesbar ist und somit ein Kompilierung für ein weiteres Betriebssystem das Nutzen von systemspezifischen Funktionen ausschließt.

6.3.1.4 Lotus Notes ODBC-Treiber für Windows

NotesSQL ist ein ODBC-Treiber für Windows, der es ODBC-fähigen Werkzeugen erlaubt auf Notes-Daten zuzugreifen[i]. Der Notes Datenbestand präsentiert sich über die ODBC-Schnittstelle im relationalen Modell und kann somit per SQL-Befehlen bearbeitet werden. Dabei werden Masken und Ansichten auf Tabellen abgebildet und Felder auf Attribute. Damit auch auf

[i] Siehe [SQ96]

sogenannte versteckte Felder zugegriffen werden kann, gibt es eine „Universal Relation“, die sämtliche Dokumente der Datenbank umfaßt.

6.3.2 Lotus NotesPump

Lotus NotesPump ist eine Server-basiertes Werkzeug für den Datenaustausch zwischen Lotus Notes und Relationalen Datenbanksystemen[i]. Zur Zeit werden die Datenbankprodukte von Sybase, Oracle und IBM unterstützt. Daneben kann über ODBC jede andere ODBC-fähige Datenbank integriert werden. Neben dem einfachen Datentransfer können über NotesPump auch zwei Datenbanken synchronisiert beziehungsweise repliziert werden. Die gesamte Verwaltung und Steuerung erfolgt über entsprechende Notes-Applikationen.

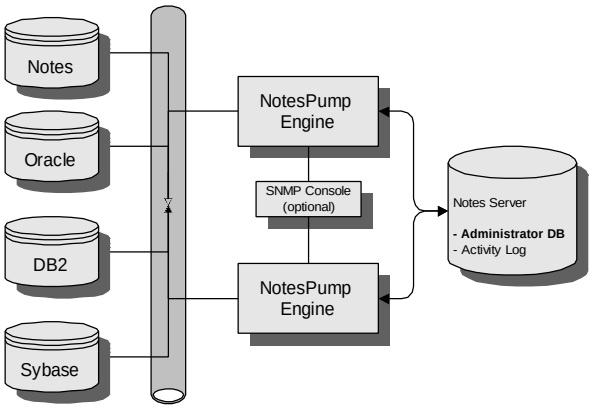


Abb. 6-7: Die NotesPump Architektur

Eine NotesPump Infrastruktur setzt sich aus folgenden Komponenten zusammen:

?? **NotesPump Administrator**
Die auch als Control Strore bezeichnete Notesdatenbank dient zur Spezifikation der beteiligten Datenbanken und Konfiguration der Aktivitäten, die den Datentransfer oder die Replikation anstoßen.

?? **NotesPump Engine**
Den eigentlichen Datentransfer übernimmt die NotesPump Engine, die, separat als multi-tasking und multi-threading Maschine, entweder auf einer OS/2 oder Windows NT Umgebung läuft. Zur Erhöhung der Performance können gleichzeitig mehrere Engines eingesetzt werden. In

[i] Siehe [DI96] Seite 14 - High-Volume Data Transfer Tools
Siehe [LI96] Seite 232 - Lotus Notes Pump 1.0
Siehe [NP96]
Siehe [PI96]

spezifizierten Zeitabständen fragen sie die NotesPump Administratordatenbank nach auszuführenden Aktivitäten ab. Alle durchgeführten Transferoperationen werden in eine Notes-Protokoll-Datenbank aufgezeichnet.

?? **Datenbank**
Neben Notesdatenbanken wird zur Zeit Sybase System 10, Oracle 7, IBM DB2 sowie ODBC-fähige Datenbanken unterstützt. Weitere Datenbanksysteme können mit Hilfe der erhältlichen NotesPump API hinzugefügt werden. Dabei ist nicht nur ein Datenaustausch zwischen Notes und den Relationalen Datenbanken möglich, sondern die NotesPump Engine kann auch zwischen zwei Relationalen Datenbanken eingesetzt werden.

Aktivitäten, die in der NotesPump Administrator Datenbank angelegt und gespeichert werden, sind gleichzusetzen mit Aufträgen für die NotesPump Engine. So beschreiben die Aktivitäts-Dokumente die beteiligten Datenbanken, sich eventuell anschließende Aktivitäten sowie den Ausführungszeitpunkt, der wiederum auch abhänigig von bestimmten Bedingungen sein kann. Weiterhin muß eine der gewünschten Ausführungsarten gewählt werden, die nachfolgend aufgelistet sind:

?? **Datentransfer**
Per Datentransfer können ganze Datenbanken oder bestimmte Teilmengen kopiert werden. Der betroffene Datenbereich wird bei Relationalen Datenbanken mit Hilfe von Standart-SQL bestimmt. Bei Notes Datenbanken verwendet man zur Selektion die bekannten @Function.

?? **Synchronisation von Daten**
Bei der Synchronisation werden Datensätze anhand bestimmter Schlüsselwerte abgeglichen. Nach dem Master-Slave Prinzip, bestimmt die als Master gekürte Datenbank welche Datensätze bei Ungleichheit die Slave-Datenbank übernimmt. Zusätzlich übernimmt die Slave-Datenbank von der Master-Datenbank Datensätze, die ihre Datenbasis nicht enthält, beziehungsweise löscht sie Datensätze, die nur innerhalb ihrer Datenbasis zu finden sind. Zeitstempel verfeinern das beschriebene Verfahren. Hier wird der Datensatz der Datenbank gewählt, dessen letzte Änderung zeitlich am aktuellsten ist.

Abschließend soll aus den zahlreichen Eigenschaften von NotesPump, noch dessen Zusammenarbeit mit der SNMP-Technik erwähnt werden.

7 Entwicklung eines Prototyps

Mit der nun folgenden Betrachtung des Prototyps sollen die theoretischen Überlegungen in die Praxis umgesetzt werden. Dabei soll keine 1:1 Umsetzung der vorgestellten Modelle in eine konkrete Softwarelösung erfolgen, vielmehr diente die Theorie als Ideenlieferant und Leitfaden bei der Entwicklung eines Prototyps. Aufgabe war es eine Brücke zwischen der Lotus Notes Welt und der Relationalen Datenbankwelt zu schaffen. Unterschiede und Eigenarten beider Welten, sowie deren parallele Daseinsberechtigung innerhalb einer Organisation wurden in vorausgegangenen Kapiteln beschrieben (*Siehe 6.2* ***Abgrenzung von Notes zu operativen Datenbanken***). Konkret sollten Werkzeuge für den Lotus Notes Nutzer entwickelt werden, die ihm Zugriff auf Relationale Datenbanken gestatten. Nach der genauen Spezifizierung der Aufgabenstellung soll anhand eines ausführlichen Beispiels die Möglichkeiten und die Arbeitsweise des Prototyps ausführlich vorgestellt werden. Jedes Unterkapitel schließt mit den Punkt „Programmiertechnische Realisierung“ ab. Hier wird auf die Konzeption und Programmierung des zuvor beschriebenen Teils der Architektur eingegangen. Mit Hilfe dieser Hintergrundinformationen dürfte es keine Schwierigkeiten bereiten den dazugehörigen gut kommentierten Code zu lesen.

7.1 Konkrete Aufgabenstellung

Man kann zwei grobe Teilbereiche bei der Integration von Relationalen Datenbanken in Lotus Notes ausmachen. Ähnlich wie Notes Pump (*Siehe 6.3.2* ***Lotus NotesPump***) unterscheide ich zwischen reinem Datentransfer und der Synchronisation von Daten.

7.1.1 Datentransfer für Archivierung

Beim Datentransfer werden Daten von einem System auf das andere System übertragen. Mit dieser Technik kann unter anderen eine Archivierung von Notes Daten in Relationale Datenbanken realisiert werden. In der Praxis zeigt sich, daß das Datenvolumen von einigen Notes Datenbanken erheblich schnell anwachsen kann, und sich parallel dazu die Schwerfälligkeit des Systems störend erhöht. Mit jedem zusätzlichen Kilobyte an Daten verlängert sich unter anderem der Indexaufbau einer Ansicht sowie die Laufzeit von auf der Datenbasis arbeitenden Agenten. Folglich ist es ratsam, die Datenbank dadurch zu entlasten, daß man ganze Dokumente, beziehungsweise

bestimmte Teildaten eines Dokuments, auslagert. Speicherfressende OLE-Objekt und Attachement sind hierfür Aspiranten. Ein Teil dieser ausgelagerten Daten soll auch weiterhin Online zur Verfügung stehen. Zeitliche Verzögerung beim Zugriff können in bestimmten Rahmen in Kauf genommen werden. Vom übrigen Teil der Daten verlangt man, daß diese sicher verwahrt werden. Aber auch hier soll sichergestellt sein, daß die betroffenen Dokumente in einem angemessenen Zeitraum wieder rekonstruiert werden können. Für beide Spielarten bietet sich der Einsatz einer Relationalen Datenbank an. Sie kann als Massendatenspeicher mit Hilfe der erwähnten BLOB-Technik (*Siehe 6.2.2* ***Art der Datenspeicherung***) Dokumente und Teildokumente temporär oder permanent aufbewahren. [i]Alternativ dazu ist aber auch die Datenablage in speziellen Notes Archivierungs Datenbanken denkbar. Hier kann die Archivierung manuell oder mittels spezieller Agenten durch einfaches kopieren von der original Datenbank in die Back-Up Datenbank erfolgen. Bei der Archivierung innerhalb der Notes-Umgebung treten keine Systembrüche auf, die Quelle von zahlreichen Problemen sein können. So kann beispielsweise eine archivierte Notesdatenbank auch nach Jahren noch gelesen werden, falls alle notwendigen Designelemente mitarchiviert wurden und zukünftige Notes-Systeme die kompatible Ausführung von alten Datenbanken erlauben. Will man dieses auch bei der Archivierung in Relationalen Datenbanken garantieren, so müßte entweder das Maskendesign mit jedem Dokument gespeichert werden oder man archiviert das Design separat. Für die temporäre Auslagerung kann das Design unberücksichtigt bleiben.

7.1.2 Synchronisation

So sehe ich die Hauptaufgabe bei der Kopplung von Notes und Relationalen Datenbanken in der Synchronisation von Daten beider Systeme. Synchronisation bedeutet in diesem Zusammenhang, daß logisch gleiche Daten sowohl in Notes als auch auf einer Relationalen Datenbank vorhanden sind. Diese sollen abgeglichen und abgestimmt werden. Beispielsweise kö nnen Kundenadressen innerhalb einer Notes-Office Lösung als auch in einer Relationalen Datenbank als Basis für operative Anwendungen vorhanden sein. Dabei muß nicht zwangsläufig auf beiden Systemen der identische Kundenstamm gespeichert sein. Das Dokument erfaßt zum Teil auch andere Einzeldaten als der Datensatz aus der Relationalen Datenbank. So verzichtet die Office-Lösung auf die Speicherung der Kundennummer für die Rechnungslegung. Dafür wird ein Foto des

[i] Siehe [NM95] Seite 92 bis 96 - About archiving an active database

Kunden und eventuelle Kommentare für den Außendienstmitarbeiter gespeichert. Zusätzlich ist man mit den bereits erläuterten Problemen der Schema- und Datenheterogenität konfrontiert (*Siehe 4.3* ***Schema- und Datenheterogenität***). Somit ist die Synchronisation mit der Replikation auf logisch gleichen aber eventuell strukturell unterschiedlichen Daten vergleichbar. Haben logisch gleiche Daten unterschiedliche Werte, so liegt im übertragenen Sinne ein Replikationskonflikt vor. Diese Inkonsistenz muß mit Hilfe der Synchronisation beseitigt werden. Eine Synchronisation kann im Dialog mit dem Benutzer erfolgen oder als Batch-Verarbeitung autonom ablaufen. Im Dialog kann der Benutzer die Inkonsistenz manuell beseitigen. Der Batch-Prozeß ist auf sich allein gestellt. Ihm müssen bestimmte Regeln für die Beseitigung der ermittelten Konflikte mitgegeben werden. Oder der Prozeß beschränkt sich darauf, erkannte Inkonsistenz in einer Protokolldatei festzuhalten.

Die hier vorzustellende Prototyp arbeitet im Dialog. In einem Notes Dokument kann der Benutzer Verbindung zu einer Relationalen Datenbank aufnehmen. Die hierzu notwendige Zuordnung der Notes-Felder auf bestimmte Attribute der Relationalen Datenbank gewinnt man aus einem Föderativen Schema. Dieses wurde zuvor vom Administrator definiert. Eine dafür vorgesehene Datenbank liefert ihm hierfür entsprechende Werkzeuge und Hilfsmittel. Das geöffnete Dokument wird dann mit vereinigten Tabellen einer gewählten Relationalen Datenbank synchronisiert. Kommen mehr Datensätze als logisch äquivalentes Pendant des Dokuments in Frage, so wählt der Benutzer einen Datensatz aus. Hierzu wird ihm angezeigt, wie gut jeder einzelne Datensatz mit dem Dokument harmonisiert. Differieren Attribut- und Feldwerte in der per Föderativen Schema definierten Abbildung, so wird die Abbildung als Inkonsistent bezeichnet. Inkonsistenzen können dadurch beseitigt werden, daß entweder der Notes-Wert in die Relationale Datenbank geschrieben wird, oder umgekehrt der Wert aus der Relationalen Datenbank in das Dokument kopiert wird. Ziel ist die Konsistenz zwischen Datensatz und Dokument herzustellen.

7.2 Vorbereitende Arbeiten

Zur Erläuterung der Arbeitsweise des Prototyps soll eine Microsoft Access Datenbank für Adressenverwaltung und die Office-Lösung GroupOffice der Firma Pavone herangezogen werden. Die Access Datenbank besteht aus der Relation „Unternehmen“, in der Kunden und Lieferanten

aufgeführt sind, und der Relation „Kontaktpersonen“, in der die jeweiligen Kontaktpersonen oder Ansprechpartner der Kunden beziehungsweise Lieferanten gespeichert sind.

	Feldname	Felddatentyp	Beschreibung
🔑	ID	Zahl	Identifizierungsnummer
	Firma	Text	
	Strasse	Text	
	Hausnummer	Zahl	
	Ort	Text	
	Postleitzahl	Zahl	
	Land	Text	
	Telefon	Text	
	Telefax	Text	
	Bank	Text	
	BLZ	Text	Bankleitzahl
	Konto	Text	Kontonummer
	Geschaeftsbeziehung	Text	Kunde oder Lieferant

Abb. 7-1: Access Relation: Unternehmen

Das Attribut ID dient als Fremdschlüssel, um 1:N Beziehungen zwischen Unternehmen und Kontaktpersonen zu beschreiben.

	Feldname	Felddatentyp	Beschreibung
🔑	Kontaktperson ID	AutoWert	Identifizierungsnummer
	ID	Zahl	ID-Nummer der Firma
	Name	Text	
	Vorname	Text	
	Position	Text	Position innerhalb des Unternehmens
	PrivatStrasse	Text	
	PrivatHausnummer	Zahl	
	PrivatOrt	Text	
	PrivatPostleitzahl	Zahl	
	PrivatLand	Text	
▶	PrivatTelefon	Text	

Abb. 7-2: Access Relation: Kontaktpersonen

GroupOffice wird von Pavone wie folgt beschrieben:

GroupOffice ist eine Anwendung für ein effektives und effizientes Informationsmanagement im Bürobereich, die den Schriftverkehr innerhalb eines Unternehmens auf allen notwendigen Ebenen optimiert und wieder überschaubar macht. Jedem Mitarbeiter steht dabei die gesamte Korrespondenz von der Adresse eines Kunden bis zum standardisierten Geschäftsbrief zur Verfügung. So können gezielt die benötigten Informationen genutzt und die Abwicklung des Schriftverkehrs beschleunigt werden.

Da das Beispiel ein Abgleich zwischen Adreßinformationen zeigen soll, konzentrieren wir uns auf die Adressendokumente von GroupOffice. Nachfolgend soll ein Ausschnitt der Adreßmaske im Designmodus mit allen uns interessierenden Feldern gezeigt werden.

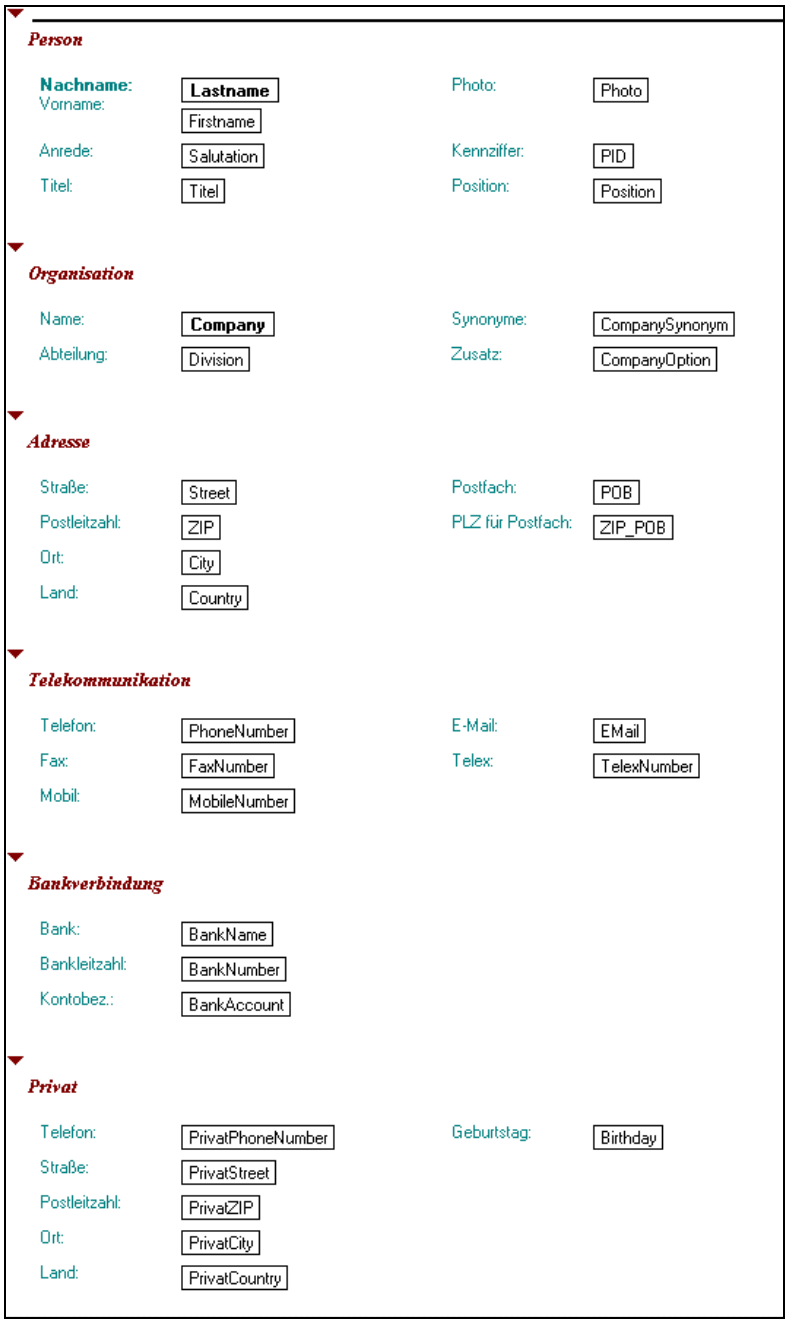


Abb. 7-3: Adressendokument von GroupOffice im Designmodus

Der eigentliche Prototyp setzt sich aus der Maske „$DialogConnectivity“ und dem Agenten „Synchronisation mit RDB“ zusammen. Beide Designelemente müssen der GroupOffice Datenbank hinzugefügt werden. Das Föderative Schema wird in der sogenannten Metadatenbank erzeugt. Neben ihr muß auf dem Notes-Workspace die Formeldatenbank plaziert werden. Letztere stellt der Metadatenbank Formeln zur Schemaübersetzung zur Verfügung. Damit GroupOffice auf das Föderative Schema der Metadatenbank und die Metadatenbank ihrerseits auf die Formeln der

Formeldatenbank zugreifen kann, muß bei der Meta- sowie bei der Formeldatenbank einmalig der Agent „Metadatenbank bekanntmachen“ beziehungsweise „Formeldatenbank bekanntmachen“ ausgeführt werden. Dieser Agent schreibt den Server- und Pfadnamen der jeweiligen Datenbank in eine dafür vorgesehenen Environmentvariable der „Notes.ini“ Datei. Dadurch geben die Datenbanken ihren Zugriffspfad preis.

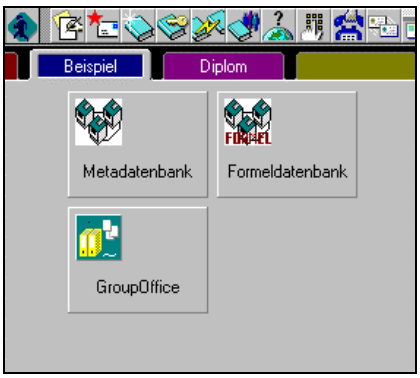


Abb. 7-4: Notes-Workspace mit allen erforderlichen Datenbanken

Der Zugriff auf die Access-Datei erfolgt per ODBC (*Siehe 5.2.5* ***Die Call-Schnittstelle ODBC***). Der entsprechende ODBC-Single-Tier Treiber wird automatisch bei der Definition des Föderativen Schemas innerhalb der Metadatenbank installiert. Hierzu ist jedoch der Pfad des ODBC-Managers in die Environmentvariable „$ODBCManagerPath“ der „Notes.ini“ Datei einzutragen. Zusätzlich muß zur Importierung von Notes-Masken die DLL „readnsf.dll“ ins Notes-Verzeichnis kopiert werden. Abschließend kann dem Notes-Workspace ein SmarIcon hinzugefügt werden, mit dem der Pfad von beliebigen Notes Datenbanken über Environmentvariablen weitergegeben werden kann. Der nachfolgende Code kann über den Menüaufruf File/Tools/Smarticons einem freien Icon zugeordnet werden:

```
ENVIRONMENT DBSCANServer := "Local";
ENVIRONMENT DBSCANPath := "nicht definiert";
server := @If( @Subset( @DbName ; 1 )= ""  ; "Local" ; @Subset(@DbName;1)  ) ;
path := @Subset( @DbName ;-1 );
antwort := @If( path="";
@Prompt([OK];"Datanbank Connectivity ";  "Öffnen Sie die gewünschte Datenbank,
um sie in die in die Konfigurationsdatenbank einzulesen !"  )-1;
@Prompt([YESNO]; "Datanbank Connectivity ";  "Soll die Datenbank " + path + "
in die Konfigurationsdatenbank eingelesen werden ?"  ) );
@If( antwort ; @SetEnvironment( "DBSCANPath" ; path ) ; @Success );
@If( antwort; @SetEnvironment( "DBSCANServer" ; server); @Success )
```

7.3 Föderatives Schema in Metadatenbank erstellen

Um Adressen zwischen der Access-Datenbank und GroupOffice abgleichen zu können, muß genau festgelegt werden welche Felder auf welche Attribute abgebildet werden sollen. Diese Zuordnung erfolgt in einem Föderativen Schema. Das Föderative Schema muß mit einem einheitlichen Datenmodell, auch CDM (Common Data Model) genannt, innerhalb der Metadatenbank beschrieben werden (*Siehe 3.4* ***Fünf-Schichten-Architektur für FDBS***). Dazu wird im ersten Schritt das Schema der Relationalen Datenbank und als Schemaersatz das Maskendesign der Notes Datenbank importiert. Die darauffolgende Schemaergänzung versucht Information über die Datenstruktur der Notesdatenbank, die aus den Notes-Masken gewonnen wurden, zu vervollständigen. Im letzten Schritt werden einzelne Notes-Felder den jeweiligen Attributen der Relationalen Datenbank zugewiesen. Mit Hilfe eines Konzepts bestehend aus Transformations- und Verschmelzungsregeln wird die Schemaübersetzung, -definition und -integration des Buttom-Up-Verfahrens nachgebildet (*Siehe 4.1* ***Der Bottom-Up Entwicklungsprozeß***). Neben dem Föderativen Schema kann in Anlehnung an das Modell zur Spezifizierung von Datenbankbeziehungen (*Siehe 4.4.3* ***Modell zur Spezifizierung von Datenbankbeziehungen***) jeder Abbildung eine Konsistenzregel zugeordnet werden. Diese definiert bestimmte Spielräume bei der Konsistenzbewertung. Zusätzlich können Schreiboperationen auf die eine oder die andere Datenbank beschränkt werden.

7.3.1 Das CDM (common data modell)

Die Aufgabe eines CDMs wurde bereits ausreichend erläutert (*Siehe 3.4* ***Fünf-Schichten-Architektur für FDBS***). Knapp zusammengefaßt dient es als gemeinsames Datenmodell für die gesamte Föderation und sollte so möglichst reich an semantischer Beschreibungskraft sein. In unserem speziellen Fall soll eine Access -Datenbank und eine Notes Datenbank in der Föderation vereint werden. Das vom Prototyp verwendete CDM orientiert sich am Relationalen Modell. Die Beschreibung einer Relation erfolgt nicht mit Hilfe von Tabellen, sondern den Designmöglichkeiten von Notes Datenbanken angepaßt, aus einer Reihe von Anwort-Dokumenten. Jedes Anwort-Dokument steht für ein CDM-Attribut, das sich aus einem Feld der Notes Datenbank und mehreren Attributen der Relationalen Datenbank zusammensetzt. Die Beschränkung auf nur ein Feld wurde aus Gründen der Komplexitätsreduktion gewählt und ist

somit keine Beschränkung des Modells selber. So kann beispielsweise das Feld „Kundenname“ auf die Attribute „Vorname“ und „Nachname“ abgebildet werden.

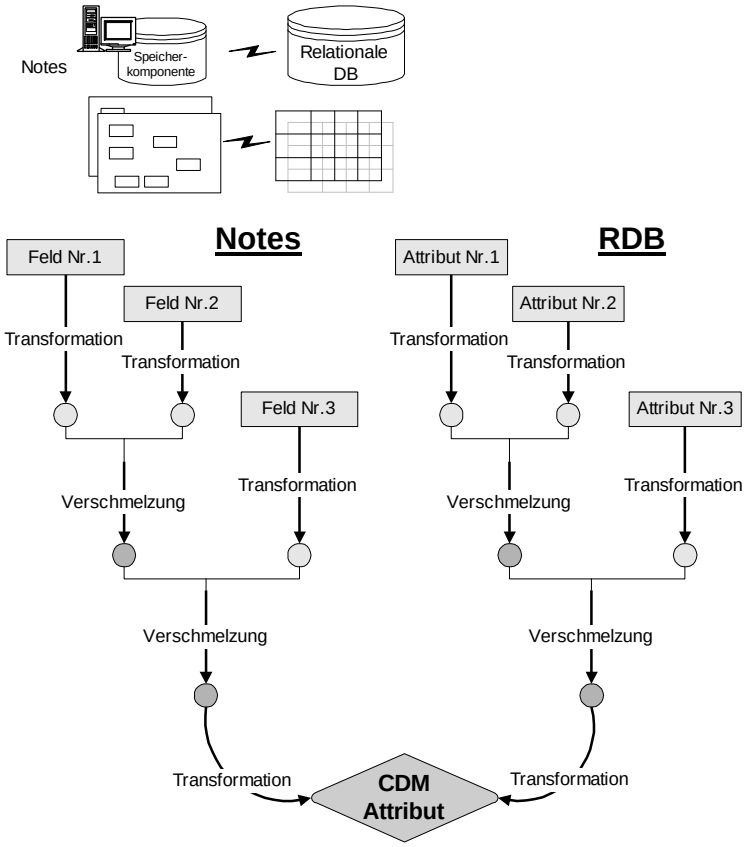


Abb. 7-5: Die Abbildung des CDM-Attributs

Die Abbildung wird durch Zuweisung von Transformations- und Verschmelzungsregeln vervollständigt. Jedem Feld/Attribut kann optional eine Transformationsformel zugewiesen werden. Danach sorgen Verschmelzungsregeln dafür, daß die Felder/Attribute paarweise nacheinander verknüpft werden. Als Resultat verbleibt ein verschmolzener Wert, der erneut transformiert werden kann. So wird das CDM-Attribut einmal über Notes Felder und ein zweites Mal über Attribute der Relationalen Datenbank ermittelt. In unserem Beispiel durchläuft das Feld „Kundenname“ eine Transformationsformel, bei der eine eventuelle Anrede gelöscht wird. Ein konkreter Wert für Kundenname wie beispielsweise „Herr Jens Winkelmann“ würde zu „Jens Winkelmann“ transformiert. Die Relationale-Seite kommt im Beispiel ohne Transformationsformeln aus. Das Vorhandensein von zwei Attributen verlangt jedoch eine Verschmelzungsregel, die die Attribute „Vorname“ und „Nachname“ getrennt durch ein Leerzeichen miteinander verkettet. So würde man den Wert „Jens Winkelmann“ aus den beiden

Attributwerten „Jens“ und „Winkelmann“ erhalten. Das Problem der inversen Transformation und Verschmelzung wurde bereits im theoretischen Teil abgehandelt (*Siehe 4.2* ***Schemaübersetzung***). Nicht immer können befriedigende Lösungen gefunden werden. Möchte man beispielsweise den CDM-Wert „Jens Winkelmann“ zurück in sein original Feldwert transformieren, so kann die dazu notwendige Anrede nicht aus dem Wert selber hergeleitet werden. Aus diesem Grund verwehrt der Prototyp bei nicht inversen Transformations- und Verschmelzungsregeln das Zurückschreiben auf die entsprechende Datenbankkomponente.

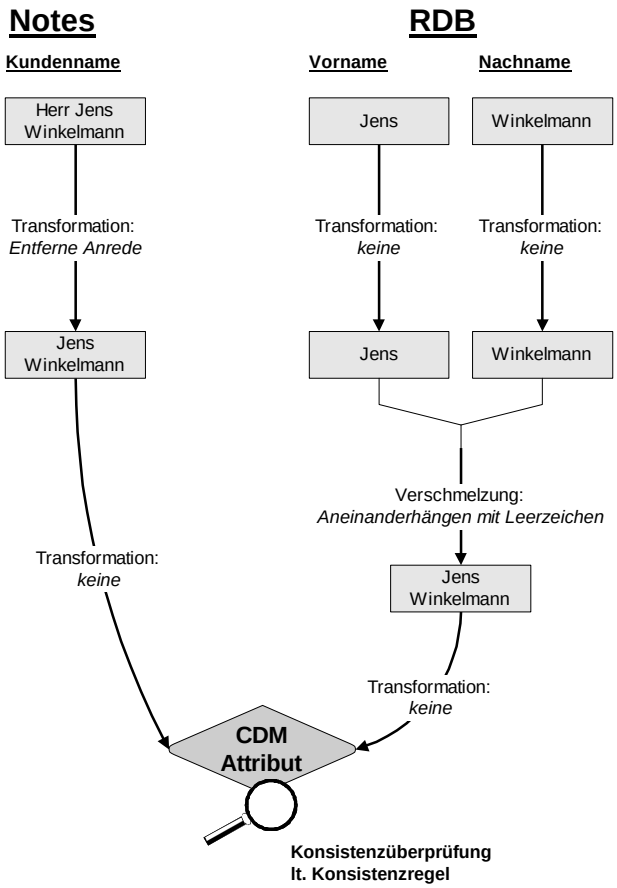


Abb. 7-6: Beispiel: Transformations- und Verschmelzungsregeln

Die Abbildung von Attributen und Feldern mittels Transformations- und Verschmelzungsregeln entspricht der Schemaübersetzung und Schemaintegration des Bottom-Up-Verfahrens (*Siehe 4.1* ***Der Bottom-Up Entwicklungsprozeß***). Daneben bestimmen frei definierbare Konsistenzregeln, wann der aus Attributen hergeleitete CDM-Wert und der aus Feldern hergeleitete CDM-Wert als Konsistent beziehungsweise Inkonsistent gilt.

7.3.2 Schemaimportierung

Bei der Schemaimportierung sollen die notwendigen Teilschemata der beteiligten Datenbankkomponenten eingelesen werden, damit aus ihnen innerhalb der Definition des Föderativen Schema das gewünschte Föderative Schema konstruiert werden kann. Somit stellt die Schemaimportierung die Schemadefinition des Bottom-Up-Verfahrens dar (*Siehe 4.1* ***Der Bottom-Up Entwicklungsprozeß***). Da Notes Datenbanken keinen Datenbankkatalog besitzen, aus dem das entsprechende Datenbankschema abzufragen ist, nimmt man sich die Strukturinformationen des Maskendesigns zur Hilfe. Dessen Problematik wurde bereits ausführlich bei der Abgrenzung von Notes zu Relationalen Datenbanken diskutiert (*Siehe 6.2.2* ***Art der Datenspeicherung***). Die daraus gewonnenen Erkenntnisse werden bei der Schemaergänzung berücksichtigt. Die Schemaergänzung folgt der Schemaimportierung und wird ausführlich im nachfolgenden Unterkapitel beschrieben. Somit beschränkt sich die Schema importierung bei Notes Datenbanken auf das Einlesen des Designs der Masken.

7.3.2.1 Einlesen der Notes Masken

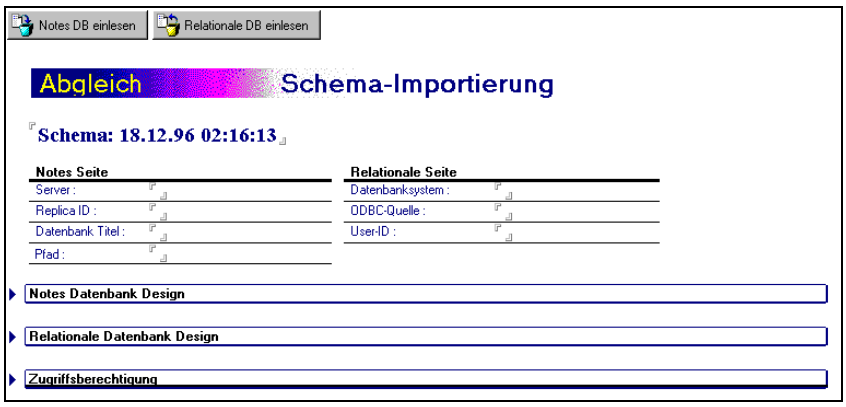


Abb. 7-7: Dokument zur Schemaimportierung

Über den Menüaufruf „Create - Notes—RDB Ableich“ der Metadatenbank wird das Dokument für die Schemaimportierung geöffnet. Das Einlesen des Maskendesigns einer beliebigen Notes Datenbank wird mit dem Mausklick auf den linken Button <Notes DB einlesen> gestartet. Daraufhin erscheint folgende Dialogbox, die dem Benutzer auffordert den korrekten Pfad der entsprechenden Datenbank einzugeben.

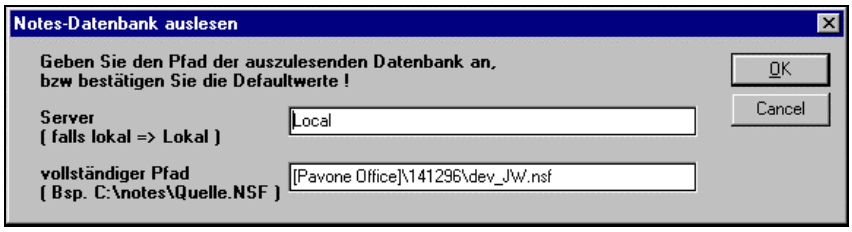


Abb. 7-8: Dialogbox: Notes Datenbank einlesen

Wurde vorher mit Hilfe des angelegten Smarticons der Pfad der gewünschten Datenbank ausgelesen, indem man die Datenbank öffnet und das Smarticon anklickt, so ist wie in der oberen Abbildung der entsprechende Pfad bereits eingetragen (*Siehe 7.2 Vorbereitende Arbeiten*). Bestätigt man mit <OK>, so werden die einzelnen Maskennamen und Teilmaskennamen der Datenbank ausgelesen und zusammen mit allgemeinen Informationen über die Datenbank ins Dokument geschrieben. Gleichzeitig wird im Hintergrund für jede Maske beziehungsweise Teilmaske ein Anwort-Dokument angelegt, das als Grundlage für die spätere Schemaergänzung dient.

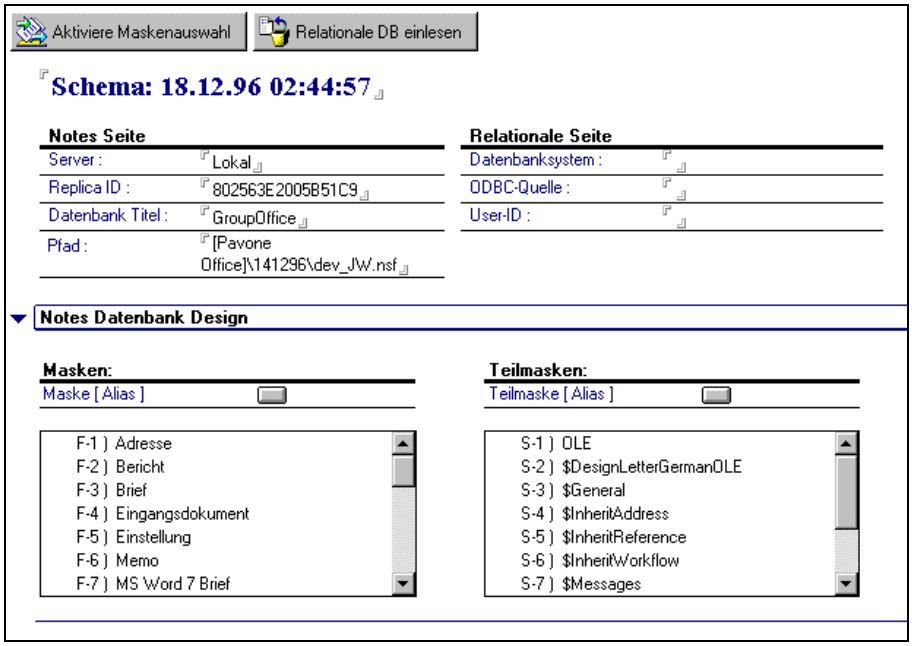


Abb. 7-9: Auflistung aller eingelesenen Masken und Teilmasken

Klappt man der Abschnitt „Notes Datenbank Design“ auf, so kann man hier die Maske auswählen, die zur Definition der Föderation herangezogen werden sollen. Mit den ebenfalls aufgeführten Teilmasken könnte man bei der Erweiterung des Prototyps die gewünschten Felder auf die einer

Teilmaske beschränken. Letzteres ist zur Zeit jedoch noch nicht realisiert. Für unser Beispiel wählen wir die gewünschte Maske „Adresse“ und bestätigen dann anschließend mit dem Button <Maskenauswahl aktivieren>. Hierdurch kann das zugeordnete Anwort-Dokument, mit der später die Schemaergänzung durchgeführt wird, über eine Ansicht aufgerufen werden[i].

7.3.2.2 C-API DLL

Programmtechnisch ruft der LotusScript-Code, der über <Notes DB auslesen> gestartet wird, eine DLL auf. Die DLL ist ein Notes C-API-Programm, das das Maskenkendesign der per Parameterübergabe angegebenen Datenbank ausliest. Da LotusScript kein Zugriff auf das Maskendesign erlaubt, kann diese Aufgabe nur von einem API Programm durchgeführt werden. Vorteile und Nachteile bei der Programmierung mit der API im Vergleich zu LotusScript wurden bereits ausführlich behandelt (*Siehe 6.3.1.3* ***Notes API***). Der DLL-Code basiert auf den Arbeiten von Choukri Drira aus seiner Diplomarbeit mit dem Thema „Information sharing and archiving in heteregeneous database environments“.[ii] Dieser Code, basierend auf 16 Bit für Lotus Notes Version 3, wurde im Rahmen dieser Arbeit zur aktuellen Lotus Notes Version 4, basierend auf 32 Bit, portiert und entsprechend den neuen Anforderungen erweitert. So daß der überarbeitete und erweiterte Code auch mit neuen Notes-Features wie den Teilmasken zurechtkommt. Die grobe Arbeitsweise der DLL ist geblieben. Für jede gefundene Maske wird ein Anwort-Dokument des geöffneten Dokuments angelegt. Im Anwort-Dokument werden Informationen über die Designelemente der jeweiligen Maske vermerkt, wie beispielsweise alle Felder mit Namen und Typ. Das geöffnete Dokument (Schemaimportierung) erfaßt nur den Namen und Aliasnamen der gefundenen Masken beziehungsweise Teilmasken.

[i] Es wird ein bestimmtes Flag im Anwort-Dokument gesetzt, so daß es durch die Auswahlformel der Ansicht schlüpft.
[ii] Siehe [DC96]

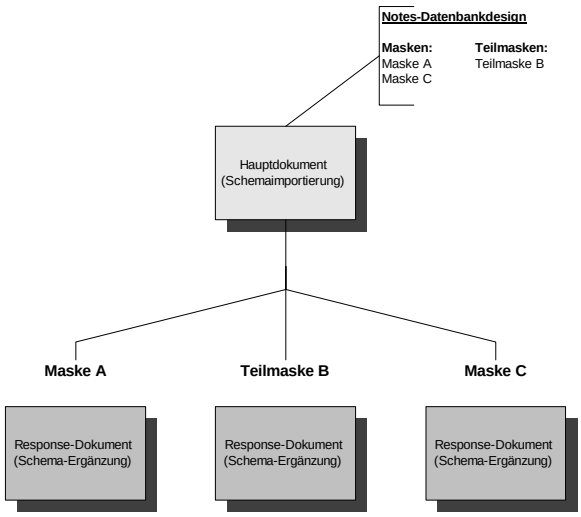


Abb. 7-10: API-DLL erzeugt Anwort-Dokumente

Das Auslesen des Maskendesigns geschieht folgendermaßen. Aus allen Dokumenten der Datenbank werden diejenigen herausgefiltert, die das Maskendesign speichern. Diese Dokumente haben ein Rich Text Feld mit dem Namen „$Body“, indem die eigentlichen Designelemente wie beispielsweise Felder, statischer Text usw. in den einzelnen CD-Records gespeichert sind (*Siehe 6.2.2* ***Art der Datenspeicherung***). Nun werden alle CD-Records nacheinander durchsucht. Findet man solche vom Typ „Field“, so hat man ein Feld gefunden. Der im CD-Record gespeicherte Feldname und Feldtyp wird herausgelesen. Ein anderer Typ von CD-Record markiert, daß alle nun folgenden Designelemente von einer eingesetzten Teilmaske stammen. Das Ende der Teilmaske wird durch ein weiteres spezielles CD-Record gekennzeichnet. Alle dazwischen liegenden Felder sind somit Felder der eingesetzten Teilmaske. Errechnete Teilmasken erweitern erst beim Aufruf der Maske dessen Design. Dabei bestimmt eine entsprechende Formel, ob oder welche Teilmaske im konkreten Fall gewählt wird. Diese Formel ist im CD-Record für errechnete Teilmasken gespeichert. Sie wird von der DLL ausgelesen, um später bei der Schemaergänzung Hilfe zu geben.

Neben den normalen Feldern gibt es sogenannte Shared-Fields, die zentral definiert werden und dann ähnlich wie eingesetzte Teilmasken ins Maskendesign importiert werden können. In der Notes-Architektur sind sie mit Maskendesign Dokumenten vergleichbar. Ihr „$Body“-Feld enthält jedoch nur die Definition eines Feldes. Findet man ein solches Shared-Field in einer Maske, so wird dessen Datentyp fälschlicherweise immer als Text angegeben, obwohl in Wirklichkeit

eventuell ein anderer Datentyp vorliegt. Den korrekten Datentyp kann man nur über das Designdokument des Shared-Fields ermitteln. Deshalb müssen vor dem Auslesen aller Masken alle Shared-Fields mit Namen und Datentyp aus deren Designdokumenten herausgelesen werden. Trifft man dann beim Inspizieren des „$Body“-Felds im Maskendesign Dokument auf ein Shared-Field, so ist nicht der dort angegebene Datentyp zu wählen, sondern der bereits korrekt ermittelte aus dem Shared-Field-Design.

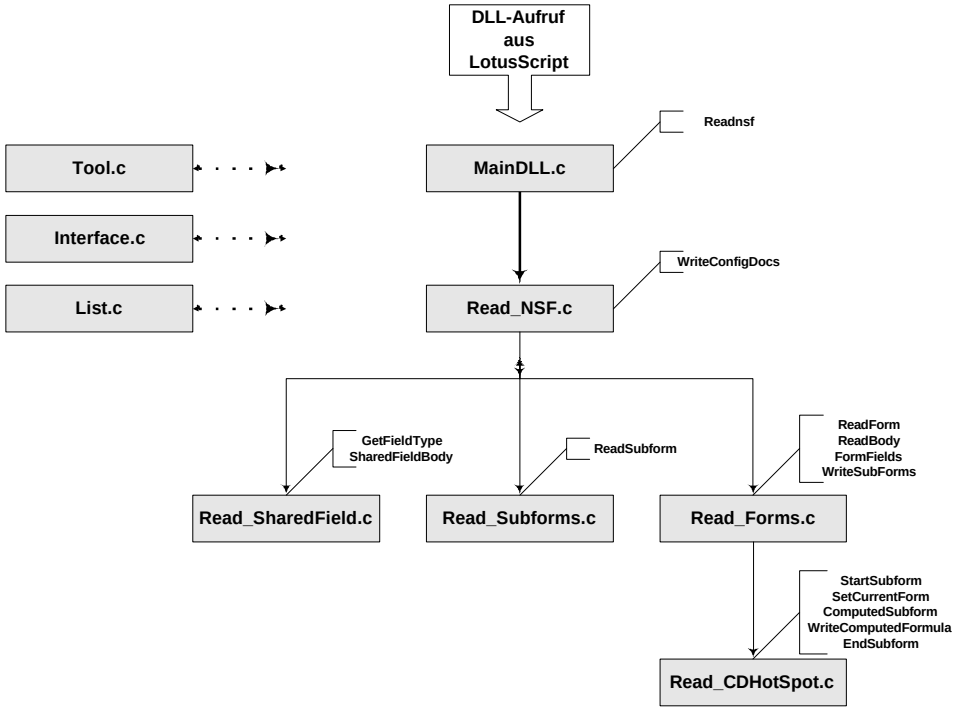


Abb. 7-11: Alle Projektdateien der API-DLL

Der Quellcode der DLL setzt sich aus mehreren Dateien zusammen, die nun nacheinander näher betrachtet werden sollen. Dabei soll nur auf dessen grobe Arbeitsweise und auf etwaige Besonderheiten eingegangen werden, um dem Leser einen Leitfaden für die Durchsicht des Codes zu geben. Der Code selber ist reichlich mit Kommentaren bedacht, so daß der geübte C-API-Programmierer auf keine Schwierigkeiten stoßen dürfte.

? **MainDLL.c**
Diese Datei enthält die Funktion „Readnsf“, die aus LotusScript heraus aufgerufen werden kann. Hierzu muß der Rückgabewert in der Funktionsdeklaration der zugehörigen Header-Datei „__declspec(dllexport)“ heißen. Als Parameter wird der Pfad der auszulesenden Datenbank, der Pfad zur Metadatenbank sowie die UniversalNoteID des Dokuments „Schemaimportierung“ übergeben. Der vierte Parameter hat keine Bedeutung. Als erstes wird

die als String übergebene UniversalNote-ID in eine Notes-API gerechte Datenstruktur umgewandelt. Hierzu wird die Funktion „UNIDString2API“ aus der Datei „Tool.c“ aufgerufen. Danach wird eine Notes-Session gestartet, um daran anschließend die Metadatenbank und die auszulesende Datenbank zu öffnen. Über den Aufruf der Funktion „WriteConfigDocs“ wird quasi das Hauptprogramm in der Datei „Read_NSF.c“ gestartet.

? **Read_NSF.c**
Zuerst wird der zusammengesetzte Datentyp „INFOSTRUCT“ mit Daten gefüllt, die im weiteren Verlauf von den jeweiligen Unterfunktionen genutzt werden. Hierzu wird immer der Pointer auf die Datenstruktur als Parameter übergeben. Danach werden die Datentypen aller Shared-Fields der Datenbank ermitteln. Hierzu erfolgt ein Aufruf der Funktion „GetFieldTyp“ aus der Datei „Read_SharedField.c“. Anschließend wird für jede Maske die Funktion „ReadSubforms“ der Datei „Read_Subforms.c“ aufgerufen. Abschließend wird ebenfalls für jede Maske die Funktion „ReadForm“ der Datei „Read_Forms.c“ gestartet.

? **Read_SharedField.c**
Die beiden Funktionen „GetFieldTyp“ und „SharedFieldBody“ dieser Datei wurden unverändert aus dem Originalcode übernommen. Sie dienen zur Ermittlung des Datentyps von Shared-Fields.

? **Read_Subforms.c**
Die Funktion „ReadSubforms“ ermittelt alle Teilmasken der Datenbanken und numeriert diese fortlaufend durch. Hierdurch wird jeder Teilmaske eine eindeutige Nummer zugewiesen.

? **Read_Forms.c**
Die Funktion „ReadForm“ wird pro Maskendokument aufgerufen. Als erstes erzeugt sie ein Anwort-Dokument, indem alle Designelemente der jeweiligen Maske abgelegt werden sollen. Dazu wird für jedes „$Body“-Item[i] die Funktion „ReadBody“ aufgerufen. Innerhalb von „ReadBody“ wird wiederum für jedes CD-Record die Funktion „FormFields“ aufgerufen. „FormFields“ überprüft als erstes den Typ des CD-Records. Falls es sich um ein Feld handelt, wird unter anderem der Feldname und Feldtyp sowie der zugeordnete Name und Aliasname der Maske im Anwort-Dokument vermerkt. Ist das Feld gleichzeitig ein Shared-Field, so wird, statt des Feldtyps aus dem CD-Record, der bei der Durchsicht der Shared-Fields in der Datei „Read_SharedField.c“ verwendet. Markiert das CD-Record jedoch Anfang oder Ende einer Teilmaske, so wird entsprechend die Funktion „StartSubform“ beziehungsweise „EndSubform“ aus der Datei „Read_CDHotSpot.c“ aufgerufen.

? **Read_CDHotSpot.c**
Die Funktion „StartSubform“ überprüft ob es sich um den Anfang einer eingesetzten Teilmaske (Inserted Subform) oder den Anfang einer errechneten Teilmaske (Computed Subform) handelt. In beiden Fällen wird mittels der Funktion „SetCurrentForm“ unter anderem die entsprechende Teilmaske als zugeordnete Maske aller nun folgenden Felder gekennzeichnet. Im Falle einer errechneten Teilmaske wird danach die Funktion „CumputedSubform“ aufgerufen. Diese dekompiliert die entsprechende Formel zur Berechnung der Teilmaske und legt sie zu den anderen ausgelesenen Designelementen. Letzteres erfolgt über die Funktion

[i] Rich Text-Felder können sich aus mehreren Items zusammensetzen.

„WriteComputedFormula“.
Im Falle daß der CD-Record-Typ das Ende einer Teilmaske markiert, wird anstatt der Funktion „StartSubform“ die Funktion „EndSubform“ aufgerufen. Als zugeordnete Maske aller nun folgenden Felder wird wieder die ursprüngliche Maske bestimmt.

? **Tool.c**
In dieser Datei sind diverse selbstgeschriebene Hilfsfunktionen abgelegt, die von anderen Funktionen aufgerufen werden. Die Funktionsnamen „UNIDString2API“, „UNIDAPI2String“, „UNID2NoteHandle“, „NoteCreateResponseByHandle“, „TranslateFlag“ und „FindFormCount“ sind selbsterklärend. Auf sie soll im Rahmen dieser Arbeit nicht weiter eingegangen werden.

? **Interface.c**
Alle API-Aufrufe, die direkt auf das Hauptdokument (Schemaimportierung) und den neu erzeugten Anwort-Dokumenten (Schemaergänzung) operieren, wurden aus den anderen Funktionen herausgenommen und speziellen Funktionen dieser Datei zugeordnet. Damit wurde versucht eine strikte Trennung zwischen normalem Code und Code, der sich mit der Abspeicherung der ausgelesenen Designelemente beschäftigt, zu vollziehen.

? **List.c**
Die Hilfsfunktionen dieser Datei erzeugen und bearbeiten Daten der Datenstruktur Liste. In Listen werden unter anderem Teilmasken, Shard-Fields und Aliasnamen zwischeng espeichert.

Allen hier aufgeführten Dateien ist eine Header-Datei mit gleichem Namen zugeordnet, die jedoch wie üblich auf „.h“ endet. Zusätzlich gibt es die Header-Datei „ConfiDef.h“, in der alle Notes-Felder als Macros definiert sind, und eine Header-Datei „DataStruct.h“, in der die benutzerdefinierten Datenstrukturen abgelegt sind.

7.3.2.3 Einlesen eines Relationalen Schemas

Nach dem Einlesen der Notes Masken soll nun das Schema der Relationalen Datenbank importiert werden. Hierzu ist der Button <Relationale DB einlesen> zu wählen. Dadurch wird parallel der ODBC-Manager und eine Dialogbox aufgerufen. Falls noch keine ODBC-Quelle für die Relationale Datenbank angelegt ist, kann dieses mit Hilfe des Managers nachgeholt werden. In der Dialogbox ist in diesem Fall der Punkt „<Neuangelegte Datenquelle>“ zu wählen. Daneben listet die Dialogbox alle bereits angelegten ODBC-Quellen auf, die alternativ gewählt werden können. Zusätzlich ist die Art der Datenbank anzugeben. Hierdurch wird die korrekte Datentypabbildung verwendet (*Siehe 7.3.2.5 Programmiertechnische Realisierung*). In unserem Beispiel handelt es sich um eine Access-Datenbank. Andere Datenbanken erfordern zusätzlich die Angabe der UserID und eines Passworts.

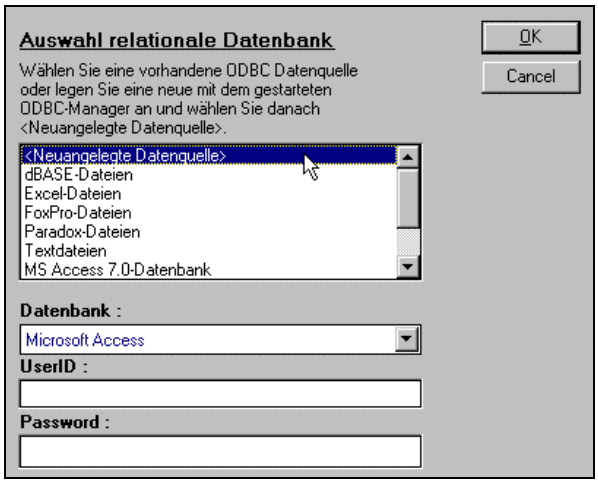


Abb. 7-12: Dialogbox: Auswahl der ODBC-Quelle

Wendet man sich dem Abschnitt „Relationale Datenbank Design“ zu und wählt den Button neben „Auswahl Tabellen“, so erscheint nachfolgende Dialogbox. In ihr sind alle Tabellen der Access-Datenbank aufgeführt. Uns interessieren die Relationen „Kontaktpersonen“ und „Unternehmen“, die nach den Systemtabellen aufgelistet sind. Nach dem Schließen der Dialogbox werden alle Felder und deren Datentypen aus der Datenbank ausgelesen.

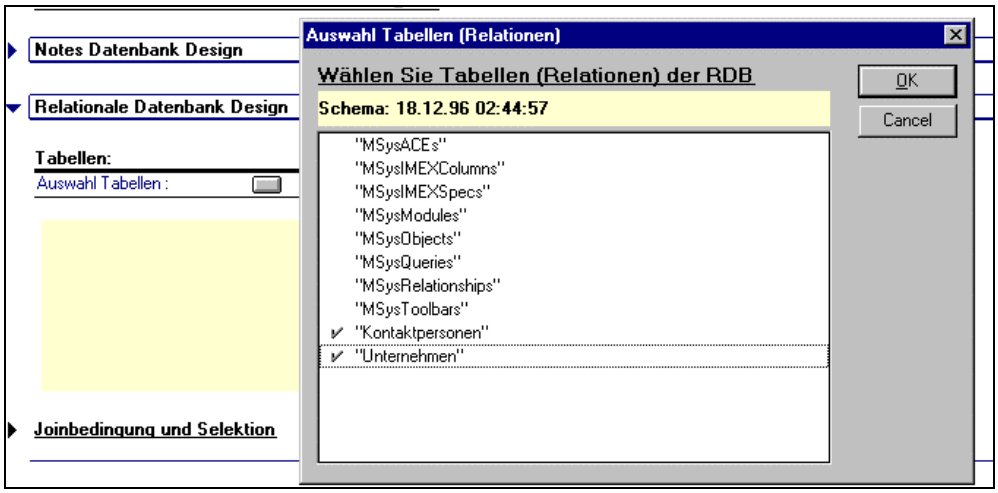


Abb. 7-13: Dialogbox: Auswahl von Tabellen

Rechts neben dem Bereich für Tabellen befindet sich der für die Attribute. Drückt man die als Hotspot angelegt und grün umrahmte Überschrift „Attribute“, so öffnet sich ein Fenster mit der Auflistung aller Attribute der ausgewählten Tabellen.

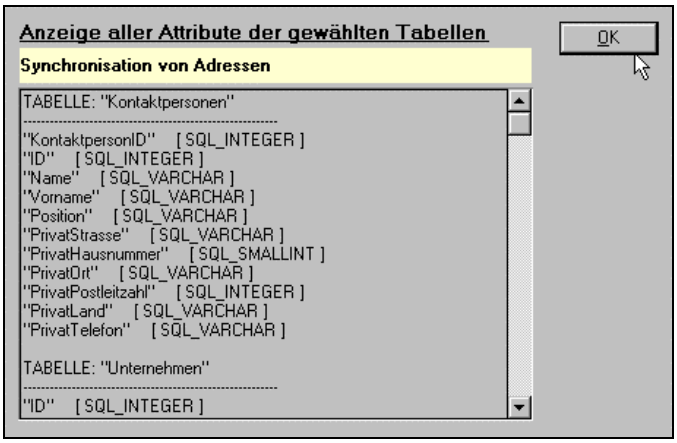


Abb. 7-14: Auflistung aller Attribute

Da es nur erlaubt ist den Notes-Feldern Attribute einer einzigen Tabelle zuzuweisen, müssen die beiden ausgewählten Tabellen zu einer einzigen Relation verbunden werden. Hierzu dient der Abschnitt „Joinbedingungen und Selektion“. In unserem Beispiel müssen beide Tabellen über ihr Attribut „ID“ verbunden werden (Join). Würde man dieses unterlassen, so würde fälschlicherweise automatisch das Katesische Produkt aus beiden Tabellen gebildet. Bei der Auswahl mehrerer Tabellen und deren Vereinigung kann das Problem des aus der Literatur bekannten View-Updates auftreten[i]. Probleme bereiten hier Schreiboperationen, die auf die darunterliegenden Tabellen unterschiedlich abgebildet werden können. So sollte dieser Schritt von Administrator wohlüberlegt sein. Neben einer Joinbedingung darf zusätzlich eine Selektionsbedingung formuliert werden, um die Anzahl der Datensätze von vornherein zu verringern. In unserem konkreten Beispiel würde sich die Selektionsbedingung „Geschaeftsbeziehung = Lieferant“ anbieten, da die Synchronisation nur mit Lieferanten nicht mit Kunden erfolgen soll. Eine Dialogbox für die Formulierung der Joinbedingung sowie der Selektionsbedingung wird per Mausklick auf die jeweiligen Button gestartet. Nach korrekter Eingabe werden die Bedingungen im Anzeigefeld des Abschnitts aufgelistet. Über den Button <Löschen> können beliebige Bedingungen wieder entfernt werden.

[i] Siehe [EN94] Seite 217 und 218 - 7.4.3 Updating of Views and View Implementation

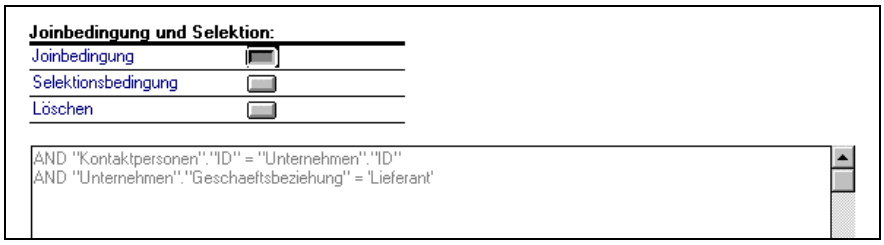


Abb. 7-15: Joinbedingung und Selektionsbedingung

Abschließend sollten noch die Primärschlüssel der aus dem Joinprozeß hervorgegangenen Tabelle angegeben werden. Hierzu ist im Bereich „Attribute“ der Button <Primärschlüssel angeben> zu wählen. In der daraufhin gestarteten Dialogbox wählen wir das Attribut „KontaktpersonenID“ der Relation „Kontaktpersonen“.

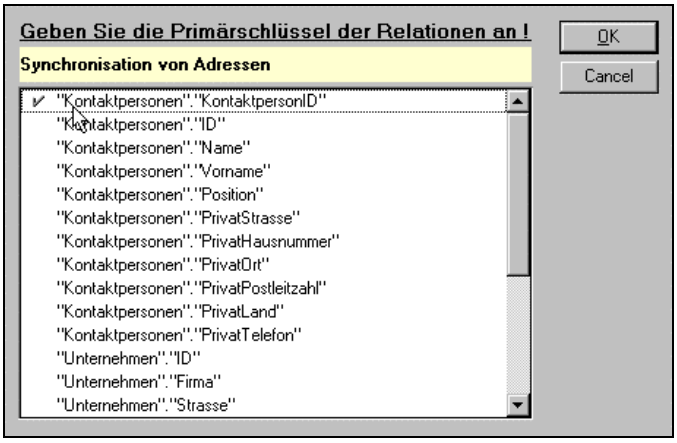


Abb. 7-16: Dialogbox: Angabe der Primärschlüssel

7.3.2.4 Schreibberechtigung bei Konsistenzwiederherstellung

Im letzten Abschnitt des Schemaimportierungs Dokuments kann die Schreibberechtigung bezüglich der Relationale Datenbank und der Notes Dokument bestimmt werden. Jeder Datenbankkomponente kann man das Merkmal „Schreibverbot“, „Schreibschutz“ oder „Schreibberechtigung“ zuordnen. „Schreibverbot“ verhindert jegliche Schreiboperationen auf die Datenbank beziehungsweise dem Dokument. Beim gesetzten Merkmal „Schreibschutz“ wird der Benutzer bei Schreiboperationen nur über dessen Vorhandensein aufgeklärt, so daß er die Ausführung der Operation nochmals überdenken kann.

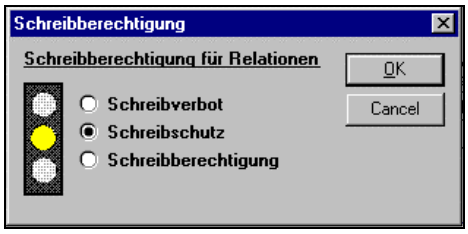


Abb. 7-17: Dialogbox: Schreibberechtigung für Datenbankkomponente

Hat die Relationale Datenbank einen „Nur-Lese“ Zugriff, sollte die entsprechende Schreibberechtigung auf „Schreibverbot“ gesetzt werden.

7.3.2.5 Programmiertechnische Realisierung

Das Auslesen von Tabellen, Attributen und deren Datentypen erfolgt über LotusScript-Actions. Der eigentliche Zugriff auf die ODBC-fähige Datenbank wird über die LotusScript-Erweiterung LotusScript Data Object (LS:DO) realisiert, die im sechsten Kapitel diskutiert wurde (*Siehe 6.3.1.2.2* ***LotusScriptEXtension (LSX)***). Der LotusScript-Code, der sich hinter dem Button <Relationale DB einlesen> verbirgt, stellt eine Verbindung zur ODBC-Datenbank her. Das dafür notwendige LS:DO-Objekt wird als globale Variable gespeichert, so daß beim nachfolgenden Auslesen der Attribute die vorhandene Verbindung erneut genutzt werden kann. Die entsprechenden Attribute werden erst nach der Auswahl der gewünschten Tabellen eingelesen. Dieses ist notwendig, da das Einlesen aller Attribute aller Tabellen zu einer nicht akzeptablen Laufzeit führen würde. Die LS:DO Methode „FieldNativeDataType“ liefert den Datentyp des Attributs in Form einer Kennzahl. Die Einleseroutine weist der Kennzahl einen Namen zu. Die Namen für die einzelnen Datentypen können für jede Datenbank gesondert definiert werden. Hierzu dient ein „Datentypabbildungsdokument“, das für jede Art von Datenbank angelegt werden muß. Über den Menüaufruf View/Konfiguration/Datentypabbildung kann es geladen und modifiziert werden.

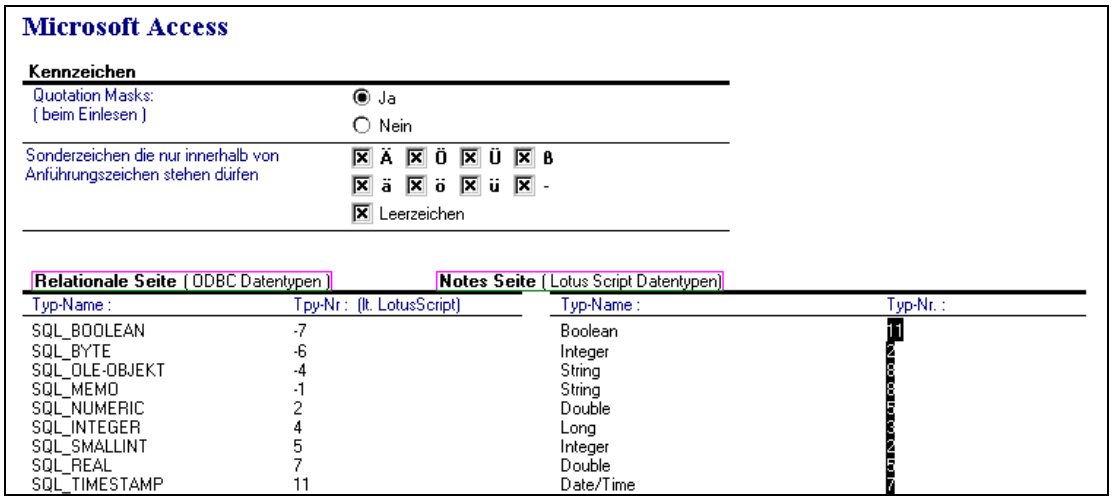


Abb. 7-18: Datentypabbildungs-Dokument für Access-Datenbank

Zusätzlich wird in diesem Dokument jedem ODBC-Datentyp ein entsprechender LotusScript Datentyp zugeordnet. Letzterer schränkt später bei der Spezifikation eines CDM-Attributs die Auswahl von Transformations- und Verschmelzungsformeln ein (*Siehe 7.3.4.5 Programmiertechnische Realisierung*). Die explizite Definition von Datentypabbildungen ist notwendig, da man nicht voraussetzen kann, daß jeder ODBC-Treiber die gleiche Kennzahl für den gleichen Datentyp zurückgibt. Ebenfalls kann im Datentypabbildungs-Dokument bestimmt werden, ob die Namen der einzulesenden Tabellen und Attribute in Anführungszeichen gesetzt werden. Ohne Anführungszeichen könnten bei späteren ODBC-Aufrufen Schwierigkeiten auftreten, falls sich Leerzeichen oder Sonderzeichen in einem angegebenen Namen befinden (*Siehe 7.4.1 Komprimiertes Föderatives Schema bereitstellen*).

7.3.3 Schemaergänzung

In der Ansicht „Schemen nach Name“ und „Schemen nach DB“ erkennt man, daß dem Dokument für die Schemaimportierung ein Anwort-Dokument für die Schemaergänzung zugewiesen wurde.

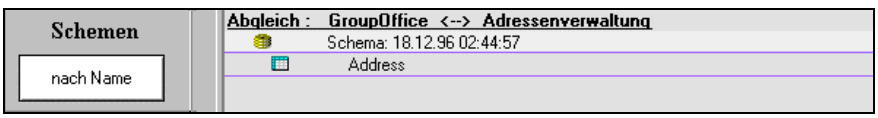


Abb. 7-19: Dokumente für Schemaimportierung und Schemaergänzung in der Ansicht

In diesem Dokument soll aus den gesammelten Designinformationen der Maske ein Schema für alle Adressendokumente von Group-Office konstruiert werden. Die Schwierigkeiten der Herleitung eines Notes-Schemas über die Masken wurden bereits ausführlich in Kapitel 6.2.2 **Art der Datenspeicherung** diskutiert.

7.3.3.1 Werkzeuge für Schemaergänzung

Öffnet man das entsprechende Dokument, so sieht man alle Felder der Adressenmaske getrennt nach Datentyp aufgelistet. Über den Button <Wechsel Ansicht> erhält man eine alternative Darstellung, bei der alle Felder unabhängig vom Datentyp vereint in nur einer Auswahlbox zu finden sind. Die Aufgabe besteht nun darin alle Felder zu wählen, die das zu erzeugende Schema beinhalten soll. Felder mit dem Feldtyp „Berechnet zur Anzeige“ sind durch einen Stern vor ihrer Numerierung gekennzeichnet. Sie sind für die Definition des Schemas unerheblich, so daß ihre Auswahl keine weitere Aktion nach sich zieht.

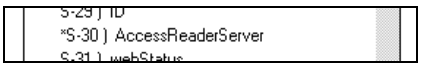


Abb. 7-20: Feld vom Typ „Berechnet zur Anzeige“

Soll das Schema auch Felder von errechneten Teilmasken erhalten, so sind diese zusätzlich zu importieren. Neben der Auflistung aller eingesetzten Teilmasken, deren Felder sich bereits im Dokument befinden, liegt eine Auswahlbox mit allen anderen Teilmasken. Wählt man eine beliebige Zahl von Teilmasken aus dieser Box und bestätigt man die Auswahl mit <Aktiviere Teilmaskenauswahl>, so werden alle Felder der markierten Teilmasken den bereits vorhandenen Feldern hinzugefügt. Drückt man den Button neben „Errechnete Masken [Alias]“, so erhält man eine Dialogbox, die neben allen Teilmaskennamen auch deren Aliasnamen enthält.

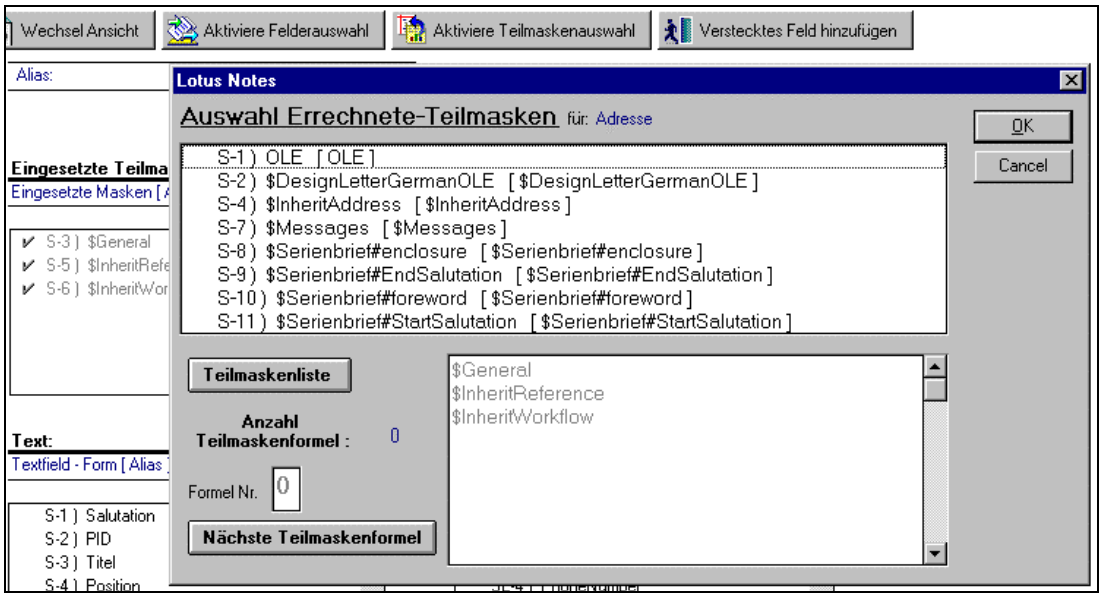


Abb. 7-21: Dialogbox: Importieren von errechneten Teilmasken

In dieser Dialogbox findet der Administrator diverse Hilfsmittel, um aus allen potentiellen Teilmasken diejenigen für die Importierung ausfindig zu machen (*Siehe 6.2.2* ***Art der Datenspeicherung***). So erfährt man wieviele Formeln für errechnete Teilmasken sich im jeweiligen Maskendesign befinden. Die Formeln selber können über den Button <Nächste Teilmaskenformel> nacheinander angezeigt werden. Ein anderer Button mit der Aufschrift <Teilmaskenliste> liefert eine Liste aller Teilmasken, die aus einem gesonderten Feld „$Subforms“ des Masken-Designdokuments ausgelesen wurde. Sie ist nur eingeschränkt nutzbar, da ihre Vollständigkeit nicht garantiert werden kann. Über <OK> verläßt man die Dialogbox. Versteckte, beziehungsweise verborgene Felder können über den Button <Verstecktes Feld hinzufügen> spezifiziert werden (*Siehe 6.2.2* ***Art der Datenspeicherung***). Die Dialogbox verlangt neben der Eingabe des Namens die Zuweisung eines Datentyps.

Abb. 7-22: Dialogbox: Verborgenes Feld spezifizieren

Hat man alle gewünschten importierten, neu angelegten und bereits vorhandenen Felder markiert, so muß man die Auswahl mit <Aktiviere Felderauswahl> bestätigen. Hierdurch wird für jedes Feld ein Antwort-Dokument angelegt, indem jeweils ein CDM-Attribut für das Föderative Schema spezifiziert werden soll. In der Ansicht entdeckt man nun eine zweistufige Hierarchie aus Antwort-Dokumenten.

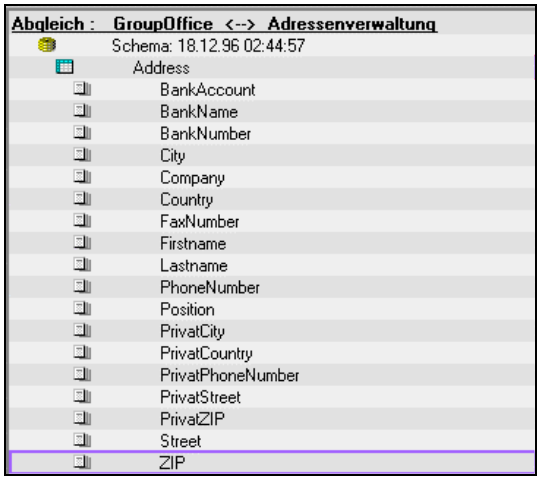


Abb. 7-23: Ansicht mit allen Anwort-Dokumenten

7.3.3.2 Programmiertechnische Realisierung

Alle hier verwendeten Actions sind ausschließlich mit LotusScript programmiert worden. Bei der Importierung einer errechneten Teilmaske wird auf das Anwort-Dokument zugegriffen, das die im Notes-API DLL angelegt hat (*Siehe 7.3.2.2 C-API DLL*). Die dort gespeicherten Informationen über alle Felder der Teilmaske werden dem aktuellen Dokument hinzugefügt. Der Feldnumerierung wird um ein c verbunden mit der lfd. Nummer der Teilmaske verlängert.

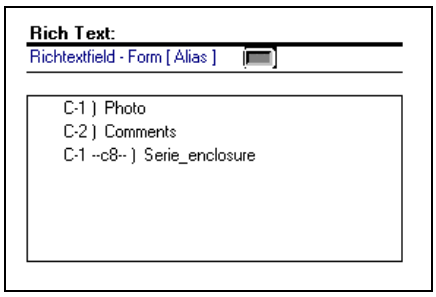


Abb. 7-24: Importiertes Rich Text Feld aus Teilmaske

So kann man zwischen gleichlautenden Feldnamen aus verschiedenen Teilmasken unterscheiden. Entfernt man die Markierung einer errechneten Teilmaske und drückt erneut den Button <Aktiviere Teilmaskenauswahl>, so werden die hinzugefügten Feldinformationen der entsprechenden Teilmaske wieder entfernt. Ein Löschen von hinzugefügten verborgenen Feldern ist zur Zeit nicht möglich. Über den Button <Aktiviere Felderauswahl> wird für jedes neu markierte Feld ein Anwort-Dokument angelegt, dem alle notwendigen Informationen hinzugefügt werden. Wurde die Markierung eines Feldes entfernt, so wird das entsprechende Anwort-Dokument gelöscht.

7.3.4 Definintion des Föderativen Schemas

Die durch die Schemaergänzung erzeugten Anwort-Dokumente bilden das Gerüst für die Definition der CDM-Attribute. Diese wiederum bilden eine Relation des Föderativen Schemas. In der derzeitigen Version des Prototyps wird jedes CDM-Attribut aus nur einem Notes Feld gebildet, welche über die Schemaergänzung den Anwort-Dokumenten bereits zugeordnet sind. Die Aufgabe besteht nun darin, jedem CDM-Attribut zusätzlich die entsprechenden Attribute der Relationalen Datenbank zuzuordnen sowie Transformation-, Verschmelzungs- und Konsistenzberechnungsformeln zuzuweisen. Einige der CDM-Attribute müssen zu Schlüsselattributen gekürt werden. Nur wenn ein Dokument und ein Datensatz identische Werte in ihren Schlüsselattributen aufweisen, sind sie Kandidaten für die Synchronisation. Bei allen anderen CDM-Attributen spezifiziert die Konsistenzberechnungsformel ob Dokument und Datensatz bezüglich des jeweiligen Attributs als Konsistent oder Inkonsistent gelten. Nachfolgend soll im Rahmen unseres Beispiels die Definition folgender Abbildungen exemplarisch gezeigt werden:

- Schlüsselattribut „Company ? Firma“
- Attribut „Street ? „Strasse Hausnummer“ mit Verschmelzungsformel
- Attribut „PhoneNumber“ ? „Telefon“ mit Transformationsformel

Abschließend soll auf die Definition der Konsistenzregeln näher eingegangen werden.

7.3.4.1 Definition eines Schlüsselattributs

Im Beispiel soll der Nachname, Vorname und die Firma als Schlüssel fungieren. Folglich werden Datensätze, die mit dem jeweiligen Dokument in diesen Attributen identisch sind, als Kandidaten für das logische Pendant des Dokuments betrachtet. Nicht immer ist es möglich die Schlüssel so zu wählen, daß dem Dokument nur ein Datensatz gegenübergestellt wird. Falls beispielsweise mehrere Personen mit gleichem Vor- und Nachnamen erfaßt sind, die zusätzlich bei der selben Firma beschäftigt sind, werden sie alle über unsere Schlüsselattr ibute gefunden. So sollte man möglichst eindeutige Schlüsselattribute wie beispielsweise eine Kundennummer wählen. Jedoch müssen Notes Dokumente nicht unbedingt Felder mit Primärschlüsseleigenschaft besitzen, da sie alleine mit Hilfe von System-IDs identifiziert werden. Auch sollte bei der Wahl der Schlüsselattribute beachtet werden, daß sie nunmehr für die eigentliche Synchronisation verloren sind. Wird der Nachname „Meyer“ im Dokument mit „Y“ geschrieben in der Relationalen Datenbank jedoch mit „ei“, so wird der entsprechende Datensatz nicht gefunden, anstatt daß die entsprechende Inkonsistenz erkannt wird.

Für unser Beispiel öffnen wir das Anwort-Dokument „Company“. Unterhalb der Angabe, welche Notes und welche Relationale Datenbank involviert sind, kann man bestimmen, ob man eine Schlüsselbeziehung oder eine normale Abbildungsbeziehung definieren will.

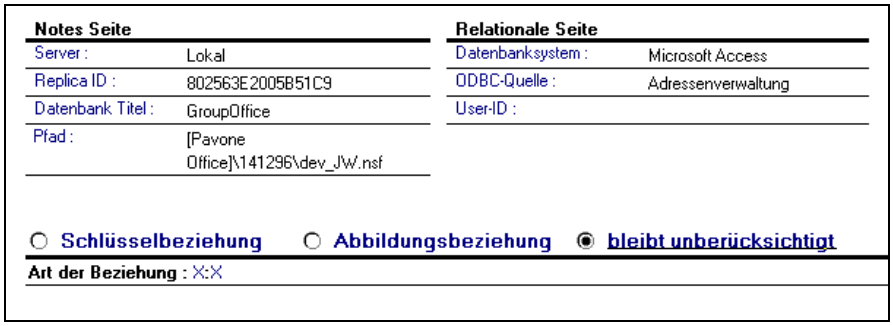


Abb. 7-25: Wahl zwischen Schlüssel- und Abbildungsbeziehung

Klappt man den Abschnitt „Schemaschlüsseldefinition“ auf, so ist der linke Bereich für Lotus Notes reserviert und der rechte Bereich kümmert sich um die Spezifikation der Attribute der Relationalen Datenbank. Ist das hier definierte Feld bei der späteren Synchronisation des Dokuments nicht vorhanden oder ist es leer, so kann hier ein Ersatzwert bestimmt werden. Desweiteren kann dem Feld eine Transformationsformel zugewiesen werden.

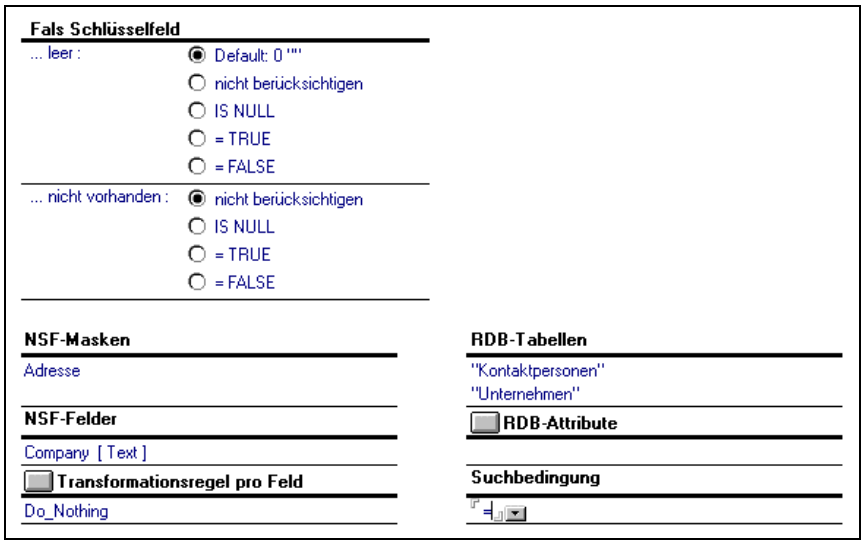


Abb. 7-26: Definition einer Schlüsselbeziehung

Im Falle einer Schlüsselbeziehung kann der Abbildung nur ein Attribut zugewiesen werden. Hierzu ist der Button neben „RDB-Attribute“ zu klicken. In der daraufhin erscheinenden Dialogbox muß für unser Beispiel das Attribut „Firma“ der Relation „Unternehmen“ gewählt werden.

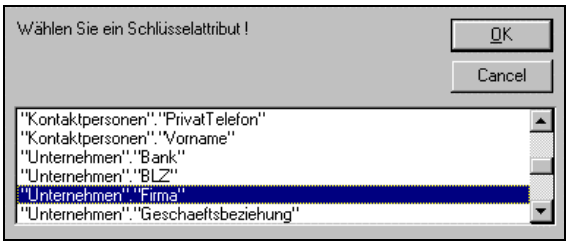


Abb. 7-27: Dialogbox: Auswahl Schlüsselattribut

Abschließend kann der Abbildung eine Suchbedingung zugeordnet werden. Dadurch kann die bei Schlüsselabbildungen ansonsten verlangte Identität auch umdefiniert werden. Nachfolgende Abbildung zeigt alle Wahlmöglichkeiten.

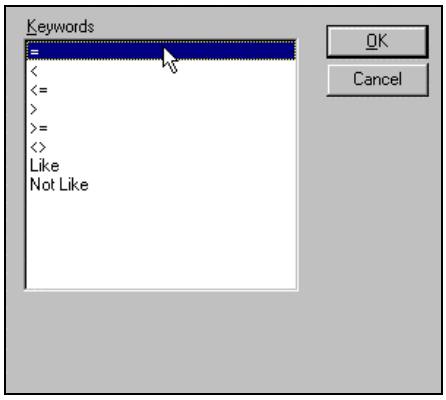


Abb. 7-28: Auswahl von Suchbedingungen bei Schlüsselabbildungen

In unserem Beispiel sollen alle Schlüssel aber wie vorgegeben die Gleichheit der Werte abfragen. Hat man alle Schlüsselbeziehungen definiert, so sind diese in der Ansicht durch ein Schlüssel-Icon gekennzeichnet.

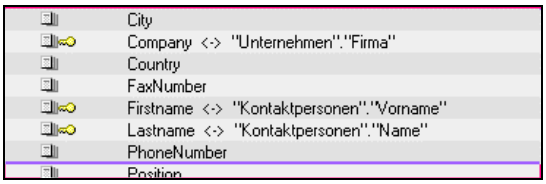


Abb. 7-29: Schlüsselbeziehungen in der Ansicht

7.3.4.2 Attribut mit Verschmelzungsformel

Will man einem Feld mehrere Attribute gegenüberstellen, so müssen die Attribute nach einer bestimmten Regel verkettet werden. Hierzu dienen die sogenannten Verschmelzungsformeln (*Siehe 7.3.1Das CDM (common data modell)*). Nur das erste Attribut kommt ohne eine Verschmelzungsformel aus. Jedem zusätzlichen Attribut muß eine entsprechende Formel zugewiesen werden. In unserem Beispiel wird das Feld „Street“ auf die Attribute „Strasse“ und „Hausnummer“ abgebildet. Die entsprechende Verschmelzungsformel für „Strasse“ und „Hausnummer“ mit dem Namen „Verketten mit Leerzeichen“ wird in der Formeldatenbank definiert. Alle Verschmelzungsformeln und deren inverse Formeln werden dort mit LotusScript programmiert. Bei der Synchronisation werden sie über den Befehl „Execute“ ausgeführt. Der Administrator kann bei Bedarf schnell eine Formel erstellen, die dann in der Formeldatenbank auch für spätere Schemadefinitionen bereitsteht. So entsteht im Laufe der Zeit ein Vorrat an diversen

Verschmelzungsformeln. Mit dieser Technik erreicht man die nötige Flexibilität, um auch auf unvorhersehbare Konstellationen reagieren zu können.

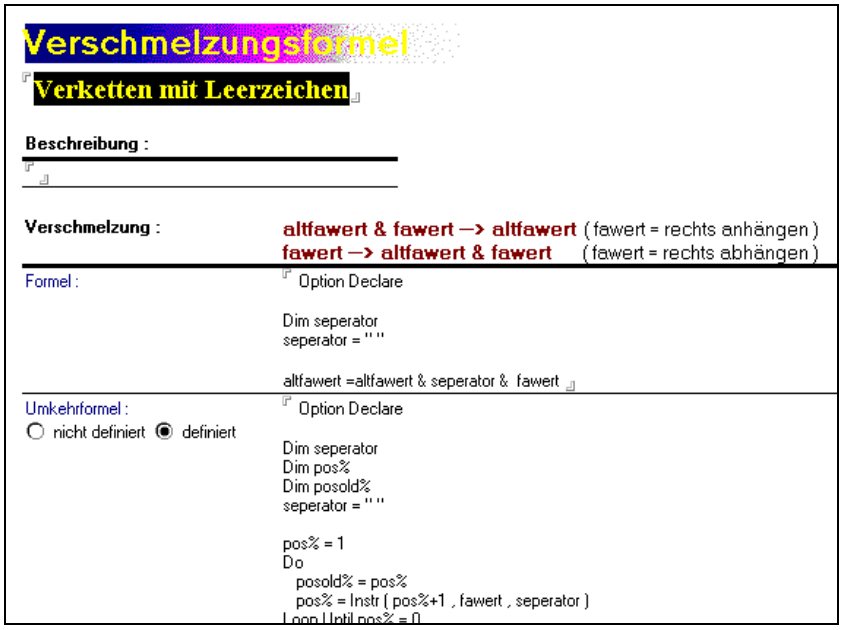


Abb. 7-30: Dokument für Verschmelzungsformel der Formeldatenbank

Bei der späteren Synchronisation beziehungsweise dem Datentransfer werden die beiden zu verschmelzenden Werte in die globalen Variablen „altfawert“ und „fawert“ gelegt. Die Verschmelzungsformel hat nun die Aufgabe, beide Werte zu verketten und das Resultat in die Variable „altfawert“ zurückzuschreiben. Bei der inversen Verschmelzungsformel muß umgekehrt ein Wert, der über die globale Variable „fawert“ bereitgestellt wird, in die entsprechenden Ausgangswerte aufgeteilt werden. Dabei muß der abgespaltene Wert in die Variable „fawert“ zurückgeschrieben werden. Der Restwert ist in die Variable „altfawert“ zu legen. Probleme der inversen Formel wurden bereits ausführlich diskutiert (*Siehe 4.2* ***Schemaübersetzung***). So kann es vorkommen, daß man keine Regel findet, die vom CDM-Attribut auf die einzelnen Werte schließen kann. Beim Fehlen einer solchen inversen Formel muß der Punkt „Umkehrformel nicht definiert“ markiert werden. Dadurch wird eine Schreiboperation auf die Datenbankkomponente, die dieser Formel zugeordnet ist, unterdrückt. Im unteren Teil des Dokuments muß man den Datentyp der Inputwerte bestimmen. Zusätzlich ist der Typ des nunvereinigten Wertes anzugeben.

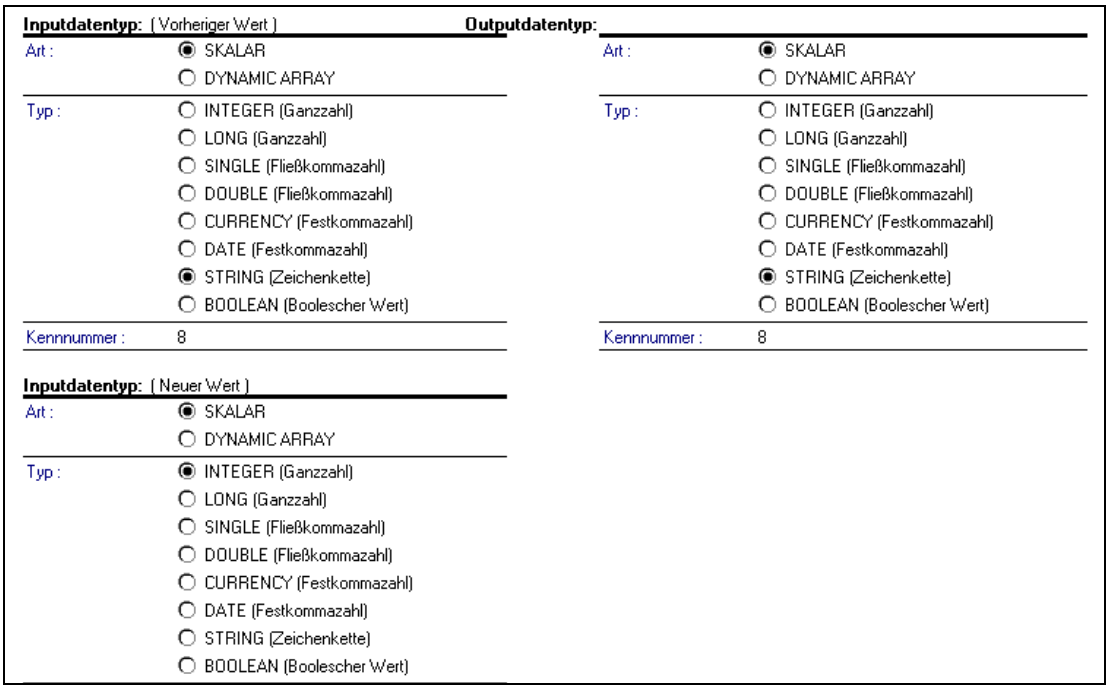


Abb. 7-31: Input- und Outputdatentyp der Verschmelzungsformel

Dieses ist notwendig, da bestimmte Operationen nur bei bestimmten Datentypen Sinn machen oder sogar erlaubt sind. So ist beispielsweise die Division von Zeichenketten nicht definiert. Über den Namen der Formel sollte man auf dessen Funktionsweise schließen können. Unterhalb des Namens befindet sich ein Feld, indem man weitere hilfreiche Informationen angeben kann.

Kommen wir zurück auf die Definition des CDM-Attributs in der Metadatenbank. Im entsprechenden Anwort-Dokument muß diesmal der Punkt „Abbildungsbeziehung“ gewählt werden. Der Aufbau des Abschnitts „Schemadefinition“ ist ähnlich dem bei der Definition einer Schlüsselbeziehung. Die Auswahl des ersten Attributs erfolgt hier per Mausklick auf das Plus-Zeichen neben der Bezeichnung „RDB-Attribute“. In der daraufhin erscheinenden Dialogbox wird mit dem ersten Button das Attribut gewählt. Nachdem wir das Attribut „Strasse“ gewählt haben, könnten wir dem Attribut mit Hilfe des zweiten Buttons eine Transformationsformel zuweisen. Wir akzeptieren die vorgegebene Formel „Do_Nothing“, die, wie der Name andeutet, keine Transformation durchführt.

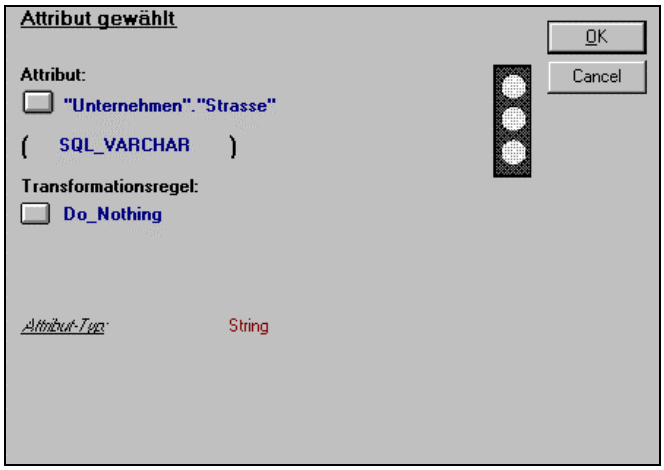


Abb. 7-32: Dialogbox: Spezifizierung des ersten Attributs

Mit Klick auf „OK“ verlassen wir die Dialogbox. Für jedes weitere Attribut müssen wir erneut die Dialogbox per Mausklick auf das Plus-Zeichen aufrufen. Wie beim ersten Mal wählen wir mit dem ersten Button das neu hinzuzufügende Attribut. Dieses Mal wählen wir „Hausnummer“. Nun ist ein dritter Button sichtbar, mit dem eine Verschmelzungsformel ausgewählt werden kann. Vorgegeben ist die sogenannte „Dummy_Formel“, die die einfachste Verschmelzung vorsieht. Eine „Dummy_Formel“ ist für jede Datentypkombination in der Formeldatenbank definiert. Da sie in unserem Fall nicht invers ist, zeigt die Ampel am rechten Rand der Dialogbox auf rot. Eine rote Ampel gibt zu erkennen, daß für die entsprechenden Attribute ein Schreibverbot gilt.

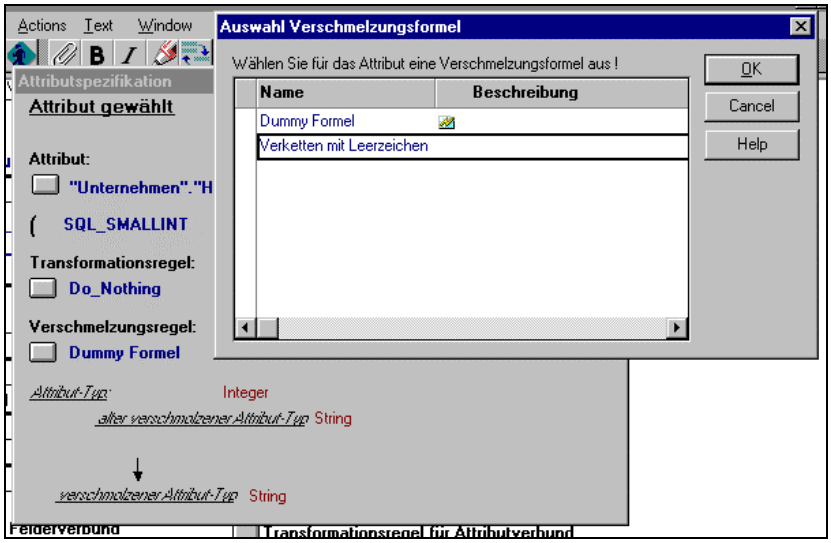


Abb. 7-33: Auswahl einer Verschmelzungsformel

Über den dritten Button wählen wir nun eine andere Verschmelzungsformel. Die Formeln werden mit ihren Namen und einer etwaigen Beschreibung in einer zusätzlichen Dialogbox angezeigt. Formeln, die nicht invers sind, werden durch ein Blitz-Icon gekennzeichnet. Angeboten werden uns alle Verschmelzungsformeln der Formeldatenbank, die als ersten Inputdatentyp „String“ und als zweiten Inputdatentyp „Integer“ spezifiziert haben, entsprechend den von uns gewählten Attributen. Wir wählten für unser Beispiel die Formel „Verkettung mit Leerzeichen“. Sie hängt ein Leerzeichen und die Hausnummer hinter den Straßennamen. Da diese Formel invers ist, erlischt die Ampel. Folglich wird ein späteres Schreiben auf die beteiligten Attribute wieder gewährt. Nach dem Schließen der Dialogbox sieht der Bereich für die Relationale Datenbank folgendermaßen aus:

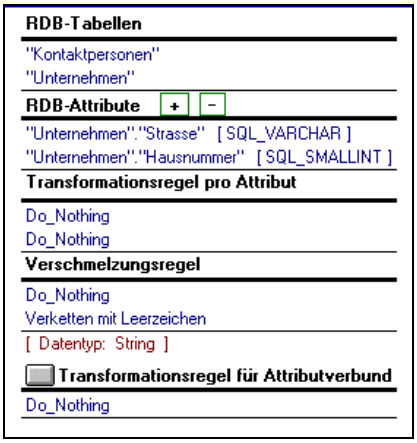


Abb. 7-34: Attribute und Formeln der Relationalen-Seite

Die beiden Attribute werden mit ihren Datentypen aufgelistet. Aus den Werten „Do_Nothing“ für die Transformationsregeln läßt sich schließen, daß beide Attribute nicht transformiert werden sollen. Unterhalb der Verschmelzungsformel steht der aktuelle Datentyp des CDM-Attributs. Dieser könnte mit Hilfe des letzten Buttons „Transformationsregel für Attributverbund“ erneut transformiert werden, um ihn beispielsweise an den Datentyp seines Notes-Pendant anzupassen. Diese ist für die im nächsten Abschnitt beschriebenen Definition der Konsistenzregel zwingend erforderlich (*Siehe 7.3.4.4 Definition einer Konsistenzregel*).

Danach kann im letzten Abschnitt „Schreibberechtigung bei Konsistenzwiederherstellung“ das hier definierte CDM-Attribut mit „Schreibberechtigung“, „Schreibschutz“ und „Schreibverbot“ belegt

werden. Was innerhalb der Schemaimportierung für die gesamte Datenbank definiert wurde, kann hier auf Attributebene verfeinert werden (*Siehe 7.3.2 Schemaimportierung*).

7.3.4.3 Attribut mit Transformationsformel

Mit Transformationsregeln kann der Wert eines Attributes oder Feldes umgeformt werden. So ist es möglich, den Datentyp entsprechend anzupassen oder auch den eigentlichen Wert umzuformen. In unserem Beispiel ist dieses bei der Telefonnummer notwendig. Das CDM-Attribut soll durch die Abbildung des Feldes „PhoneNumber“ auf das Attribut „Telefon“ realisiert werden. Die Werte beider Systeme unterscheiden sich jedoch dadurch, daß die Vorwahl der im Notes-Dokument gespeicherten Telefonnummer in Klammern gestellt wird. So sind die Rufnummern „(05251) 409780“ und „05251409780“ zwar logisch gleich, werden aber unterschiedlich dargestellt. Dieses ist ein klassisches Beispiel für unterschiedliche Ausdrucksweise im Rahmen von semantischer Heterogenität (*Siehe 4.3.2.2* ***Unterschiedliche Repräsentation***). Gesucht wird nun eine einheitliche Ausdrucksweise für das CDM-Attribut innerhalb des hier zu definierenden Föderativen Schemas. So könnte man beispielsweise entweder die Darstellung des Notes Feldes oder die des Attributs der Relationalen Datenbank übernehmen. Denkbar ist aber auch eine Darstellung, die weder der einen noch der anderen Datenbank entspricht. Dann müßte sowohl der Feldwert als auch der Wert des Attributs zur unabhängigen CDM-Darstellung transformiert werden. Die Transformation mit Hilfe der Transformationsregel ist Ausdruck der Schemaübersetzung innerhalb des Bottum-Up-Entwicklungsverfahrens (*Siehe 4.1* ***Der Bottom-Up Entwicklungsprozeß***). Eine Transformationsregel übersetzt somit ein Attribut des Lokalen Schemas ins CDM-Modell des Komponenten Schemas. Je geringer die Anzahl von Transformationen innerhalb der Definition des Föderativen Schemas, desto geringer ist die Anzahl der damit verbundenen möglichen Problemen. Grund hierfür sind nicht nur allein die inversen Transformationsregeln, die bereits ausführlich diskutiert wurden (*Siehe 4.2* ***Schemaübersetzung***). Aus diesem Grund bietet sich die Übernahme der Ausdrucksweise einer Datenbankkomponente an. In unserem Beispiel wollen wir die klammerlose Darstellung der Telefonnummer verwenden, so wie sie die Access Datenbank verwendet. Demzufolge ist der Feldwert des Dokuments mit Hilfe einer Transformationsformel zu übersetzen. Hierzu muß eine entsprechende Regel in der Formeldatenbank formuliert werden.

Nachfolgende Abbildung zeigt ein Ausschnitt des zu diesen Zwecken angelegten Transformationsregel-Dokuments aus der Formeldatenbank.

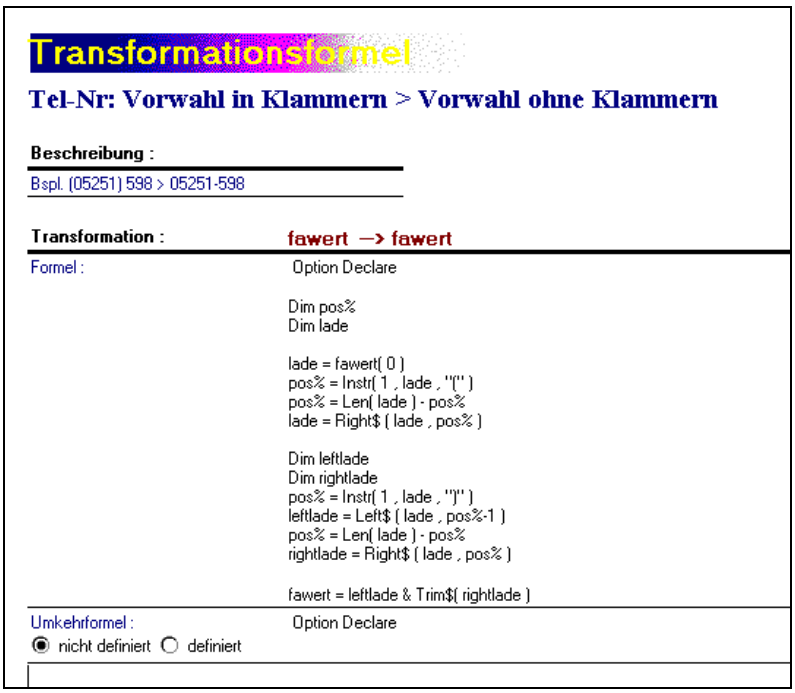


Abb. 7-35: Transformationsregel für Telefonnummer aus Formeldatenbank

Es ähnelt dem von Verschmelzungsformeln. Hier ist jedoch entsprechend nur ein Inputwert beteiligt, den der Programmierer aus der globalen Variable „fawert“ holt. Nach gewünschter Transformation wird der Wert in die gleiche Variable zurückgeschrieben. Inverse Transformationsregeln sind identisch zu programmieren.

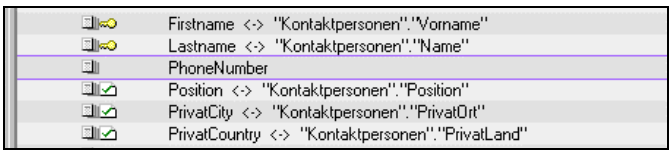


Abb. 7-36: Resonse-Dokument „PhoneNumber“ in der Ansicht

Um nun das CDM-Attribut „Telefonnummer“ zu definieren, laden wir das entsprechende Anwort-Dokument innerhalb der Ansicht. Im geöffneten Dokument „Definition Föderatives Schema“ wählen wir den Punkt „Abbildungsbeziehung“ und öffnen den Abschnitt „Schemadefinition“. Für die Relationale-Seite wählen wir das Attribut „Telefon“. Da dieses nicht transformiert werden soll, übernehmen wir bei der Wahl der Transformationsregel den Vorgabewert „Do_Nothing“. Der

Wert „Do_Nothing“ besagt, daß keine Transformation vorgenommen wird. Nach Spezifizierung des CDM-Attributs über die Relationale-Seite zeigt sich folgendes Bild.

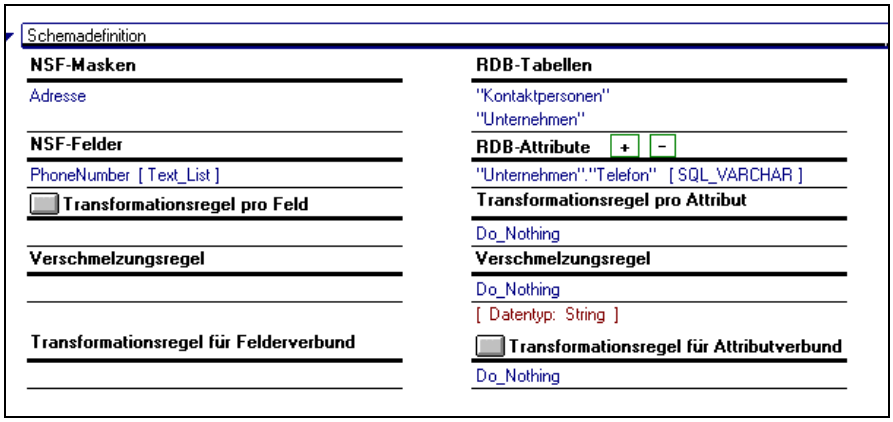


Abb. 7-37: Nach Spezifizierung der Relationalen-Seite

Für die Definition über die Notes-Seite muß dem vorgegebenen Feld „PhoneNumber“ die entsprechende Transformationsregel zugewiesen werden. Hierzu ist der Button neben „Transformationsregel pro Feld“ zu drücken. In der daraufhin hochgefahrenen Dialogbox werden alle zur Verfügung stehenden Transformationsformeln angeboten. Hieraus können wir die gewünschte Formel wählen.

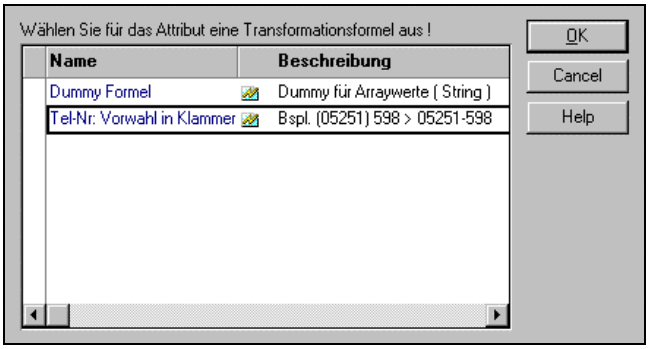


Abb. 7-38: Auswahl Transformationsregel

Das Blitz-Icon neben der gesuchten Formel „Tel-Nr: Vorwahl in Klammern > Vorwahl ohne Klammern“ signalisiert, daß keine inverse Rücktransformation definiert ist. Dies hat zur Folge, daß Schreiboperationen in das Feld „PhoneNumber“ bei der späteren Synchronisation verhindert werden. Davon kann man sich überzeugen, indem man sich die automatisch berechneten Schreibrechte im Abschnitt „Schreibberechtigun g bei Konsistenzwiederherstellung“ ansieht.

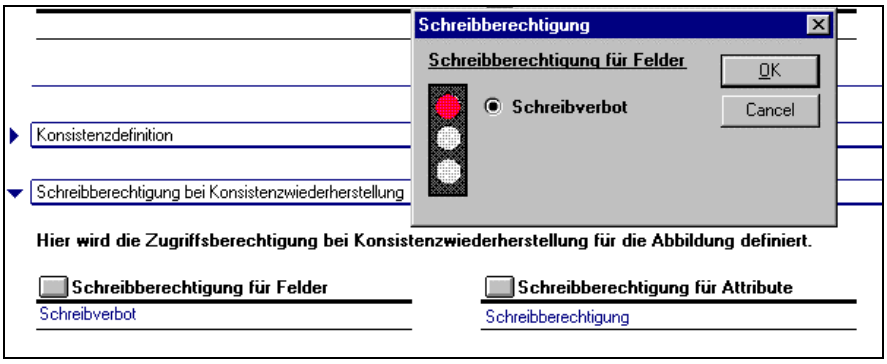


Abb. 7-39: Schreibberechtigung für CDM-Attribut „Telefonnummer“

Die Berechtigung für schreibende Operationen auf das entsprechende Feld wird nicht gegeben. Auch der Versuch, die Berechtigung manuell zu ändern, scheitert. In der dafür zur Verfügung stehenden Dialogbox wird nur die Auswahl „Schreibverbot“ angeboten. Die ansonsten möglichen Einstellungen „Schreibberechtigung“ und „Schreibschutz“ werden in diesem Fall automatisch unterdrückt. Nachfolgende Abbildung zeigt die fertig definierte Abbildung.

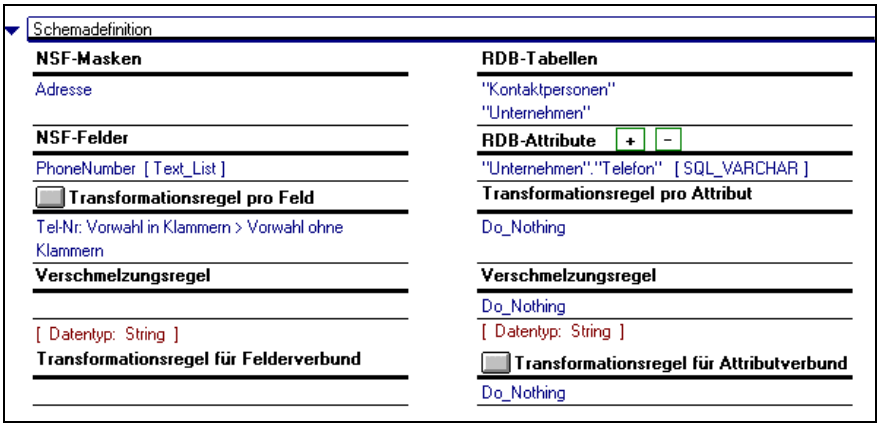


Abb. 7-40: Definition Föderatives Schema für CDM-Attribut „Telefonnummer“

Abschließend ist noch im mittleren Abschnitt eine Konsistenzregel zu bestimmen.

7.3.4.4 Definition einer Konsistenzregel

Jeder Abbildungsbeziehung muß eine Konsistenzregel zugewiesen werden. Mit der Konsistenzregel wird versucht, Überlegungen des Kapitels 4.4.3 **Modell zur Spezifizierung von Datenbankbeziehungen** in die Praxis umzusetzen. In ihr wird also berechnet, wann die beiden CDM-Attribut-Werte, die zum einen über die Notes-Seite und zum anderen über die Relationale-

Seite hergeleitet wurden, als Konsistent oder Inkonsistent gelten sollen. Wie alle Regeln wird auch sie in der Formeldatenbank mit LotusScript Code programmiert.

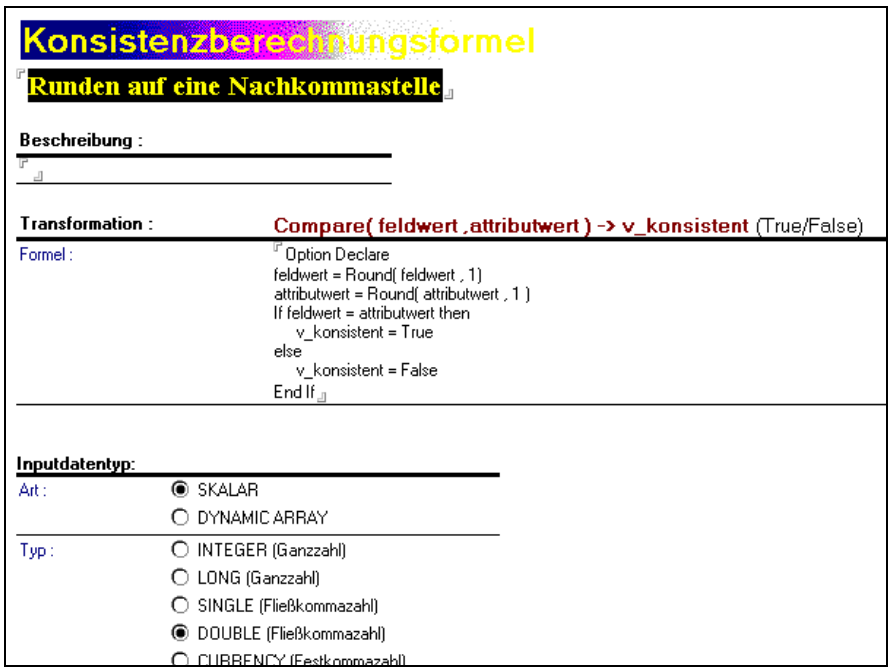


Abb. 7-41: Konsistenzberechnungsregel in Formeldatenbank

Die Formel der obigen Abbildung verlangt, daß zwei Fließkommazahlen mindestens auf eine Nachkommastelle genau übereinstimmen müssen. Die Formel erhält von der Anwendung die erforderlichen CDM-Attributwerte aus den beiden globalen Variablen „feldwert“ und „attributwert“. Der Code setzt die boolsche Variable „v_konsistent“ auf „True“, wenn beide Werte als Konsistent betrachtet werden können. Im anderen Fall wird die boolsche Variable auf „False“ gesetzt. Unterhalb des Codes wird der Datentyp für beide Inputwerte bestimmt.

Bei der Definition des Föderativen Schemas muß für jede Abbildungsbeziehung eine Konsistenzregel spezifiziert werden. Hierzu wechselt man in den zweiten Abschnitt „Konsistenzdefinition“. Voraussetzung ist, daß beide CDM-Attribute den gleichen Datentyp haben. Alle Konsistenzberechnungsformeln dieses gemeinsamen Datentyps werden per Mausklick auf den Button angeboten.

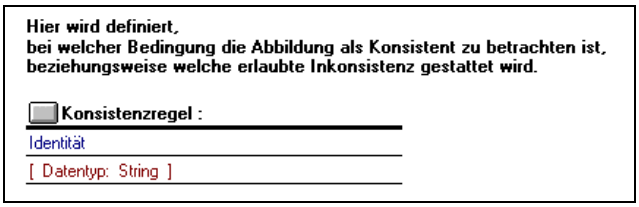


Abb. 7-42: Definition einer Konsistenzberechnungsregel

In unserem Beispiel wählen wir für alle Abbildungen die Formel „Identität“. Wie der Name bereits vermuten läßt, werden bei ihr nur identische Werte als Konsistent betrachtet.

7.3.4.5 Programmiertechnische Realisierung

Die Transformations-, Verschmelzungs- und Konsistenzberechnungsformeln sind in Dokumenten der Formeldatenbank gespeichert. In der Metadatenbank werden sie bei der Definition des Föderativen Schemas über Auswahlboxen zur Verfügung gestellt. Eine Auswahlbox wird hier mit Hilfe der @Function „@PickList“ erzeugt, in der man ein Dokument aus einer bestimmten Ansicht der Formeldatenbank wählen kann. Jede Ansicht der Formeldatenbank enthält nur Formeln einer bestimmten Art, die für die selben Inputdatentypen zuständig sind. So wird immer nur die Ansicht in der Dialogbox gezeigt, die den Datentypen der betroffenden Felder oder Attribute entspricht. Wählt man beispielsweise eine Transformationsformel für ein Feld vom Typ „Number“, so werden nur Formeln für die Transformation dieses Datentyps angeboten. Ebenso entsprechen die Inputdatentypen der aufgelisteten Verschmelzungsformeln den Datentypen des zu verschmelzenden Wertepaares. Bei der Auswahl einer Formel wird auch dessen zugewiesener Outputdatentyp ermittelt, da dieser wiederum Inputdatentyp bei der nächsten Transformation, Verschmelzung oder Konsistenzberechnung ist. Für die einheitliche Notation von Datentypen verwendet man die von LotusScript. Intern wird für jeden Datentyp eine Nummer geführt, die einen bestimmten Datentyp entspricht.

Return	Value type
0	EMPTY
1	NULL
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date/Time
8	String
9	OLE object or NOTHING
10	OLE error
11	Boolean
12	Variant list or array
13	IUNKNOWN (OLE value)
34	User-defined object
35	Product object
2048	List
8192	Fixed array
8704	Dynamic array

Abb. 7-43: LotusScript Datentypen

Bei der Schemaimportierung wird aufgrund der Angaben im Dokument „Datentypabbildung“ jeder ODBC Datentyp in den jeweiligen LotusScript Datentyp übersetzt. Dadurch wird eine einheitliche Notation auf allen Ebenen der Schemadefinition garantiert.

Neben der eigentlichen Definition der Abbildung kann eine Schreibberechtigung bei Konsistenzwiederherstellung für das hier definierte CDM-Attribut bestimmt werden. Pro Datenbankkomponente ist im letzten Abschnitt entweder der Wert „Schreibverbot“, „Schreibschutz“ oder „Schreibberechtigung“ zu wählen. „Schreibverbot“ verhindert eine Schreiboperation auf das Dokument beziehungsweise den Datensatz. Bei der Wahl von „Schreibberechtigung“ wird der Benutzer bei der späteren Synchronisation gefragt, ob er tatsächlich auf die Datenbankkomponente schreiben will. Jedoch ist es nicht möglich die Einschränkung der Zugriffsberechtigung, die im Dokument „Schemaimportierung“ für die gesamte Datenbank bestimmt wurde, hier auf Abbildungsebene zu entschärfen. Wurde beispielsweise die gesamte Dankbankkomponente mit einem „Schreibschutz“ belegt, so kann hier auf Abbildungsebene nicht mehr „Schreibberechtigung“ gegeben werden. Eine Verschärfung durch die Wahl von „Schreibverbot“ ist jedoch möglich. Neben dieser expliziten Zuweisung einer

Schreibberechtigung, kann das System einer Abbildung automatisch „Schreibverbot“ zuweisen. Dieses geschieht bei der Definition eines CDM-Attributs in folgenden Fällen:

? Wahl einer nicht inversen Transformations- oder Verschmelzungsformel.

? Wahl von Attributen mit nicht erlaubten Sonderzeichen.

? Wahl von Attributen, die ohne Tabellenbezeichner nicht eindeutig sind.

Auf die Problematik einer nicht inversen Transformationsformel wurde bereits ausreichend eingegangen (*Siehe 4.2* ***Schemaübersetzung***). Schreiboperationen auf Relationale Datenbanken verlangen bestimmte Konventionen bei den beteiligten Attributen, auf die näher im Kapitel 7.4.1 „Komprimiertes Föderatives Schema bereitstellen“ eingegangen wird.

7.4 Synchronistation

Die Synchronisation erfolgt im Dialog mit dem Benutzer. Aus einem offenen Dokument wird eine ODBC-Verbindung zur Relationalen Datenbank hergestellt. Nach Abgleich der Daten kann man eventuelle Inkonsistenz manuell beseitigen, indem man die korrekten Werte von der einen Datenbank in die jeweils andere Datenbank kopiert. Da der Prototyp mit dem im vorherigen Kapitel erstellten Föderativen Schema arbeitet, muß dieses dem Werkzeug zugänglich gemacht werden. Die Erstellung des Föderativen Schemas und dessen Bereitstellung gehören zum Verantwortungsbereich eines Administrators. Letzterer benötigt ausreichende Kenntnisse über Aufbau und Struktur der involvierten Notes Datenbank und Relationalen Datenbank. Die eigentliche Synchronisation wird vom Benutzer innerhalb der ihm vertrauten Office-Umgebung durchgeführt. Seine Kenntnisse über beide Datenbanken müssen nicht so detailliert sein.

7.4.1 Komprimiertes Föderatives Schema bereitstellen

Das Föderative Schema setzt sich aus den Anwort-Dokumenten für die Definition der CDM-Attribute der Metadatenbank zusammen. Um ein CDM-Attribut zeitweise aus dem Föderativen Schema zu nehmen, wählt man im entsprechenden Dokument den Punkt „bleibt unberücksichtigt“.

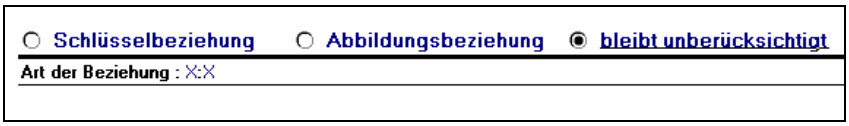


Abb. 7-44: Herausnahme von CDM-Attribut aus dem Föderativen Schema

Die Synchronisation basiert auf den Abbildungen des Schemas. Demzufolge muß das in vorausgegangenen Kapiteln entwickelte Föderative Schema dem Synchronisationswerkzeug zur Verfügung gestellt werden. Hierzu muß man erneut das Dokument Schemaimportierung aufrufen, das nunmehr die Wurzel der Hierarchie von Anwort-Dokumenten bildet. Oberhalb der Spezifizierung der Notes Datenbank und der Relationalen Datenbank kann dem Föderativen Schema ein beliebiger Name zugeordnet werden. Später wählt der Benutzer anhand dieses Namens das gewünschte Schema. Somit sollte der Name den Zweck des Schemas möglichst kurz und prägnant beschreiben.

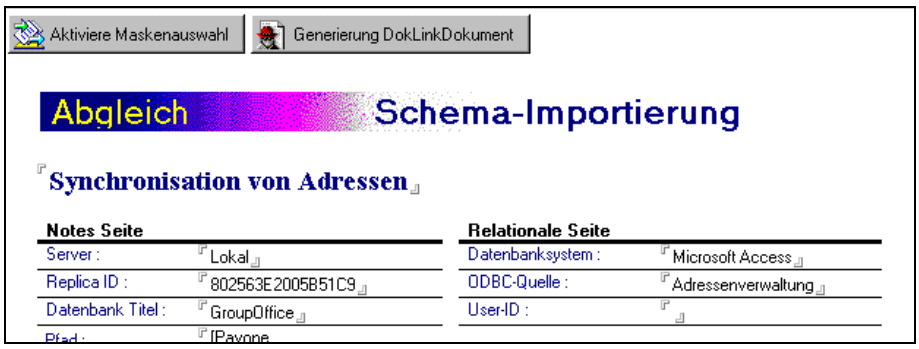


Abb. 7-45: Dokument für Schemaimportierung

So benennen wird das Föderative Schema unseres Beispiels mit „Synchronisation von Adressen“. Hierzu überschreiben wir den Vorgabewert entsprechend. Über den Button <Generierung DokLinkDokument> wird ein Dokument erzeugt, das alle notwendigen Informationen über das Schema konzentriert in einem Dokument sammelt. Ein Agent wird gestartet, der alle Anwort-Dokumente durchläuft und ausliest. Für jede gewählte Maske wird ein sogenanntes DocLink-Document angelegt, das über die Ansicht „Konfiguration / DocLinkDocument“ betrachtet werden kann. Am Anfang des Dokuments findet man Informationen über die beteiligten Datenbanken. Neben der Angabe über die Notes Datenbank und der betroffenen Maske, ist die ODBC-Quelle der Relationalen Datenbank verzeichnet. Das Föderative Schema ist also nur für eine bestimmtes Paar von Notes Datenbank und ODBC-Datenbank einsetzbar. Deshalb wird später nur Benutzern,

die innerhalb einer Adressenmaske von GroupOffice die Synchronisation starten, unser Föderatives Schema angeboten, mit dem sie dann die Synchronisation durchführen können.

DocLink Abgleich

Synchronisation von Adressen

Notes Seite		Relationale Seite	
Server :	Lokal	Datenbanksystem :	Microsoft Access
Replica ID:	802563E2005B51C9	ODBC-Quelle :	Adressenverwaltung
Datenbank Titel :	GroupOffice	User-ID :	
Pfad :	[Pavone Office]\141296\dev_JW.nsf		

Maske	
Maskenart :	Maske
Aliasname :	Address

Tabellenspezifizierung	
Select - Teile :	SELECT Position
	SELECT Strasse , Hausnummer
	SELECT Postleitzahl
	SELECT Ort
	SELECT Land
From :	FROM Kontaktpersonen , Unternehmen
Where -Teil :	AND "Kontaktpersonen" "ID" = "Unternehmen" "ID" AND "Unternehmen" "Geschaeftsbezie

Abb. 7-46: DocLink-Document des Föderativen Schemas

Im DocLink-Document sind dafür alle Abbildungen mit ihren Transformations-, Verschmelzungs- und Konsistenzberechnungsformeln aufgeführt. Der LotusScript Code der Formeln wurde aus den entsprechenden Dokumenten der Formeldatenbank kopiert, so daß die eigentliche Synchronisation ohne die Formeldatenbank auskommt. Soweit möglich werden die später aufzurufenden SQL-Befehle bereits vorformuliert, damit diese nicht bei der eigentlichen Synchronisation berechnet werden müssen.

7.4.2 Synchronisation im Dialog

Erst jetzt tritt der eigentliche Benutzer in Aktion. Aus einer Ansicht in GroupOffice lädt er das Dokument, das er mit der Access Datenbank abgleichen will. In unserem Beispiel ist es das Adressendokument des Lieferanten „Jens Winkelmann“.

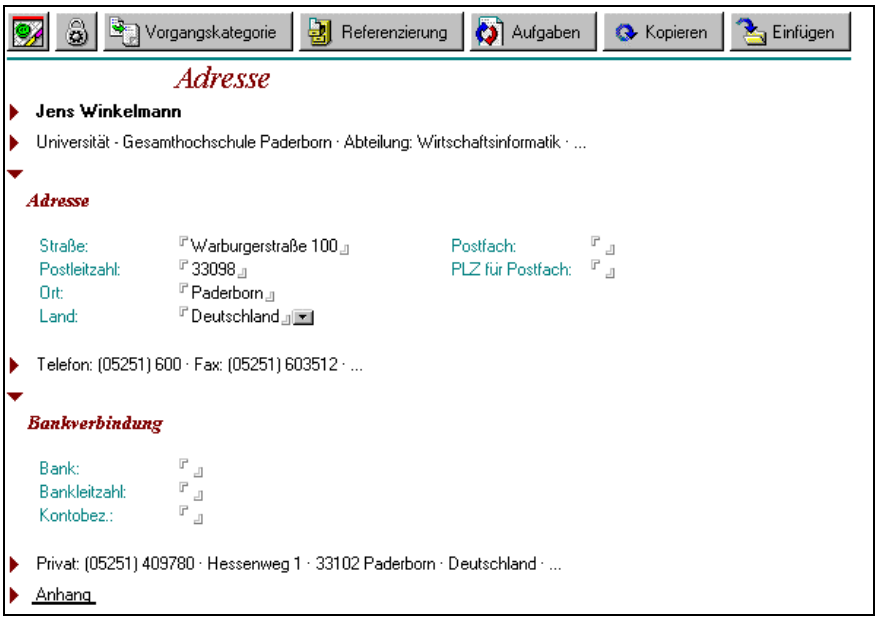


Abb. 7-47: Geöffnetes Adressendokument

Wie man der obigen Abbildung entnehmen kann, fehlen die Angaben über die Bankverbi ndung. Diese wollen wir aus der entsprechenden Access Datenbank gewinnen, die parallel Kunden- und Lieferantenadressen verwaltet. Weiterhin wurde uns zugetragen, daß die Access-Datenbank vermutlich die falsche Postleitzahl für diesen Lieferanten gespeichert hat. Auch dieses soll überprüft werden. Im Menü „Actions“ steht der Agent „Synchronisation mit RDB“ zur Verfügung. Nach dessen Aufruf erscheint folgende Dialogbox.

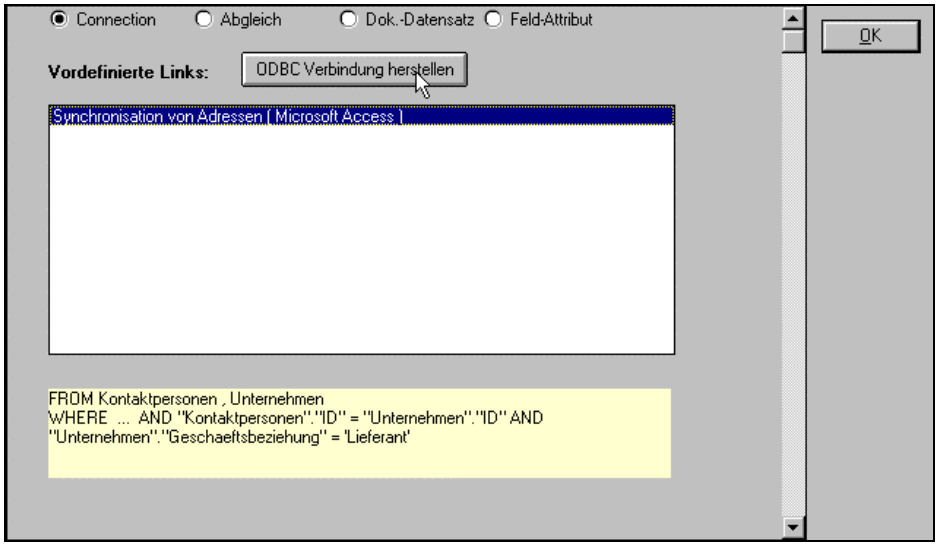


Abb. 7-48: Dialogbox: Auswahl Föderatives Schema

Hier werden alle DocLink-Documents angeboten, die entsprechend ein Föderatives Schema für Adressendokumente der GroupOffice Datenbank beschreiben. Wir wählen das Schema unseres Beispiels über dessen Namen „Synchronisation von Adressen“ aus. Das untere Fenster zeigt entsprechende Informationen über die Relationale-Seite. Bei der Markierung eines Schemanamens werden dessen Tabellen und deren Join- und Selektionsbedingungen angezeigt. Mit dem Button <ODBC Verbindung herstellen> wird eine Verbindung zur Access Datenbank hergestellt. Sollte auf dem Rechner die notwendige ODBC-Quelle nicht installiert sein, so wird dieses dem Benutzer mitgeteilt. Mit Hilfe des ODBC-Managers muß der Benutzer die Quelle anlegen.

Abb. 7-49: Fehlermeldung für fehlende ODBC-Quelle

Am oberen Rand der Dialogbox befindet sich ein Reiter mit dem zwischen verschiedenen Ansichten der Dialogbox gewechselt werden kann. Hat man die Verbindung aufgebaut, so wechselt man von der Ansicht „Connection“ zu Ansicht „Abgleich“. Dort wählt man den Button <Abgleich>. Daraufhin wird eine Synchronisation für jedes CDM-Attribut durchgeführt.

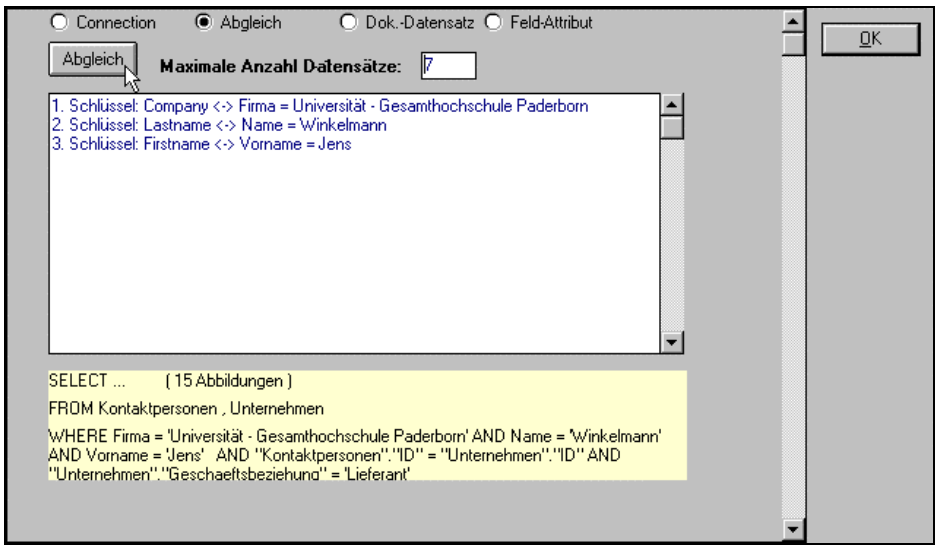


Abb. 7-50: Ansicht nach Abgleich aller Abbildungen

Im unteren Fenster wird nacheinander der entsprechende SQL-Befehl für den Zugriff auf die Relationale Datanbank eingeblendet. Wurden alle Abbildungen abgearbeitet, so erstarrt das

Fenster mit der Angabe über die Anzahl der Abbildungen sowie des From- und Where-Teils des ODBC-Zugriffs. Im Hauptfenster werden alle verwendeten Schlüsselabbildungen gezeigt. Desweiteren kann man hier ablesen, ob ein Schlüsselwert leer oder nicht vorhanden war und wie darauf reagiert wurde (*Siehe 7.3.4.1 Definition eines Schlüsselattributs*). Fehlermeldungen werden hier ebenfalls angezeigt. Wechselt man danach in die Ansicht „Dok.-Datensatz“, so kann im dortigen Fenster die Anzahl der gefundenen Datensätze abgelesen werden. Die Anzahl der maximalen Treffer konnte in der vorigen Ansicht begrenzt werden. In unserem Beispiel haben wir zwei Treffer gelandet. Folglich besitzt die Access-Datenbank zwei Datensätze, deren Attribute „Firma“ „Name“ und „Vorname“ identisch mit den Feldwerten des geöffneten Dokuments sind. Das heißt die Access-Adressendatenbank hat zwei Personen mit dem Namen „Jens Winkelmann“ gespeichert, die beide an der Universität Paderborn eingeschrieben oder tätig sind. Um nun aus beiden den von uns gesuchten zu ermitteln, drücken wir den Button <Konsistenz aller Datensätze>. Hierdurch werden CDM-Attribute, die über die Notes-Seite und diejenigen die über die Relationale-Seite berechnet worden, verglichen. Mit Hilfe der Konsistenzberechnungsformel werden die dazugehörigen Abbildungen als konsistent oder inkonsistent eingestuft.

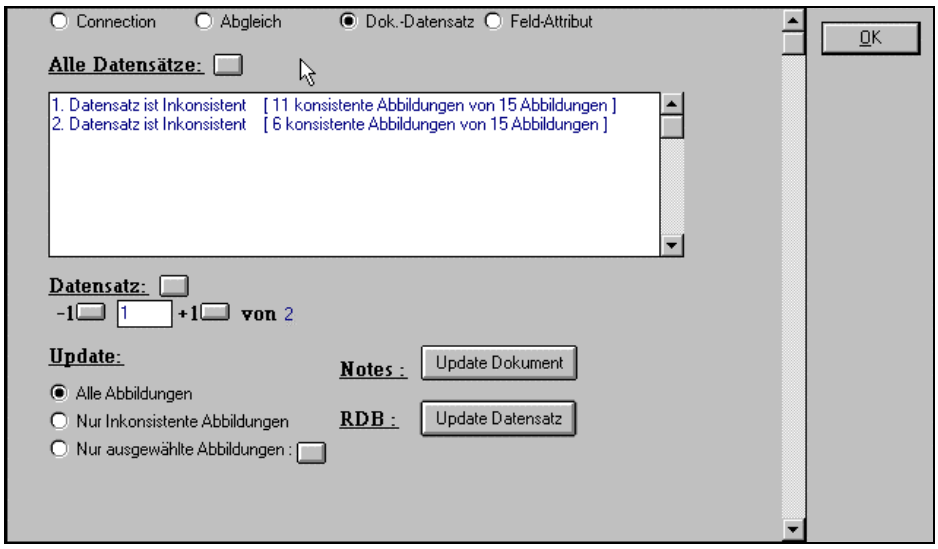


Abb. 7-51: Bewertungsquote aller Datensätze

Im Fenster wird eine Quote für jeden Datensatz angegeben, die besagt wieviele konsistente CDM-Attribute der jeweilige Datensatz im Vergleich zu allen CDM-Attributen hat. Wir erkennen, daß der erste Datensatz 11 konsistente Abbildungen von 15 möglichen Abbildungen hat, der zweite Datensatz aber nur eine Quote von 5 zu 15 aufweist. Hieraus schließen wir, daß der erste

Datensatz der von uns gesuchte ist, da er die bessere Quote besitzt. Eine bessere Quote bedeutet im diesen Zusammenhang, daß dieser Datensatz dem geöffneten Dokument mehr ähnelt als alle anderen gefundenen Datensätze. Über die Buttons, die direkt unterhalb des Fensters liegen, können wir die einzelnen Abbildungen eines Datensatzes genauer betrachten.

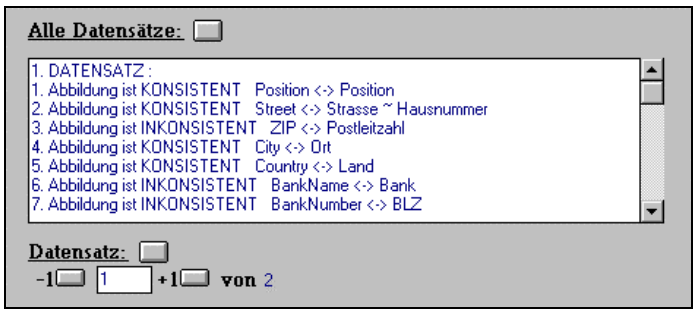


Abb. 7-52: Datensatz aufgeschlüsselt nach Abbildungen

Scrollen wir durch das Fenster, so werden wir bestätigt, daß bei der Abbildung der Postleitzahl eine Inkonsistenz vorliegt. Daneben sind auch alle Abbildungen bezüglich der Bankverbindung inkonsistent. Dieses ist verständlich, da diese Angaben im Dokument fehlen. Bevor wir die Inkonsistenz der Postleitzahl näher untersuchen, wollen wir Bankbezeichnung, Bankleitzahl und Kontonummer des Kunden „Jens Winkelmann“ aus der Access-Datenbank ins Dokument kopieren. Hierzu wenden wir uns dem unteren Teil der Dialogbox zu. Unterhalb der Überschrift „Update“ können wir wählen, ob wir alle Abbildungen des gewählten Datensatzes, nur Abbildungen, die inkonsistent sind, oder bestimmte ausgewählte Abbildungen kopieren wollen. Da uns nur die drei oben genannten interessieren, wählen wir den dritten Punkt „Nur ausgewählte Abbildungen“. Mit dem daneben liegenden Button wählen wir die gewünschten Abbildungen aus.

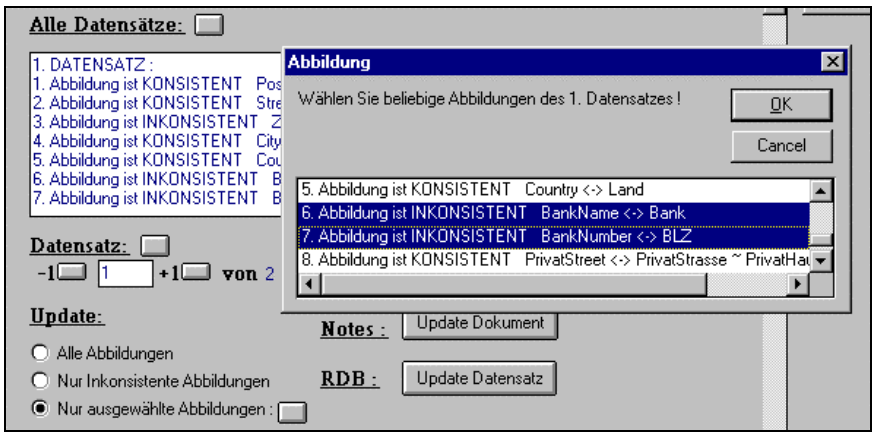


Abb. 7-53: Auswahl von CDM-Attribute für Importierung

Per Mausklick auf <Update Dokument> werden die ausgesuchten Attributwerte des gewählten Datensatzes von der Access-Datenbank geholt und in die entsprechenden Felder des geöffneten Dokuments geschrieben. Wäre eine Abbildung mit einem Schreibschutz belegt, so würde uns das System darauf hinweisen, daß das entsprechende Attribut nicht kopiert werden darf. Im Falle eines Schreibschutzes würden wir gefragt, ob wir tatsächlich den entspechenden schreibgeschützen Wert überschreiben wollen. Die Ansicht der Dialogbox wechselt a utomatisch auf „Abgleich“, in der wir nun erneut die Abgleich-Routine starten können. Nach Verlassen der Dialogbox mit <OK>, würden wir sehen, daß der Abschnitt „Bankverbindung“ mit den korrekten Werten der Relationalen Datenbank gefüllt ist.

Abb. 7-54: Importierte Daten im Adressendokument

Zuvor wollen uns jedoch Klarheit über die Inkonsistenz beim CDM-Attribut „Postleitzahl“ verschaffen. Hierzu wechseln wir nach einer erneuten Synchronisation in die Ansicht „Feld-Attribut“. Die Ansicht „Dok-Datensatz“ gab uns Informationen auf Datensatzebene. Hier wird ein einziges CDM-Attribut aufgeschlüsselt.

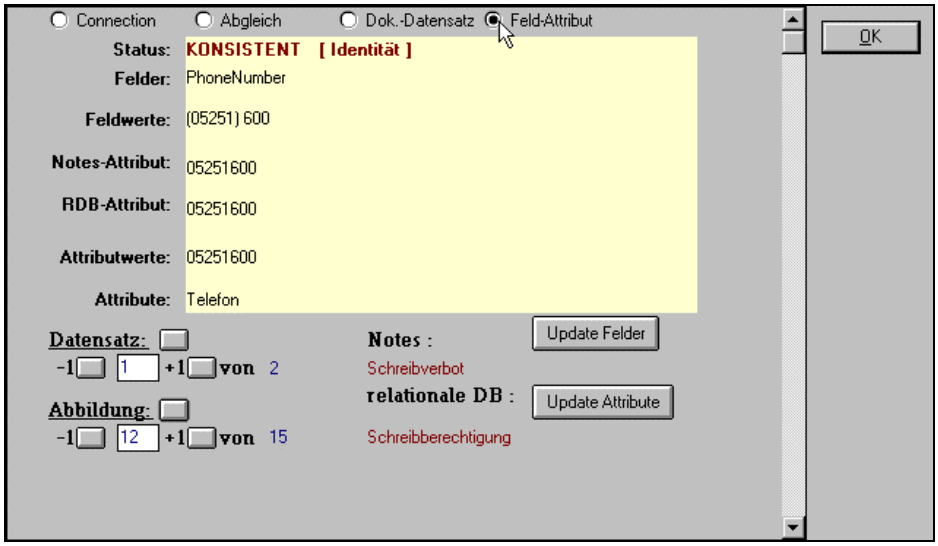


Abb. 7-55: Aufschlüsselung einer Abbildung

Aus der ersten Zeile des Fensters entnehmen wir, ob die Abbildung konsistent oder inkonsistent ist. In Klammern ist die hierfür verantwortliche Konsistenzberechnungsformel angegeben. In der nächsten Zeile steht der Name der Felder beziehungsweise des Feldes. Darunter sind deren Werte aufgelistet, so wie sie im Dokument zu finden sind. Aus diesen Werten wird mit Hilfe der Transformations- und Verschmelzungsformeln das sogenannte CDM-Attribut berechnet. In der vierten Zeile finden wir den Wert des CDM-Attributs, der über die Feldwerte des Notes-Dokuments berechnet wurde. Die letzten drei Zeilen sind für die Relationale-Seite reserviert. In umgekehrter Reihenfolge wird als erstes das über die Attribute berechnete CDM-Attribut angegeben. Danach folgen die Attributwerte und als letztes die Namen der Attrbute. Obige Abbildung zeigt das mit Hilfe einer Transformationsformel die Klammern der Telefonvorwahl entfernt wurden. Somit haben beide CDM-Attribute die gleiche Darstellungsweise. Die nun identischen Werte bei gleicher Darstellung sind verantwortlich für die Konsistenz der Abbildung.

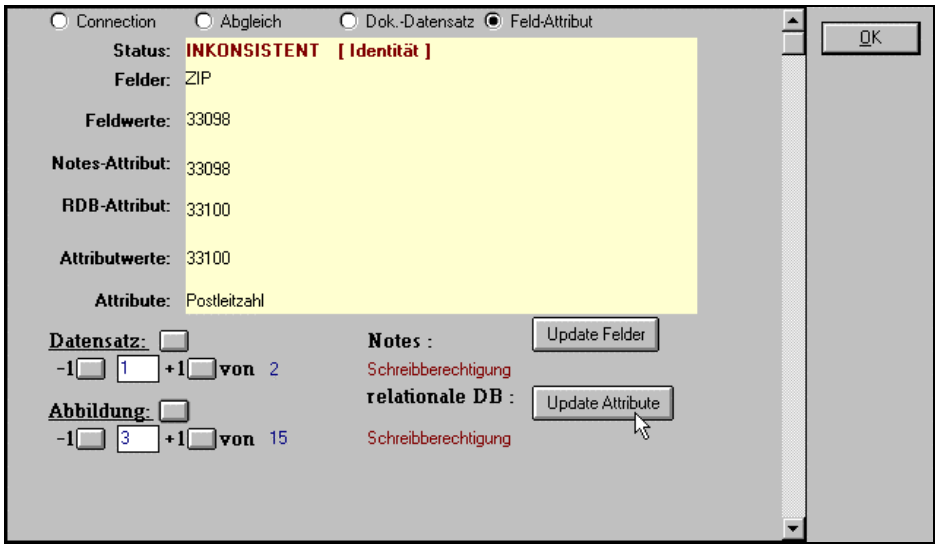


Abb. 7-56: Abbildung der Postleitzahl

Im linken unteren Bereich der Dialogbox findet man Buttons, mit denen zwischen Datensätzen und deren Abbildungen gewechselt werden kann. Die gesuchte Postleitzahl finden wir unter der dritten Abbildung des ersten Datensatzes. Wir erkennen, daß in der Relationalen Datenbank ein nicht korrekter Wert gespeichert ist. Im rechten unteren Bereich der Box finden wir das Werkzeug um die Konsistenz wiederherzustellen. Da wir für eine Schreiboperation auf der Relationalen Datenbank Berechtigung haben, klicken wir auf den Button <Update Attribut>. Dadurch wird die Postleitzahl des Adressendokuments in die Access-Datenbank geschrieben. Ein nochmaliges

Ausführen der Abgleichroutine würde uns zeigen, daß Datensatz und Dokument nun über alle Abbildungen konsistent sind. Über <OK> verlassen wir die Dialogbox.

7.4.3 Programmiertechnische Realisierung

Fast der gesamte Code für die Synchronisation ist in der Dialogmaske enthalten. Die Aufgabe des Agents besteht hauptsächlich im Aufruf der Dialogbox. Ebensogut könnte der Dialog auch mit Hilfe eines in der Maske verankerten Actions gestartet werden. Direkt nach Aufruf der Dialogbox werden alle entsprechenden Föderativen Schemen angeboten. Hierzu wird auf die sogenannten DocLink-Dokumente in der Metadatenbank zugegriffen. GroupOffice arbeitet mit einer Repository-Datenbank zusammen, aus der es unter anderem allgemeine Schlüsseldaten sowie Briefköpfe ausliest. Hier wäre für unser Beispiel der bessere Ort für die Speicherung der DocLink-Dokumente gewesen. Der Code müßte entsprechend angepaßt werden. Das Kopieren eines Dokuments von der Metadatenbank in das Repository würde dann das entsprechende Föderative Schema bereitstellen. Ein einfaches Löschen des Dokuments würde das Schema wieder zurücknehmen. Der Zugriff auf die Relationale Datenbank erfolgt wie bei der Schemaimportierung mit Hilfe der LotusScript Data Object (LS:DO) (*Siehe 6.3.1.2.2* ***LotusScriptEXtension (LSX)***). Ein Objekt der Klasse „ODBCConnection“ wird nach Aufbau der Verbindung in einer globalen Variable zur Verfügung gestellt. So bleibt die ODBC-Verbindung während des gesamten Dialogs aufgebaut. Alle weiteren LotusScript Actions der Dialogbox können auf die globale Variable zugreifen. Die Abgleichroutine legt ein neues temporäres Dokument an. In diesem Dokument speichert sie die ausgelesenen Feld- und Attributwerte zwischen. Bei der Abbildungsaufschlüsselung wird auf diese Daten zurückgegriffen. Per @Funktionen in berechneten Feldern werden die gewünschten Feldwerte und Attributwerte blitzschnell gegenübergestellt. Das temporär angelegte Dokument wird nach Beendigung des Dialogs automatisch gelöscht. Leider kann mit Hilfe von ODBC über LS:DO kein Datensatz gesperrt werden. Falls andere Applikationen diese während des Dialogs modifizieren, könnten Probleme auftreten. Ebenso muß kritisch festgestellt werden, daß die aktuelle Version der LotusScript Data Objects in Zusammenarbeit mit dem Access ODBC-Treiber einige Mängel aufweist. Leseoperationen per ODBC bereiten keine Schwierigkeiten. Setzt man sämtliche Tabellen- und Attributnamen, die Sonderzeichen, Umlaute oder Leerzeichen besitzen, in Anführungszeichen, so können alle

Datentypen gelesen werden. Schreiboperationen erlauben keine Anführungszeichen, so daß hier Namen mit diesen Merkmalen nicht erlaubt sind. So verfolgt der Prototyp folgende Strategie. Im Dokument Datentypabbildung kann pro Datenbank bestimmt werden, ob Attribut- und Tabellennamen bei der Schemaimportierung in Anführungszeichen gesetzt werden.

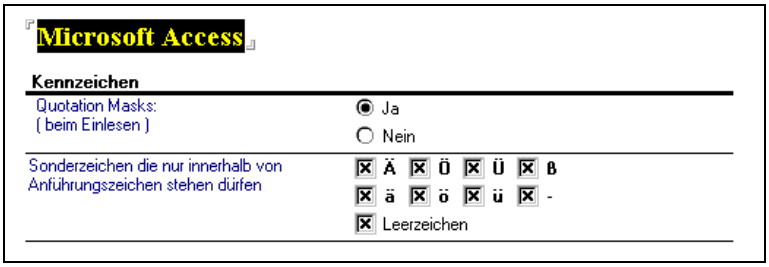


Abb. 7-57: Einstellung für Namensbehandlung

Diese Wahlmöglichkeit ist notwendig, da andere ODBC-Treiber, wie beispielsweise der für Lotus Notes (*Siehe 6.3.1.4* ***Lotus Notes ODBC-Treiber für Windows***), grundsätzlich keine Anführungszeichen erlauben. Bei der Erstellung des DocLink-Dokuments werden die Anführungszeichen wieder entfernt, soweit nicht einer der gewählten Sonderzeichen im Namen enthalten ist. Ebenso verzichtet man auf den vorangestellten Tabellennamen eines Attributs, soweit dieser eindeutig ist. Behält ein Attribut seine Anführungszeichen oder seine Tabellenerweiterung, so erhält die entsprechende Abbildung automatisch Schreibverbot auf der Relationalen Datenbank. Trotzdem gelingt nicht in allen Situationen und bei allen Datentypen eine Schreiboperation auf die Access-Datenbank. In solchen Fällen erscheint eine entsprechende Fehlermeldung.

Abb. 7-58: Fehlermeldung bei fehlgeschlagener Schreiboperation

Der bei der Schemaimportierung definierte Primärschlüsses wird jeder ODBC-Resultmenge hinzugefügt. Hierdurch wird jeder Datensatz in der Menge eindeutig identifiziert. Fehlt der Schlüssel, so wird eine Schreiboperation, die für ein bestimmten Datensatz gedacht ist, fälschlicherweise auf alle Datensätze mit gleichen Werten ausgeführt. Diese Lösung verlangt jedoch, daß man den Primärschlüssel selbst in die Datenbank zurückschreiben kann. So darf beispielsweise der Name des Schlüssels kein Sonderzeichen enthalten. Ob der Access-ODBC-

Treiber oder die LotusScriptEXtension verantwortlich für solche und andere Ungereimtheiten sind, muß ausgiebig getestet werden.

7.5 Resümee und Ausblick

Die Entwicklung eines Prototyps im Rahmen einer wissenschaftlichen Arbeit sollte nicht nur zur anschaulich Demonstration der Theorie dienen. Ebenso sollte mit ihm der praktische Nutzen ausgelotet werden, um dessen Chancen zur Weiterentwicklung zu einem marktfähigen Produkt beurteilen zu können. Deshalb sollen abschließend noch einige Gedanken über die Einordnung des Prototyps im Vergleich zu anderen Lösungen sowie Aufgaben und Probleme bei dessen Weiterentwicklung gesammelt werden.

7.5.1 Architektur

Der Prototyp ist ein fest in Notes eingebettetes Werkzeug für den Datenaustausch von Notes Daten mit Daten aus ODBC-fähigen Datenbanken. Es gibt andere mächtigere Ansätze, die man zurecht als Föderative Datenbanken bezeichnen kann. Der Prototyp ist nur ein Datenaustausch-Werkzeug, das ein spezielles eingeschränktes Föderatives Schema nutzt. Die Entwicklung einer Föderativen Datenbanken würde den Rahmen einer Diplomarbeit bei weitem sprengen. Aber nicht nur der Entwicklungsaufwand ist ein entscheidendes Kriterium für die Wahl einer Architektur. Auch Kosten für Wartung und Verwaltung einer Anbindung von Notes an eine Relationale Datenbank müssen berücksichtigt werden. Die Administration des Prototyps beschränkt sich auf die Erstellung der Föderativen Schemen innerhalb der Metadatenbank sowie deren Bereitstellung. Dieses kann vom Notes Administrator in Kooperation mit dem Administrator der Relationalen Datenbank erledigt werden. Der Aufwand zur Administration einer echten Föderativen Datenbank würde ein vielfaches der des Prototyps betragen. Dieses könnte viele potentielle Kunden abschrecken. Notes Pump kann man zwar auch nicht als echte Föderative Datenbank bezeichnen, jedoch weist sie mehr Gemeinsamkeiten mit einer solchen auf (*Siehe 6.3.2* ***Lotus NotesPump***). So erlaubt Notes Pump beispielsweise auch den Datentransfer und die Synchronisation zwischen reinen Relationalen Datenbanken. Der Code befindet sich in einem Programm, das außerhalb von Notes auf einem Server läuft. Nur die Administration erfolgt mit Hilfe von Notes Datenbanken. Für bescheidenere Ansätze, wie sie der Prototyp verfolgt, sollten die Entwicklungswerkzeuge, die

Notes bietet, ausreichen. LotusScript Agenten in Notes Datenbanken vermeiden die Kompatibilitäts- und Installationsprobleme externer Lösungen. Die notwendigen DLLs bei der Verwendung von LotusScript Data Object (LS:DO) werden bei der Installation von Notes 4 automatisch installiert. Somit läuft der Prototyp unabhängig vom Betriebssystem und Netzwerk. Jedoch stehen für Unix-Derivate und Macintosh Computer LotusScript Data Object erst ab der Version 4.5 zur Verfügung. Eine in Notes eingebettete Lösung kann außerdem dessen ausgereftes und vielschichtiges Sicherheitsmanagement nutzen. Dieses Bild trübt der auf 64 Kbyte begrenzte Speicher für Code und Daten der LotusScript Anwendungen. Bei der Entwicklung des Prototyps bin ich selbst einmal unliebsam an diese Grenze gestoßen. Auch sind für diese Zwecke die beschränkten GUI-Möglichkeiten von Notes eher hinderlich. Für die Archivierung von Notes-Daten unerläßlich ist die Einbeziehung von Rich Text Feldern. Die LotusScript Data Objects können derzeit aber nur strukturierte Daten verarbeiten. Man könnte für den Datentransfer von Rich Text eine eigene LotusScriptEXtension (LSX) schreiben, die mittels Notes-C-API und ODBC-C-API Abhilfe schafft. Damit wären die Vorteile der Kompatibilität und der einfachen Installation aber wieder verloren. Weiterhin sollte man die angesprochenen Probleme der LotusScript Data Objects bei Schreiboperationen verfolgen (*Siehe 7.4.3 Programmiertechnische Realisierung*). Hierfür sollte der Prototyp auch mit anderen ODBC-Treibern getestet werden. Der Einsatz mit einem Two-Tier Treiber (*Siehe 5.2.5.2* ***Treiberrangstufen***), der direkt mit der Datenbank kommuniziert, wäre in diesem Zusammenhang eventuell aufschlußreich. Jedoch werden die DLLs der LotusScript Data Objects auch weiterhin von Lotus gepflegt, so daß man auf Verbesserungen hoffen darf. Über die Lotus Internet Homepage kann man sich die derzeit aktuelle Version 4.11 herunterladen.

7.5.2 Werkzeuge für Administration

Neben der eigentlichen Engine für Synchronisation und Datentransfer ist ein komfortables Werkzeug für die Schemadefinition unabdingbar. Hierfür steht beim Prototyp die sogenannte Metadatenbank zur Verfügung, in der nacheinander die Schemaimportierung, die Schemaergänzung und die eigentliche Definition des Föderativen Schemas durchgeführt wird (*Siehe 7.3 Föderatives Schema in Metadatenbank erstellen*). Die Importierung des Masken-Designs und die sich daran anschließende Schemaergänzung ist nur deshalb notwendig, da Lotus

Notes keinen Datenbankkatalog oder etwas vergleichbares besitzt, aus dem der schematische Aufbau der Dokumente ermittelt werden kann. Die Herleitung des Schemas über die Maske, so wie es der Prototyp vorschlägt, ist nur eine mögliche Methode. Gerade wenn eine Maske intensiv den Einsatz von errechneten Teilmasken nutzt, kann der Nutzen dieser Methode in Zweifel gezogen werden. Alternativ könnte man die Felder aller Dokumente durchsuchen, um aus den gewonnenen Informationen das Datenbankschema der Datenbank zu konstruieren. Da dieses bei entsprechend großen Datenbanken mehrere Stunden in Anspruch nehmen kann, könnte man sich auf einige gezielt ausgewählte Dokumente beschränken, die sinnvollerweise alle möglichen Felder der jeweiligen Dokumentenklasse besitzen. Andererseits wäre es ebenso denkbar, daß man auf ein automatisches, beziehungsweise halbautomatisches Ermitteln des Schemas ganz verzichtet. Im letzten Fall müßte die Schemabeschreibung manuell per Eingabe erfolgen. Aber nicht nur bei der manuellen Schemabeschreibung würde ein Abspeichern und regelmäßiges Pflegen des Schemas im Sinne eines künstlich erzeugten Datenbankkatalogs Sinn machen.

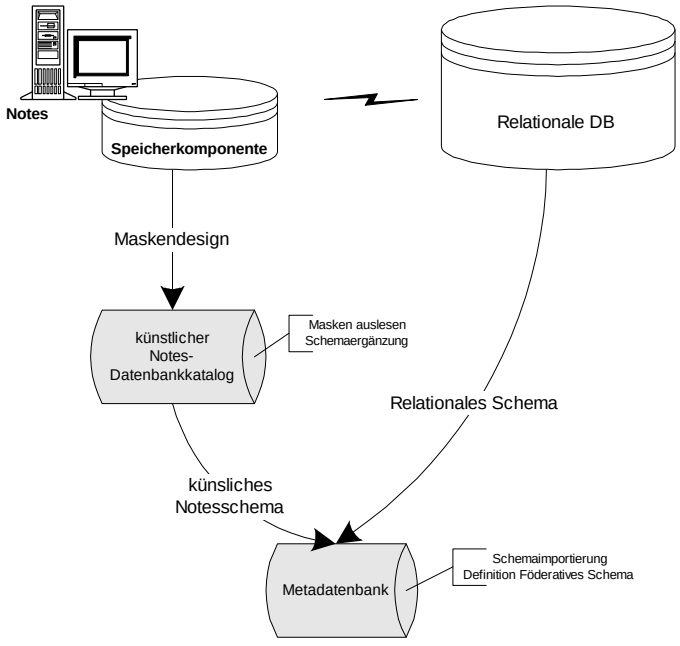


Abb. 7-59: Möglicher Aufbau für Administration

Bei der Definition des Föderativen Schemas könnte so direkt auf diesen Katalog zugegriffen werden. Dadurch würde der Nachteil der jetzigen Lösung beseitigt, daß für jedes Föderative Schema, in der die gleiche Notes Datenbank beteiligt ist, das Lotus Notes Schema erneut aus deren Masken ermittelt werden muß.

Der Prototyp in seiner derzeitigen Form kann bei der Definition einer Abbildung nur jeweils ein Feld mehreren Attributen zuordnen. Diese Einschränkung sollte bei einer Weiterentwicklung aufgehoben werden, so daß auch N zu M Abbildungen möglich sind. Dieses würde aber eine Änderung der Architektur der Metadatenbank unbedingt erforderlich machen, da derzeit die Antwort-Dokumente für die Definition einer Abbildung automatisch mit nur einem Feld erzeugt werden. Auch könnte man die Transformations- und Verschmelzungsformeln dahingehend erweitern, daß man ihnen zusätzliche Parameter übergibt. So könnte man beispielsweise einer Verschmelzungsformel mit dem Namen „Verketten mit X Leerzeichen“ die Anzahl der zu erzeugenden Leerzeichen mitgeben. Dieses würde den Umfang der Formeln in der Formeldatenbank erheblich reduzieren.

7.5.3 Werkzeuge für Synchronisation und Datentransfer

Der offenkundigste Nachteil des Prototyps ist seine Benutzerschnittstelle. Vom Administrator kann man verlangen, daß er bei der Definition des Föderativen Schemas umfangreiche Einstellungen in den Dokumenten der Metadatenbank machen muß. Dem Benutzer sollte man jedoch mit einem einfachen und klar strukturierten Dialog entgegenkommen. Eine Ideale Benutzerschnittstelle sähe folgendermaßen aus. Nach einem Abgleich werden alle gefundenen Datensätze in einer Tabelle aufgelistet.

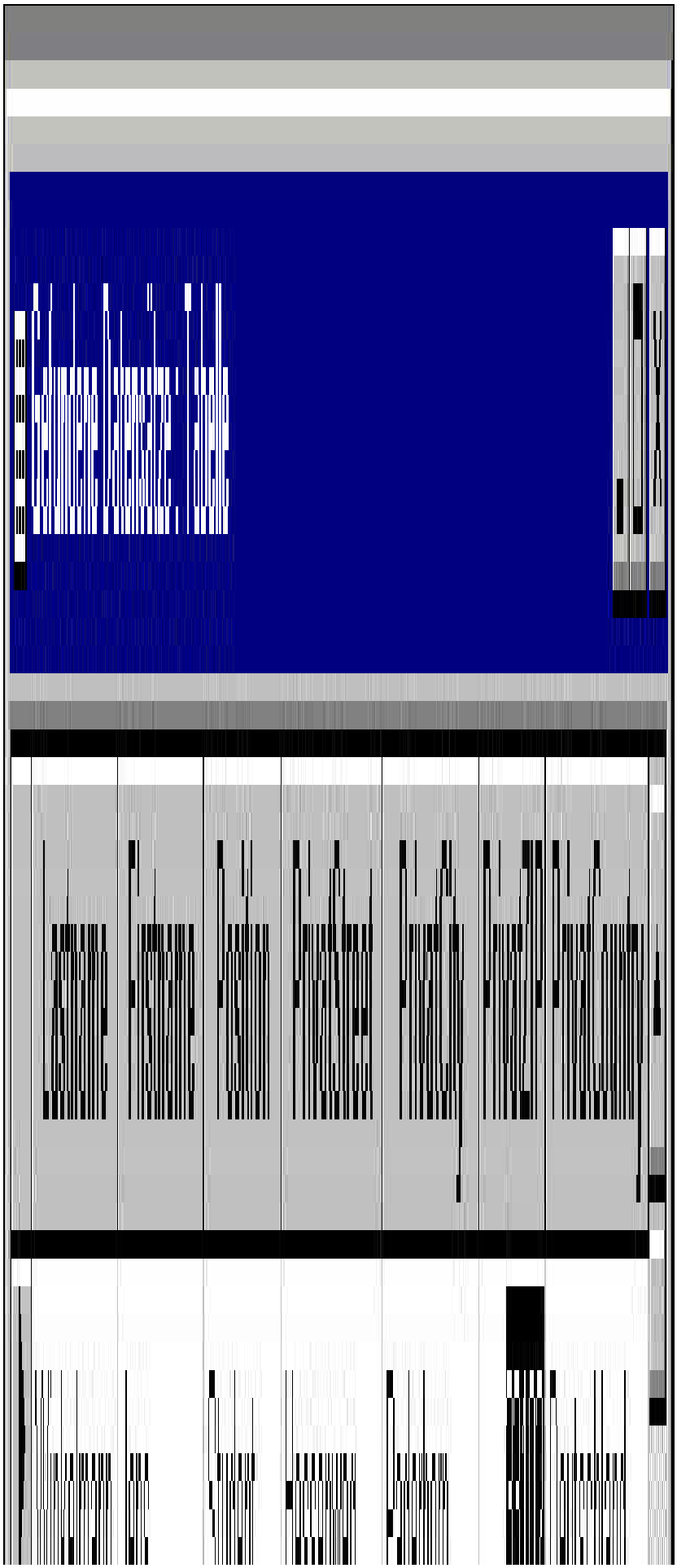


Abb. 7-60: Dialog für Synchronisation

Per Mausklick kann man dann diejenigen Werte markieren, die von der Relationalen Datenbank ins Dokument kopiert werden sollen. Ähnlich würde man beim Exportieren der Feldwerte verfahren. Leider sind die GUI Gestaltungsmöglichkeiten in Notes derart eingeschränkt, daß man sich mit der jetzigen Form begnügen muß. Möglicherweise bietet Notes 4.5 neue Features, die man zur Lösung dieses Problems nutzen kann. Das Nutzenpotential des Prototyps ließe sich auch dahingehend erweitern, daß der Benutzer erst im Dialog die Schlüssel für den Abgleich bestimmt. Bei der jetzigen Version werden, beim Erstellen des Föderativen Schemas durch den Admininstrator, einige Abbildungen als Schlüsselabbildungen bestimmt. Der Benutzer muß sic h mit dieser festen Vorgabe zufrieden geben.

Neben dem Dialogbetrieb sollte die Synchronisation auch als Stapelprozeß (Batch-Prozeß) ausführbar sein. Hierbei erfolgt der Abgleich von Relationaler Datenbank und Notes Datenbank ohne Mithilfe des Benutzers. Ähnlich wie bei einem Replikationsprozeß würden be ispielsweise über Nacht Adressen von GroupOffice und einer Relationalen Datenbank synchronisiert. Unstimmigkeiten zwischen beiden Datenbanken werden mit Hilfe vordefinierter Einbringstrategien beseitigt. Auch wäre es denkbar, daß der Prozeß erkannte Inkonsistenz nur in einer parallel geführten Protokolldatei vermerkt. Hierbei müßte der Administrator das Protokoll auswerten und könnte flexibel im Dialog Unstimmigkeiten beseitigen.

Praktischen Nutzen hat der Prototyp erst, wenn er auch den Datentransfer beherrscht (*Siehe 7.1.1 Datentransfer für Archivierung*), der unter anderem für die Archivierung notwendig ist. Datentransfer und Synchronisation sollte möglichst nahtlos in einem Werkzeuge integriert sein. So könnte man schrittweise die Fähigkeiten des Synchronisations-Werkzeugs erweitern, bis auch alle Funktionen des reinen Datentransfers abgedeckt werden.

Literaturverzeichnis

[DC96] Drira Choukri: Diplomarbeit: Information sharing and archiving in heteregeneous database environments - Architecture, Concepts and prototypes for relational - and document - oriented database systems Uni GH Paderborn Mai 1996

[FH96] Flach, G.; Heuer, A.; Langer, U.: Transparente Anfragen in föderativen Datenbanksystemen http://wwwdb.informatik.uni-rostock.de 01.07.1996

[GR96] o.V.: Groupware - Communication, Collaboration, Coordination http://www.lotus.com/devtools 14.10.1996

[DI96] o.V.: Lotus White Paper - Lotus Notes Enterprise and DBMS Integration http://www.lotus.com/devtools/250a.htm 04.12.1996

[KS96] Kusch J.; Saake G.: Verteilungstransparenz in Datenbanksystemen http://wwwiti.cs.uni-magdeburg.de/institut/veroeffentlichungen/96/KusSaa96.htm 04.12.1996

[LI96] o.V.: Lotus IBM - International Technical Support Centers - Developing Applications with Lotus Notes Release 4 http://www.lotus.com/devtools/21e6.htm 14.10.1996

[LN96] o.V.: Lotus Notes 4.1 Notes Help Online. 1995-96

[LS96] o.V.: LSX Toolkit 1.0 Documentation http://www.lotus.com/devtools/219e.htm 04.12.1996

[NP96] o.V.: Lotus White Paper - Lotus NotesPump 1.0: Server-Based Data Transfer Corporate Evaluator's Guide April 1996 http:/www.lotus.com/ntsdoc96/236a.htm 24.05.1996.

[PI96] o.V.: Lotus Notes Pump - Database Integration for Lotus Notes http://www.lotus.com/devtools 14.10.1996

[PL96] o.V.: Pressemitteilung - Lotus Notes: verbesserter Zugriff auf Transaktions- und relationale Datenbanksysteme http://www.lotus.com//devtools 14.10.1996

[SQ96] o.V.: Lotus NotesSQL 2.01 ODBC Driver Documentation http://www.lotus.com/devtools/2182.htm 04.12.1996

[TC96] Türker Can; Conrad Stefan: Using Active Mechanisms for Global Integrity Maintenance in Federated Database Systems. http://wwwiti.cs.uni-magdeburg.de/institut/veroeffentlichungen/96/TueCon96.htm 04.12.1996

[CK95] Colemann D.; Khanna R.: Groupware - Technology and Applications. Prentice Hall 1995

[LA95] Gansler Rick: Lotus Notes APIs - 12/95 Speech http://www.lotus.com/devtools/2146.htm 22.11.1996

[LL95] Lang, S.M.; Lockemann P.C.: Datenbankeinsatz , Berlin; Heidelberg u.a. 1995, S. 457-482

[LJ95] Larson, James: Database Directions - From Relational to Distributed, Multimedia, and Object-Oriented Database Systems, Upper Saddle River, N.J. 1995

[MS95] o.V.: Microsoft Visual C++ 4.0 Books Online Information System - InfoViewer. 1994-95

[NM95] o.V.: Notes Release 4 - Database Manager's Guide Cambridge 1995

[WB95] Wismans Benedikt: Database Connectivity. Wirtschaftsinformatik 37 (1995) 3, S. 317-319

[EN94] Elmasri, R.; Navathe, S.: Fundamentals of Database Systems. 2. Auflage , Redwood City California u.a. 1994

[LW94] Lamersdorf, Winfried: Datenbanken in verteilten Systemen - onzepte Lösungen Standards, Braunschweig; Wiesbaden 1994

[RE94] Rahm Erhard: Mehrrechner -Datenbanksysteme - Grundlagen der verteilten und parallelen Datenbankverarbeitung. Bonn; Paris; Reading,Mass u.a. 1994

[HR93] Hackathorn, R.; Enterprise Database Connectivity - The key to enterprise applicat ions on the desktop. New York; Chichester; Brisbane u.a. 1993

[ID93] o.V.: Informatik-Duden. Dudenverlag 2. Auflage, 1993

[KH92] Kudlich, Hermann: Verteilte Datenbanken - Systemkonzepte und Produkte. Berlin; München 1992.

[TA92] Tannenbaum Andrew: Computer -Netzwerke. 2. Auflage, Attenkirchen 1992

[HS91] Harbison, S.; Steele G.: C - A reference manual. 3. Aufl age, New Jersey 1991

[RS91] Rusinkiewicz, M.; Sheth, A.; Karabatis, G.: Specifying interdatabase depencencies in a multidatabase environment COMPUTER Innovative technology for computer professionals IEEE, 24 (1991) 12, S. 46-53

[WJ91] Won Kim; Jungyun Seo: Classifying schematic and data heterogeneity in multidatabase systems COMPUTER Innovative technology for computer professionals IEEE, 24 (1991) 12, S. 12-17

[ÖV91] Özsu M. Tamer; Valduriez Patrick: Principles of Di stributed Database Systems, Englewood Cliffs, New Jersey 1991

[AA90] Achilles, A.: SQL -Standardisierte Datenbanksprache vom PC bis zum Mainframe. 2. Auflage, München; Wien 1990

[SL90] Sheth, A.; Larson, J.: Federated Database Systems for Managing Distribu ted Heterogeneous, and Autonomous Databases. ACM Computing Surveys, 22 (1990) 3, S. 183-236.

[HH86] Hansen H.R.: Wirtschaftinformatik I - 5. Auflage, Stuttgart 1986

Ich versichere hiermit, daß ich meine Diplomhausarbeit „Informationsverteilung in Föderativen Datenbankumgebungen - Konzepte für Archivierungskomponenten in Groupwareanwendungen anhand eines Prototyps“ selbständig und ohne fremde Hilfe angefertigt habe, und daß ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Paderborn, den 21. Januar 1997