JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Entwicklung eines visuellen Integrationstools für das Design föderierter Data Warehouse Schemata

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Mag.rer.soc.oec.

im Diplomstudium

WIRTSCHAFTSINFORMATIK

Eingereicht von:
Peter Schweinzer, 0256427

Angefertigt am:
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuung:
o. Univ.-Prof. Dr. Michael Schrefl

Mitbetreuung:
Dr. Stefan Berger

Linz, Oktober 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, Oktober 2011

Peter Schweinzer

Danksagung

Hiermit möchte ich mich bei meinen Freunden und meiner Familie bedanken, die mich während meines Studiums und dem Verfassen der Diplomarbeit unterstützt haben und mir auch in schlechten Zeiten zur Seite gestanden sind.

Inhaltsverzeichnis

Kurzfassung

Fusionen bzw. Kooperationen zwischen Unternehmen führen oft zu Problemen bei der unternehmensübergreifenden Nutzung (z.B. Auswertung) der Daten. Diese Probleme können z.B. Inkonsistenzen zwischen den Schemata der einzelnen Unternehmen sein. Um diese Probleme beheben zu können, bieten sich föderierte Data Warehouses an. Diese bieten einen Zusammenschluss mehrerer autonomer Data Warehouses. Damit ist es möglich, Abfragen auf ein föderiertes System abzusetzen, welches die Teilabfragen transparent an die integrierten autonomen Data Warehouses weiterleitet.

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung eines visuellen Integrationstools, mit dessen Hilfe es möglich ist, Schemata der autonomen Data Warehouses in ein „globales Schema“ des föderierten Data Warehouses zu integrieren. Diese Arbeit ist Teil eines Gesamtsystems, mit dessen Hilfe ein föderiertes Data Warehouse System aufgebaut werden kann und Abfragen per SQL MDi (SQL for Multi-Dimensional Integration) abgesetzt werden können.

Bei dieser Arbeit wird sowohl auf die Theorie des Data Warehousing allgemein, als auch auf die Theorie zu föderierten Systemen und der Schemaintegration eingegangen.

Abstract

Mergers and acquisitions between companies cause problems for companywide analysis of data. One of these problems is the potential inconsistency between the different schemas of the companies. Federated data warehouses offer functionality to eliminate these problems. They offer a combination of autonomous data warehouses to a federated data warehouse. This way it is possible to create requests on a federated system which delegates the requests to the autonomous data warehouses.

The topic of this thesis is the development of a visual integration tool that makes it possible to integrate schemata of autonomous data warehouses into the global schema of a federated data ware house. This thesis is part of a complete system which helps to build a federated data warehouse and which is able to handle SQL MDi (SQL for Multi-Dimensional Integration) requests.

The theory of data warehousing in general, the theory of federated systems and schema integration are also explained in this thesis.

Abschnitt 1
Einleitung und Aufbau der Arbeit

1 Einleitung

In der heutigen Geschäftswelt ist es wichtig, Entscheidungen informationsunterstützt treffen zu können. Dies bedeutet, dass es Systeme geben muss, welche Information so aufbereiten, dass sie zur Entscheidungsfindung dient. So genannte „Decision Support Systems“ sollen dabei helfen, Entscheidungen aufgrund von historischen Daten und deren Analyse zu unterstützen (vgl. [Ritc04]). Dabei kommen Techniken und Methoden des Data Warehousing zum Einsatz. Data Warehousing befasst sich mit der analysegerechten Aufbereitung der Daten und den damit zusammenhängenden ETL Prozessen (Extraction, Transformation, Loading). Dies bedeutet, dass die Daten für nachfolgende Abfragen optimiert aufbereitet werden.

Möchte man bestehende Data Warehouses mit bereits bestehenden Schemata zu einem gemeinsamen Data Warehouse integrieren, stellt sich die Problematik, die Schemata aufeinander abzustimmen. Die Motivation hinter dieser Arbeit ist die Entwicklung eines Tools, welches die Integration der Schemata anhand einer grafischen Oberfläche ermöglicht.

Um diese Anforderung zu erfüllen, wird zuerst der theoretische Hintergrund erörtert. Anschließend werden bereits vorhandene Ansätze zur visuellen Datenintegration untersucht und für die Umsetzung der Implementierung verwendet.

1.1 Motivation und Zielsetzung

Ziel dieser Diplomarbeit ist die Entwicklung eines visuellen Integrationstools als Teil eines föderierten Data Warehouse Systems (siehe Abbildung 1).

Beim föderierten Data Warehouse System wird eine SQL MDi (SQL for Multi-Dimensional Integration) Abfrage von der Parser Komponente des Abfragewerkzeugs mit Hilfe der Metadaten aus dem Metadaten Repository verarbeitet. SQL MDi ist eine von Berger und Schrefl [BeSc06] entwickelte Abfragesprache. In weiterer Folge greift die Prozessor Komponente auf die lokalen Data Warehouses zu und generiert einen globalen Würfel.

Ziel dieser Diplomarbeit ist die Generierung von Metadaten und das Mapping zwischen globalem Schema und den lokalen Data Warehouse Schemata.

„Nicht-Ziel“ dieser Diplomarbeit ist die Verarbeitung von SQL MDi Abfragen.

Das Metadaten Repository dient als Schnittstelle zwischen Integrationstool und Parser Komponente.

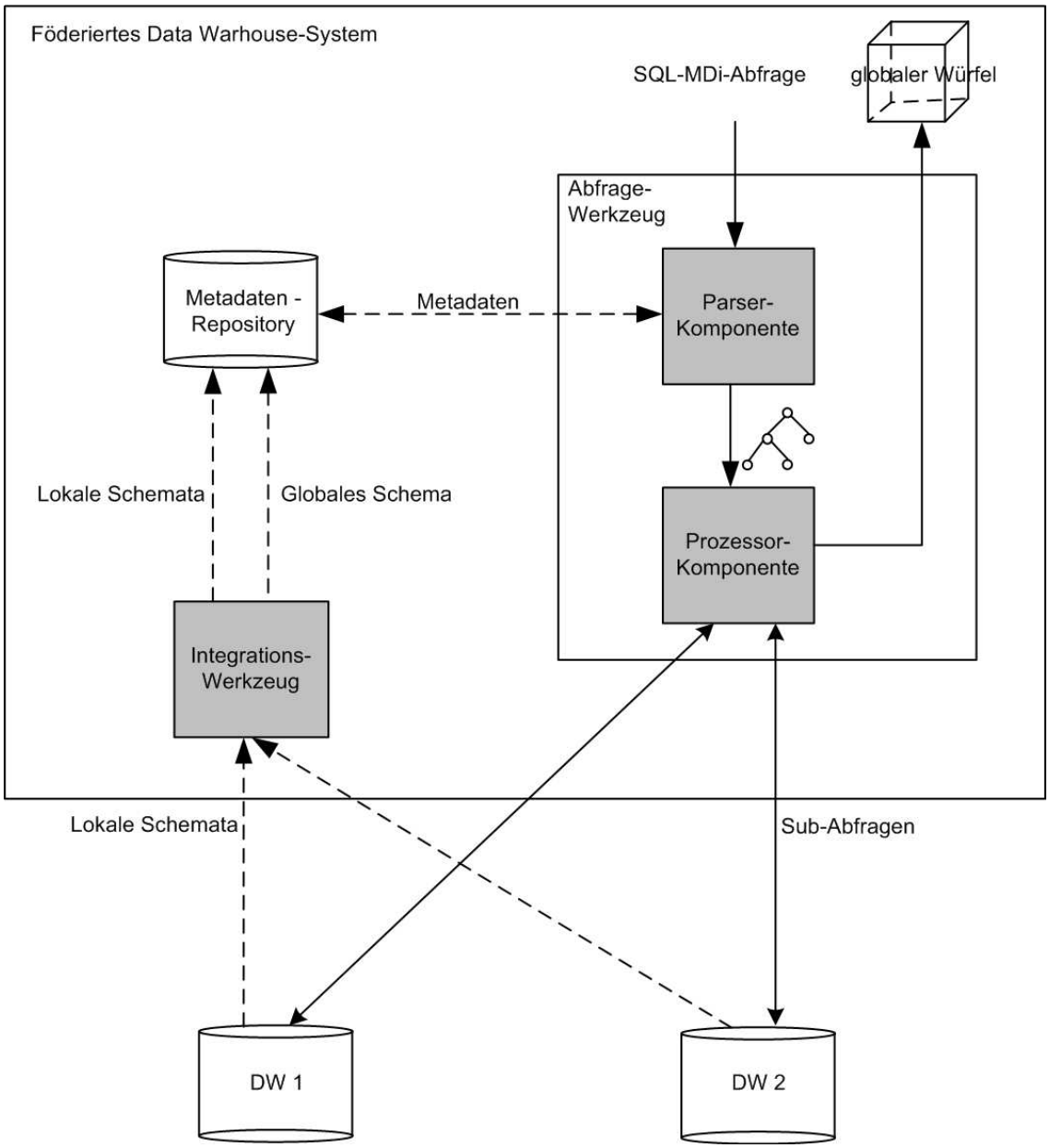

Abbildung 1: Föderiertes Data Warehouse-System [Brunn08]

Das Gesamtsystem ist in drei Komponenten aufgeteilt:

- Integrations-Werkzeug:
  Das Integrationstool (Integrations-Werkzeug) dient zur Erstellung von Mappings zwischen den lokalen Quellschemata und einem globalen Schema und der anschließenden Speicherung der Metadaten im Metadatenrepository.

- Parser-Komponente:
  Die Parser Komponente wandelt eingehende SQL MDi Abfragen in einen Operatorbaum um, welcher von der Prozessor-Komponente verarbeitet werden kann. Dafür wird auf die Metadaten im Metadatenrepository zugegriffen.
- Prozessor-Komponenten:
  Die Prozessor Komponente verwendet den Operator Baum für Abfragen auf die Quell-Data Warehouses und erstellt anschließend einen globalen Würfel zur Beantwortung der ursprünglichen, eingehenden SQL-MDi Abfrage.

Diese Diplomarbeit befasst sich nur mit dem Integrationstool. Parser-Komponente und Prozessor-Komponente werden in eigenständigen Diplomarbeiten (vgl. [Brun08] und [Ross08]) behandelt.

1.2 Aufbau der Diplomarbeit

Die Arbeit gliedert sich in vier Abschnitte. Zuerst wird der theoretische Hintergrund zu Data Warehousing und zu föderierten Datenbanksystemen allgemein erläutert. Anschließend wird auf spezifische Probleme zur Schemaintegration und zur visuellen Darstellung eingegangen. Im letzten Abschnitt wird das Visuelle Integrations-Tool (VIT) erläutert. Zuerst wird anhand einer Lösungsidee beschrieben, wie das Tool realisiert werden kann. Anschließend wird die Funktionalität anhand von Use Case- und Aktivitäts-Diagrammen erläutert. Abschließend wird noch auf die Architektur des VIT eingegangen.

Abschnitt 2
Theoretischer Hintergrund/Standards

2 Data Warehousing im Überblick

Dieses Kapitel beschäftigt sich mit Data Warehousing im Allgemeinen. Hier wird behandelt, was Data Warehousing eigentlich ist und welche Bereiche es im Data Warehousing gibt.

2.1 Data Warehouse

Das Data Warehouse bietet ein eigenes Datenmodell – das multi-dimensionale Datenmodell – sowie Techniken und Methoden für die optimierte Aufbereitung von historischen Daten für zukünftige Abfragen. Somit wird ein Data Warehouse als analytische Datenbank verwendet. Inmon und Hackathorn beschreiben das Data Warehouse wie folgt: „A data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management´s decision-making process“. (vgl. [InHa94])

Frei übersetzt bedeutet das, dass Data Warehousing eine themenorientierte, vereinheitlichte, zeitbezogene und dauerhafte Sammlung von Daten ist, die das Management bei Entscheidungen unterstützt. Nachfolgend werden diese Eigenschaften kurz erläutert, welche ein Date Warehouse auszeichnen.

2.1.1 Subject-oriented (Themenorientierung)

Die Daten werden anhand von Unternehmenssubjekten strukturiert, welche für Analysen und Entscheidungsprozesse und nicht für bestimmte operative Prozesse geeignet sind. Die Einteilung kann z.B. nach Produkt, Kunde, Standort usw. erfolgen. Im Gegensatz dazu liegen bei operativen Systemen die Daten so vor, dass sie für Prozesse und Applikationen optimiert sind. (vgl. [InHa94])

2.1.2 Integrated (Vereinheitlichung)

Die Daten des Data Warehouse stammen aus unterschiedlichen operativen Systemen welche in integrierter Form gehalten werden. Dies bedeutet, dass Daten, welche in verschiedenen Applikationen vorkommen und in verschiedenen Systemen gespeichert werden, für das Data Warehousing zu einer Datenbasis vereinigt werden. (vgl. [InHa94])

2.1.3 Time-variant (Zeitbezug)

Es wird die Dimension „Zeit“ eingeführt, um Analysen über den historischen Zeitverlauf durchführen zu können. Daher ist auch eine langfristige Speicherung der Daten notwendig. Dies bedeutet, dass Daten nicht nur über einen kurzen Zeitraum von einigen Monaten, sondern über einen langen Zeitraum von fünf bis zehn Jahren gespeichert werden. Für die Datenhaltung wird die Zeit immer als Schlüsselfeld verwendet. Ein bereits gespeicherter Datensatz kann nicht mehr nachträglich geändert werden (vgl. [InHa94]).

2.1.4 Nonvolatile (Dauerhaftigkeit)

Die Daten werden dauerhaft (nicht flüchtig) im Data Warehouse gespeichert. Bei operativen Datenbanken werden laufend Datensätze eingefügt, gelöscht und verändert. Beim Data Warehousing werden Datensätze nur einmal geladen und anschließend für Abfragen benutzt. (vgl. [InHa94])

2.2 Operative und analytische Datenbanken

Der Unterschied zwischen operativen und analytischen Datenbanken wurde von Lusti wie folgt beschrieben: „Während Produktionsdatenbanken vor allem das laufende Geschäft unterstützen, bereiten analytische Datenbanken taktische und strategische Entscheidungen vor.“ (vgl. [Lust02])

OLTP (Online Transaction Processing) beschäftigt sich damit, Produktionsdaten laufend aktuell zu halten. Dies beinhaltet das Einfügen, Löschen und Verändern von Produktionsdaten. Eine solche Datenhaltung ist für Abfragen auf wenige Datensätze optimiert. Beim Data Warehousing stehen allerdings aggregierte Daten im Vordergrund, welche für nachfolgende Analysen optimiert wurden. [Manh08].

Tabelle 1: Operative vs. analytische Datenbanken (nach [Lust02]) vergleicht die Unterschiede der Datenhaltung zwischen OLTP und Data Warehouse.

OLTP	Data Warehouse
Daten operativ	Daten strategisch
Daten vollständig	Daten periodenbezogen (historisch)
Daten detailliert	Daten oft abgeleitet (v.a. zusammenfassend)
Daten redundanzarm (fortschreibungsfreundlich)	Verarbeitung abfrageintensiv
Daten änderungsintensiv	Abfragen oft ad hoc
Datenmodell komplex	Schnittstellen vor allem endbenutzerorientiert

Tabelle 1: Operative vs. analytische Datenbanken (nach [Lust02])

2.3 Data Mart

Ein Data Mart ist eine eigene analytische Datenbank oder eine Menge von analytischen Datenbanken mit einem speziellen Fokus. Dieser Fokus kann ein spezieller Geschäftsbereich (siehe Abbildung 2) oder ein Fokus zur Entscheidungsunterstützung (z.B. Audit, Rentabilität) sein. [Silv08]

Ein Enterprise Data Warehouse (EDW) ist ein unternehmensweit genutztes Data Warehouse. Ein Data Mart wird verwendet, wenn das Enterprise Data Warehouse die Daten nicht in der Art und Weise (z.B. Aggregation) liefern kann, wie sie von den Datenkonsumenten erwartet werden und die Geschäftsbedürfnisse die Kosten für die Implementierung des Data Mart (z.B. zusätzliche Hard- und/oder Software) und den Overhead an Daten (d.h. deren doppelte bzw. mehrfache Speicherung) rechtfertigen.

Ein Data Mart ist physisch oder logisch vom EDW, von welchem es die Daten erhält getrennt. Der Data Mart ist eine Teilmenge (struktureller und/oder inhaltlicher Extrakt) eines EDW und enthält zumindest teilweise Daten vom EDW, kann aber auch Daten aus anderen Datenquellen enthalten. Dies ist einer der Gründe für die Verwendung eines Data Marts, da er einerseits die Entscheidungsunterstützung bietet, die die EDW Benutzer benötigen, und andererseits das EDW von fremden Datenquellen abschirmt. [Silv08]

Laut [Silv08] gibt es folgende Begründungen für den Einsatz von Data Marts:

- Data Marts erlauben, zusätzliche Daten aus einzelnen Geschäftsbereichen zu integrieren.
- Einzelne Geschäftsbereiche verfügen über sensible Daten, welche nicht für Personen außerhalb des Geschäftsbereichs verfügbar sein dürfen (z.B. medizinische Daten).
- Ein Geschäftsbereich kommuniziert mit anderen Firmen oder Regierungsbehörden, welche nicht auf die allgemeinen Daten im EDW zugreifen dürfen.

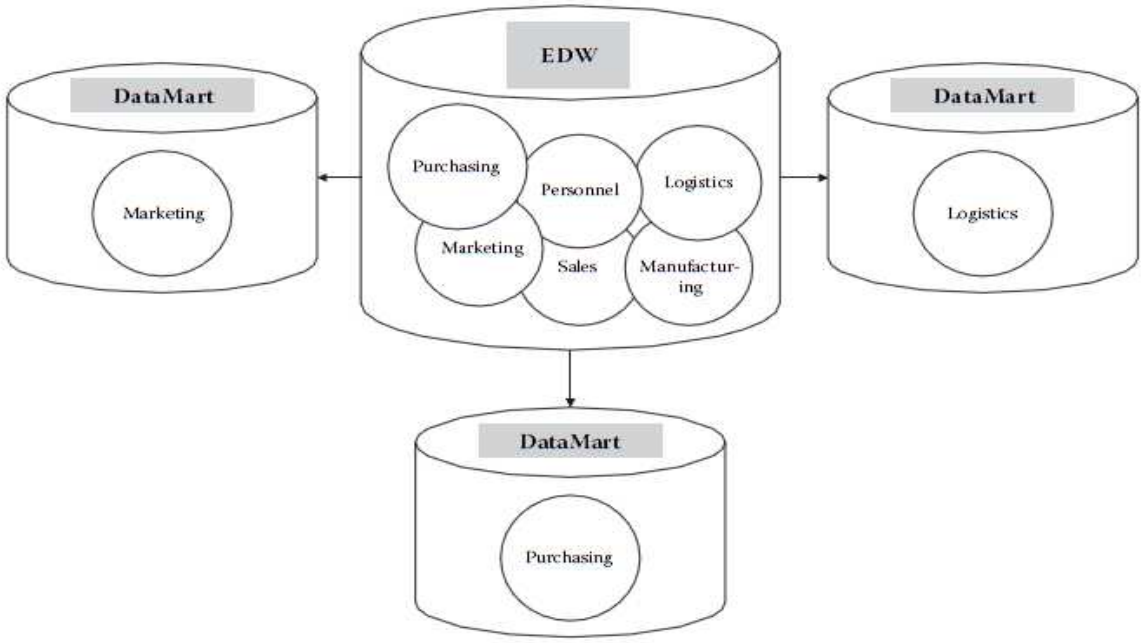

Abbildung 2: Data Mart (Quelle [Silv08])

Data Mart Architekturen

Es werden zwei grundlegende Architekturen für den Aufbau eines Data Marts [Lust02]:

- Top-down (unabhängige Data Marts - Kimball):

  Mehrere Data Marts werden zu einem zentralen Data Warehouse zusammengefasst (siehe Abbildung 3: Unabhängige Data Marts (nach [Rahm11]).

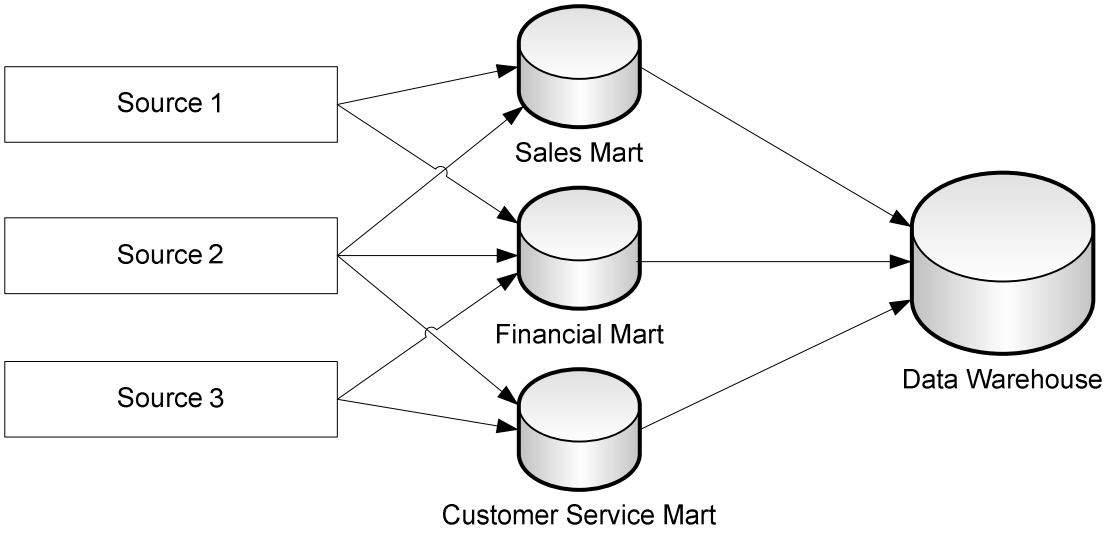

Abbildung 3: Unabhängige Data Marts (nach [Rahm11])

- Bottom-up (abhängige Data Marts - Inmon):

  Aus einem zentralen Data Warehouse werden mehrere Datamarts abgeleitet (siehe Abbildung 4).

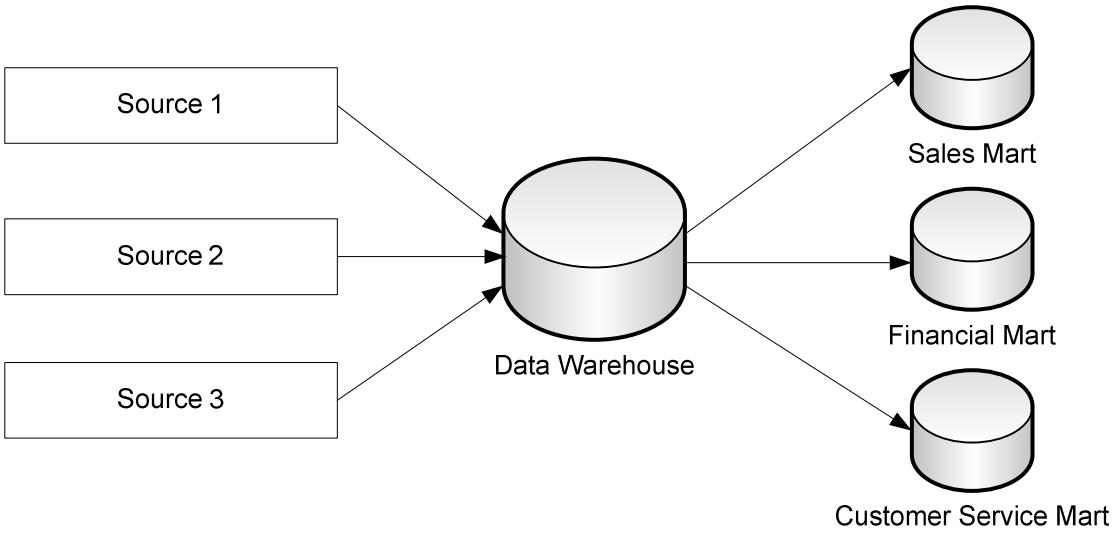

Abbildung 4: Abhängige Data Marts (nach [Rahm11])

In der vorliegenden Arbeit wird unter „Data Mart“ stets eine in sich abgeschlossene analytische Datenbank verstanden, ohne dass dabei die konkrete Architektur (abhängig vs. unabhängig) von Belang wäre.

2.4 Online Analytical Processing (OLAP)

Der Focus von OLAP ist die Bereitstellung einer Plattform für die Datenanalyse (z.B. Verkaufsdaten) mit mehreren Dimensionen (z.B. Produkt, Ort, Zeit) und mehreren Kennzahlen (z.B. Verkäufe oder Kosten). Eine Dimension kann hierarchisch sein, wobei eine Hierarchie mehrere Levels haben kann. OLAP Operationen ermöglichen die Sicht auf diese Daten aus mehreren Perspektiven. Das Objekt bzw. die Struktur, welche beim OLAP für Analysen von Interesse ist, ist der Data Cube. [Alum09] Ob ein Data Cube aus einem (E)DW oder einem Data Mart ermittelt wird, ist aus OLAP-Sicht transparent bzw. irrelevant.

OLAP ist eine Softwaretechnologie bzw. eine Klasse von Applikationen, die für Benutzer eines Data Warehouses komfortablen Zugriff auf die im Data Warehouse gespeicherten Daten ermöglicht.

Abbildung 5 zeigt die Mehrdimensionalität des Data Warehousing. Es wird ein Data Cube gebildet, welcher aus den Dimensionen Produkt, Region und Jahr besteht. Die Dimension Produkt ist in hierarchisch aufgebaut und enthält die Level Produktkategorie, Produktgruppe und Produkt. Die einzelnen Zellen werden Fakten genannt, welche durch Kennzahlen (z.B. Verkäufe) repräsentiert werden.

Ausgehend von a) [Produktkategorie] über b) [Produktgruppe] bis hin zu c) [Produkt] sieht man die jeweils feinere Granularitätsstufe der Dimension Produkt. [DeEa05]

Fasst man mehrere Produkte in eine Produktkategorie zusammen nennt man diesen Vorgang „Roll up“ bzw. „Drill up“. Den umgekehrten Vorgang von einer gröberen auf eine feinere Granularitätsstufe nennt man „Roll down“ bzw. „Drill Down“. Neben diesen Funktionen gibt es noch weitere wie: Slicing and Dicing, Drill Through, Drill Across und Filtern [Lust02]

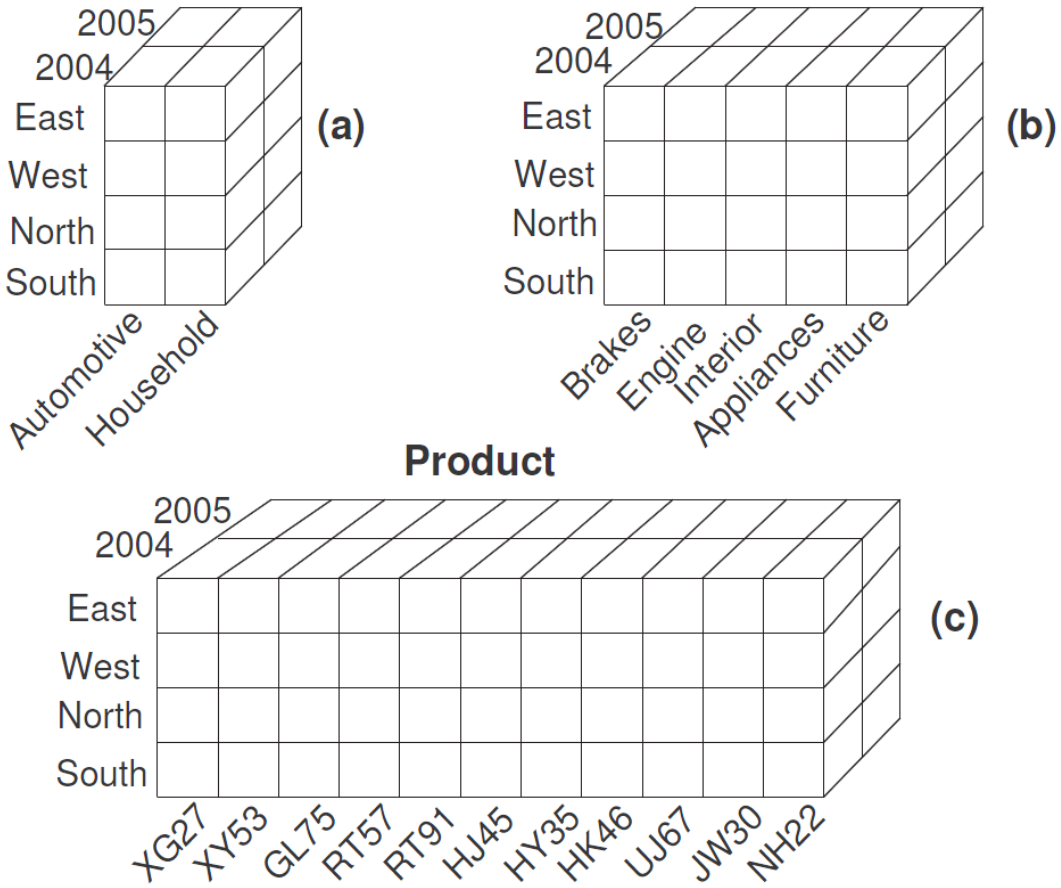

Abbildung 5: Dimensionen und Fakten (Quelle [DeEa05])

Beispiel:

In diesem Beispiel wird dargestellt, wie die Kennzahlen mittels „Drill up“ in den Dimensionen aggregiert werden können. Die Kennzahlen für die Verkäufe sind nicht aus der Abbildung ersichtlich. Das Produkt „NH22“ wurde im Jahr „2004“ in der Region „East“ 50-mal verkauft. Mittels Drill up kann das Produkt mit anderen Produkten (z.B. JW30 und UJ67) zur Produktgruppe „Furniture“ aggregiert werden, in welcher im Jahr „2004 in der Region „East“ 300 Produkte verkauft wurden.

Es gibt drei Arten der technischen Realisierung des Zugriffs auf Data Warehouse Daten nach der „Data Cube“ – Metapher: ROLAP, MOLAP und DOLAP. [Lust02].

2.4.1 ROLAP (relational OLAP)

Die Objektdaten eines Data Cube werden in einem relationalen Datenbanksystem gespeichert und erzeugen Multidimensionalität durch Abbildung der Objektdaten auf Indikatoren und ihre Dimensionen. Der Vorteil von ROLAP ist die bessere

Skalierbarkeit mit der Größe des Data Cube (d.h. mit einer Anzahl an Fakten) aufgrund der relationalen Datenbankstruktur. Nachteile sind die langsamere Zugriffszeit im Vergleich zu MOLAP und die Notwendigkeit der Abbildung der Multidimensionalität in einer relationalen Datenbankstruktur. [Claus98].

2.4.2 MOLAP (multidimensional OLAP)

Die Objektdaten eines Data Cube werden multidimensional in einem multidimensionalen Datenbanksystem gespeichert, welches die meisten Abfragen vorberechnet. Der Vorteil von MOLAP ist die Abfrageeffizienz. Ein Nachteil ist, dass für MOLAP im Vergleich zu ROLAP mehr Speicherplatz benötigt wird [Claus98].

2.4.3 DOLAP (Desktop OLAP)

Beim DOLAP liegen die Daten auf dem Client Rechner. Die Daten können dabei relational oder multidimensional vorliegen. Nachteil bei DOLAP ist die schwächere Performance aufgrund der weniger leistungsfähigen Hardware von Desktop Rechnern im Vergleich zur Hardware von Server-Systemen. [Claus98]

2.5 Modellierung

„Ein wichtiges Ziel der Datenmodellierung ist die Datenintegrität, das heißt die Korrektheit, Konsistenz und Vollständigkeit der Daten.“ [Lust02]

Die Daten in Data Warehouses sind statisch und werden normalerweise nicht in Echtzeit, sondern nur in bestimmten Zeiträumen aktualisiert. Daher muss das operative Datenmodell transformiert werden, um für analytische Abfragen im Data Warehouse optimiert zu werden.

Die Transformation operativer Datenmodelle in analytische Datenmodelle bedeutet:

- Überflüssige operative Daten entfernen: Daten, welche nicht für die Auswertung benötigt werden, werden beim Laden der Daten ignoriert (z.B. Kontaktdaten bei Datensätzen für Lieferungen).
- Die Zeitdimension integrieren: Um Verändungen der Daten über einen Zeitverlauf auswerten zu können, wird die Zeitdimension integriert. Dies ist z.B. für Monats- bzw. Jahresauswertungen wichtig.

- Ableitung definieren: Kenngrößen/Attribute, die nicht in direkter Form vorhanden sind, können über Ableitungen definiert werden (z.B. Umsatz = Preis * Menge).

Die Modellierung von Data Warehouses wird in die folgenden drei Bereiche gegliedert [Lust02]:

- Konzeptuelle Modellierung:
  Dies ist eine systemunabhängige Modellierung.
- Logische Modellierung:
  Modellierung, welche von der OLAP Variante (siehe Kapitel 2.4) abhängig ist.
- Physische Modellierung:
  Modellierung, welche vom Datenbanksystem abhängig ist.

2.5.1 Konzeptuelle Modellierung

Das konzeptuelle Modell identifiziert die Geschäftsbereiche für das Data Warehouse. Hierbei werden die einzelnen Teilbereiche abgegrenzt und Dimensionen und Fakten ermittelt (siehe Abbildung 5). Die konzeptuelle Modellierung ist von der verwendeten OLAP Variante und vom verwendeten Datenbanksystem unabhängig.

Die einzelnen Geschäftsbereiche werden in Faktschemata abgebildet. Dabei handelt es sich um Schemata mit Fakten, Kennzahlen, Dimensionen und Hierarchien. Zur Darstellung des nachfolgenden Beispiels wird das Dimensional Fact Model (DFM) [GoMa98] verwendet.

Beispiel:

Es sollen Verkäufe von Produkten in den Filialen eines Handelsunternehmens in einer gewissen Periode abgebildet werden. Abbildung 6 zeigt das zugehörige Faktschema. Für die Analyse von Verkäufen soll die Menge je Produkt und Standort ermittelt werden. Dafür wird das Fakt Verkäufe mit der Kennzahl Menge verwendet. Für die Zeithierarchie werden die dimensionalen Attribute Datum, Monat, Jahr und Quartal verwendet. Für die Standort Hierarchie, werden die dimensionalen Attribute Region und Standort verwendet. Für die Produkt Hierarchie werden die dimensionalen Attribute Produktkategorie, Produktgruppe

und Produkt verwendet. Weiters gibt es noch ein nichtdimensionales Attribut Beschreibung in der Dimension Produkt, über welches nicht aggregiert werden kann.

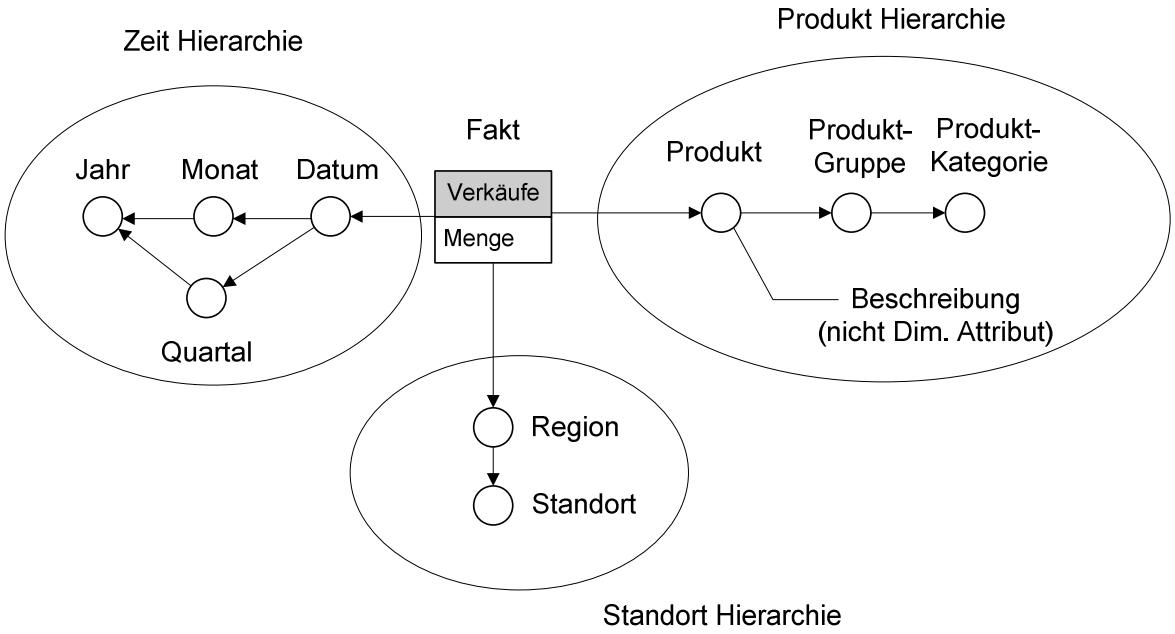

Abbildung 6: Konzeptuelles Schema (DFM)

2.5.2 Logische Modellierung

Bei der logischen Modellierung wird das konzeptuelle Schema in das Datenmodell der gewählten OLAP-Variante transformiert und um nachfolgende Information erweitert (vgl. [Silv08]):

- Definition des Primärschlüssels
- Definition der Attribute der Dimensions- bzw. Fakttabellen
- Primär-/Fremdschlüsselbeziehung zwischen den einzelnen Tabellen
- Kardinalität zwischen den Tabellen

Meist wird ROLAP verwendet. Dafür stehen Star-Schema und Snowflake-Schema zur Verfügung. Das logische Modell kann auf verschiedene Arten realisiert werden. Nachfolgend sind die zwei gängigsten Entwurfsmuster (Star-Schema und Snowflake-Schema) beschrieben.

2.5.2.1 Star-Schema

Das Star-Schema ist das am weitesten verbreitete logische Modell analytischer Datenbanken auf relationalen Systemen. [Lust02]

Abbildung 7 zeigt für das in Abbildung 6 gewählte Beispiel eine Fakttabelle Verkäufe mit den dazugehörigen Dimensionstabellen Standort, Produkt und Datum. Die jeweiligen Primärschlüssel der Dimensionstabellen sind der Fakttabelle als Fremdschlüssel eingetragen und bilden gemeinsam den Primärschlüssel der Fakttabelle.

Ein Star-Schema hat folgende Eigenschaften [Lust02]:

- Mehrere Dimensionstabellen beziehen sich auf genau eine Fakttabelle
- Die Fakttabelle enthält die Attribute (meist mit stetiger Skala), die betriebliche Erfolgskriterien messen (= Kennzahlen)
- Dimensionstabellen enthalten meist symbolische und diskrete Attribute und erlauben die Auswahl, Zusammenfassung und Navigation der Fakten
- Jede Dimensionstabelle steht in einer 1:n Beziehung zur Fakttabelle
- Die 1:n Beziehung wird über einen Schlüssel der Dimensionstabelle und einen Fremdschlüssel der Fakttabelle vermittelt

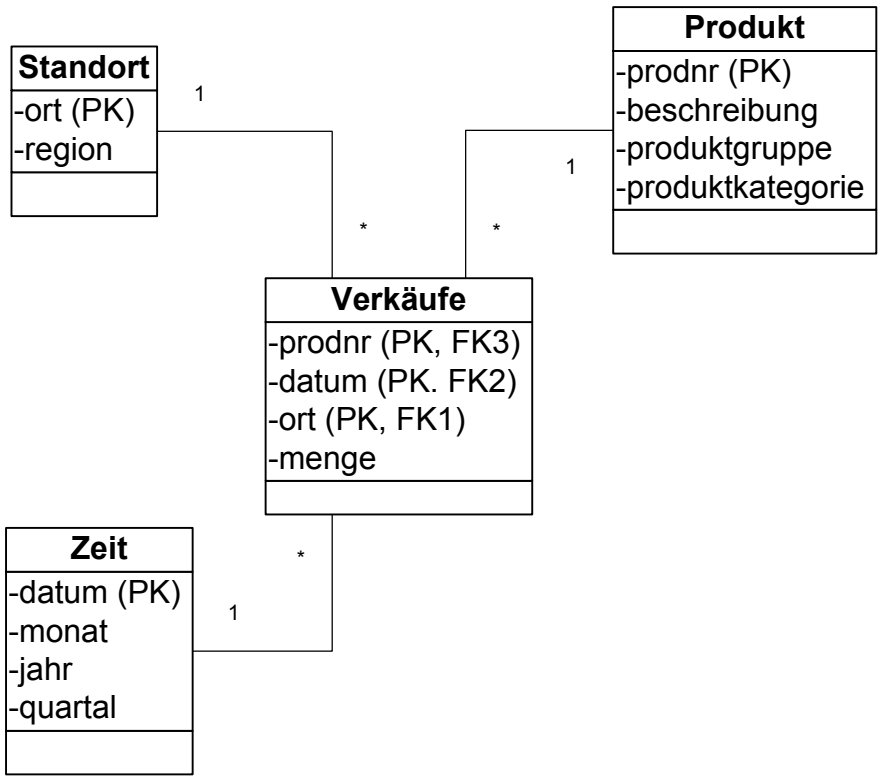

Abbildung 7: Star-Schema

2.5.2.2 Snowflake-Schema

Beim Snowflake-Schema werden die Dimensionen im Vergleich zum Star-Schema normalisiert. Dimensionen werden abgetrennt, um die Redundanz zu reduzieren und die Fortschreibungsfreundlichkeit und Speichereffizient zu verbessern (z.B. Dimensionstabelle Produktkategorie mit Schlüssel von Dimension Produktgruppe; Dimensionstabelle Produktgruppe mit Schlüssel von Dimension Produkt). Weiters werden Fakten unterschiedlicher Granularität unterschieden um z.B. Gesamtkosten und Einzelkosten zu berechnen. [Lust02]

Abbildung 8 zeigt für das in Abbildung 6 gewählte Bespiel eine Fakttabelle Verkäufe und die zugehörigen Dimensionstabellen für die Produkthierarchie. Die Zeithierarchie und die Standorthierarchie wurden aufgrund der Übersichtlichkeit nicht dargestellt.

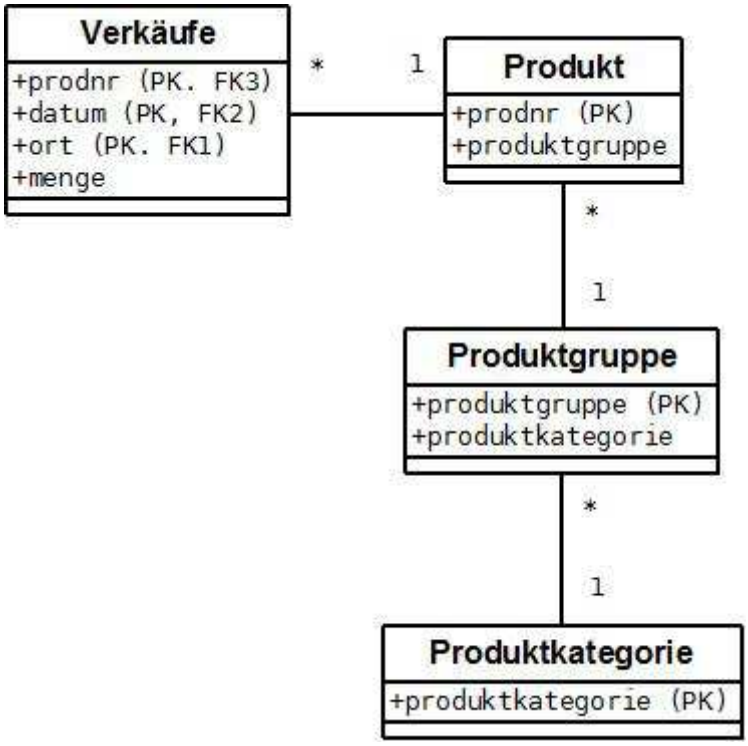

Abbildung 8: Snowflake-Schema (Produkthierarchie)

2.5.3 Physische Modellierung

Bei der physischen Modellierung wird auf Gegebenheiten der verwendeten Datenbanksysteme Rücksicht genommen. Hier sind die Auswahl der optimalen Speicherstrukturen und der Entwurf der Indexstrukturen von entscheidender Bedeutung. Auf Details wird im Rahmen dieser Diplomarbeit nicht näher eingegangen.

2.6 Entwicklung und Betrieb

In diesem Abschnitt wird auf die Entwicklung und die Optimierung im Betrieb eines Data Warehouse eingegangen.

2.6.1 Entwicklungsphasen

Spezifikationsphase

In der Spezifikationsphase wird eine Ist/Soll-Analyse durchgeführt und der derzeitige Stand der Datenquellen erhoben. Es werden die Anforderungen und Ziele formuliert. Allgemeine Anforderungen sind: Abbildungstreue, Benutzerfreundlichkeit, Flexibilität, Skalierbarkeit, Performance, Verfügbarkeit, Wartbarkeit und Kompatibilität. [Lust02]

In der Spezifikationsphase müssen die genauen Wünsche und Forderungen des Managements definiert werden und die optimale System-Plattform für die Realisierung gefunden werden. (vgl. [InHa94])

Realisierungsphase

In der Realisierungsphase geht es darum, wie das Ziel erreicht werden soll. Angefangen mit dem Entwurf, über die Implementierung bis hin zum Betrieb wird der gesamte Entwicklungslebenszyklus durchlaufen. Speicher- und Zugriffsoptimierung stehen in der Realisierungsphase oft im Konflikt mit der Datenverantwortung und –sicherheit. [Lust02]

Bei der Realisierung wird zunächst das konzeptuelle Schema (siehe Kapitel 2.5.1) erstellt, welches anschließend in ein logisches Schema (siehe Kapitel 2.5.2) und zum Schluss in ein physisches Schema (siehe Kapitel 2.5.3) abgeleitet wird.

2.6.2 Laden operativer Daten

Operative Daten müssen extrahiert, transformiert und in ein Data Warehouse geladen werden (Extraction, Transformation, Loading → ETL). Die Produktionsdaten werden aus unternehmensweiten Client/Server-Datenbanken oder Legacy-Systemen in ein Data Warehouse übernommen.

Die Daten aus den operativen Systemen werden so transformiert, dass sie für nachfolgende Analysen besser geeignet sind, z.B. Entfernung von Duplikaten und irrelevanten Attributen. Nicht automatisierbar ist die Neudefinition und Zusammenfassung detaillierter Merkmale in Attribute geringerer Granularität.

Nach Extraktion und Transformation erfolgt die Integration in ein Data Warehouse für nachfolgende OLAP-Abfragen. [Lust02]

2.6.3 Speicher- und Laufzeitoptimierung

Die Optimierung kann in die folgenden drei Bereiche eingeteilt werden: logisches Datenmodell, physisches Datenmodell und Hardware und Systemsoftware. Beim logischen Datenmodell ist es wichtig Datenvolumen zu reduzieren, homogene Abfragen gegebenenfalls in lokale Data Marts zusammenzufassen, Verbundoperationen durch Denormalisierung zu verringern und abfrageeffiziente Datenmodelle wie das Star-Schema zu verwenden. Beim physischen Datenmodell sollten die Ladezeiten verringert werden, indem man die ETL Prozesse optimiert. Die Antwortzeiten können optimiert werden, indem man z.B. Abfragen vordefiniert oder den Zugriff mit B-Bäumen, Bitmuster-Indizes oder Hashfunktionen beschleunigt. Der Speicheraufwand kann verringert werden, indem man nicht benötigte Daten archiviert, Nullwerte sparsam speichert und Daten und Indizes komprimiert. Die Verfügbarkeit kann verbessert werden, indem man Daten inkrementell lädt. Die Hardware und Systemsoftware kann optimiert werden, indem man Einzelprozessorsysteme beschleunigt und Auswahl- und Aktionsabfragen parallelisiert. [Lust02]

2.7 Fazit

In Kapitel 2 wurden die theoretischen Grundlagen des Data Warehousing erläutert, die für das Verständnis des Visual Integration Tool (VIT) erforderlich sind. Dabei ist besonders das Verständnis über den Aufbau eines Data Warehouse mit Dimensionen und Fakten und die Modellierung als Star-Schema erforderlich. Da das Star-Schema das am häufigsten verwendete Schema ist, wird ausschließlich dieses Schema für das VIT verwendet, welches im Rahmen dieser Diplomarbeit entwickelt wird.

3 Föderierte Data Warehouses / -Datenbanksysteme

In diesem Kapitel werden die Grundlagen zu föderierten Datenbanksystemen erläutert, welche auch für föderierte Data Warehouses gültig sind. Es wird auf die Architektur und die Möglichkeiten zur Schemaintegration eingegangen.

3.1 Überblick

„Die Integration von Datenbeständen zur Schaffung eines einheitlichen Zugriffs ist eines der zentralen Ziele, das durch die Föderierung von Datenbanken verfolgt wird“. [Conr97]

Föderierte Datenbanksysteme werden entweder für unternehmensinterne oder unternehmensübergreifende Zusammenschlüsse verwendet. Letztere werden auch häufig Global Information-Sharing Systems genannt. [Conr97]

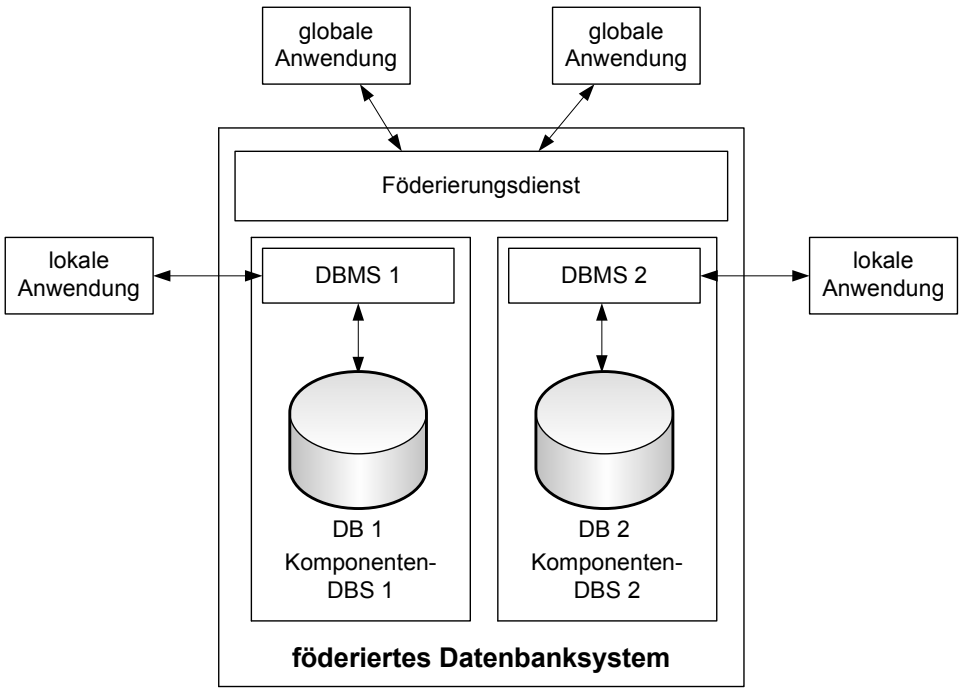

Abbildung 9: Föderiertes Datenbanksystem (nach [Conr97])

Abbildung 9 zeigt die Struktur eines föderierten Datenbanksystems. Die ursprünglichen Datenbanksysteme, auf welche die lokalen Anwendungen zugreifen, werden als Komponentensysteme für das föderierte Datenbanksystem verwendet. Die globalen Anwendungen greifen dann auf den Föderierungsdienst zu. [Conr97]

Die Abgrenzung zwischen zentralen und verteilten Datenbanksystemen, sowie Multidatenbanksystemen erfolgt dahingehend, dass letztere normalerweise ein Verbund von mehreren Datenbanksystemen sind, während bei den anderen beiden der gesamte Datenbestand von einem einzigen Datenbankmanagementsystem (DBMS) verwaltet wird. Bei Multidatenbanksystemen wird unterschieden, ob es sich um föderierte oder nicht föderierte Datenbanksysteme handelt. Sind die einzelnen Komponentensysteme weiterhin in gewissem Grad autonom, handelt es sich um ein föderiertes Datenbanksystem. Wird das Gesamtsystem auf einer übergeordneten Ebene verwaltet, handelt es sich um ein nicht föderiertes System. Föderierte Datenbanksysteme teilen sich auf in lose (Benutzer stellt die Föderation zusammen) und eng (Administrator stellt die Föderation zusammen) gekoppelte Systeme. (vgl. [Conr97] und [ShLa90])

3.2 Verteilung, Heterogenität und Autonomie

„A federated database system (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous.” [ShLa90]

Die drei Bereiche Verteilung, Heterogenität und Autonomie spielen bei der Einteilung von Datenbanksystemen eine große Rolle. Die Datenbanksysteme lassen sich an Hand von Abbildung 10 klassifizieren, wobei jeweils die Ausprägung der drei Bereiche ersichtlich ist.

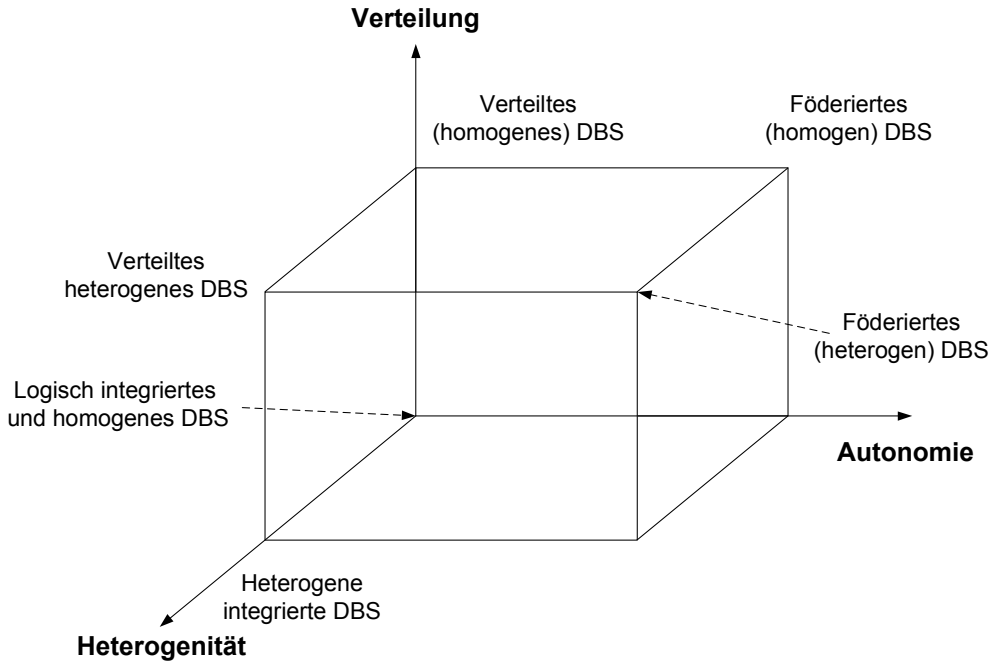

Abbildung 10: Architekturvarianten von Datenbanksystemen (nach [ÖzVa91])

3.2.1 Verteilung

Wird von Verteilung gesprochen, ist meist die Verteilung der Daten gemeint. Die Daten können auf einem Rechner oder mehreren verteilt vorliegen. Die Daten können auch auf mehrere Datenbanken verteilt vorliegen, welche von einem zentralen System verwaltet werden. Liegen Daten redundant vor, muss das System die Redundanz kontrollieren und verwalten. Operationen liegen normalerweise nicht verteilt vor. Das Schema (Metadaten) kann ebenfalls verteilt vorliegen. (vgl. [ÖzVa91], [ShLa90] und [Conr97])

3.2.2 Heterogenität

Bei Heterogenität wird zwischen heterogenen Datenbanksystemen und semantischer Heterogenität unterschieden.

Heterogene Datenbanksysteme bauen entweder auf unterschiedlichen Datenmodellen auf und haben daher auch eine unterschiedliche Struktur oder es werden gleiche Datenmodelle durch unterschiedliche Schemata dargestellt bzw. unterstützen die Datenmodelle unterschiedliche Integritätsbedingungen.

Semantische Heterogenitäten sind im Gegensatz zu Unterschieden in Datenbanksystemen schwer zu entdecken und zu behandeln. Unter semantischer Heterogenität versteht man die unterschiedliche Auffassung von Daten, wie es z.B. bei einem Preis ohne Währungs- bzw. Steuerangaben der Fall ist. (vgl. [ÖzVa91], [ShLa90] und [Conr97])

3.2.3 Autonomie

Autonomie lässt sich in die drei Teilbereiche gliedern: Entwurfs-, Kommunikations- und Ausführungsautonomie.

Entwurfsautonomie bedeutet die autonome Entwicklung der einzelnen Datenbanksysteme. Die lokalen Datenbankschemata sollen auch nach Eingliederung in die Föderation erhalten bleiben. Theoretisch sollten die Komponentensysteme jederzeit in der Lage sein, die lokalen Schemata beliebig zu ändern. Der Integrationsaufwand ist hierbei allerdings sehr hoch, da bei lokalen Änderungen der einzelnen Schemata auch das globale Schema entsprechend angepasst werden muss. (vgl. [ÖzVa91], [ShLa90] und [Conr97])

Unter **Kommunikationsautonomie** versteht man die Entscheidungsfreiheit der einzelnen Komponentensysteme mit welchen Systemen sie kommunizieren wollen. D.h. ein System kann selbst entscheiden (z.B. Datenbankadministrator trifft Entscheidung), ob und wann es in eine Föderation integriert wird.

Unter **Ausführungsautonomie** versteht man, dass jedes Datenbanksystem in der Lage sein soll, selbst zu entscheiden, welche Anwendungsprogramme, Anfragen und Änderungsoperationen es ausführt und wann die Ausführung stattfindet. In Multidatenbanksystemen kommt es bei einer übergeordneten Transaktionsverwaltung häufig dazu, dass Datenbanksysteme ihre Ausführungsautonomie verlieren.

3.3 Referenzarchitekturen

Eine Referenzarchitektur ist wichtig, um die verschiedenen Eigenschaften eines Datenbanksystems zu erklären. Jede Komponente der Referenzarchitektur befasst sich mit einem wichtigen Teil eines Datenbanksystems. Referenzarchitekturen bieten ein Framework, mit welchem man die unterschiedlichen Architekturoptionen verstehen, kategorisieren und vergleichen kann, um föderierte Datenbanksysteme zu entwickeln. Die Referenzarchitektur zeigt auf, welche Komponenten ein föderiertes Datenbank System (FDBS) enthalten kann. In vielen FDBSs werden nicht alle der oben besprochenen Elemente verwendet. [ShLa90]

Nachfolgend werden die benötigten Komponenten und Schematypen für eine Referenzarchitektur erläutert.

3.3.1 Systemkomponenten einer Referenzarchitektur

Referenzarchitekturen beinhalten verschiedene Systemkomponenten. Nach [ShLa90] wären dies folgende:

Daten: Daten sind die Basisfakten und -informationen, welche von einem Datenbanksystem verwaltet werden.

Datenbank: Eine Datenbank ist ein Repository von strukturierten Daten entsprechend eines Datenmodells.

Befehle: Befehle sind Anfragen für spezielle Aktionen, welche entweder von einem Benutzer eingegeben oder von einem Prozessor generiert werden.

Prozessoren: Prozessoren sind Softwaremodule, welche Befehle ausführen und Daten manipulieren.

Schemas: Schemas sind Beschreibungen von Daten, welche von einem oder mehreren DBMSs verwaltet werden. Ein Schema besteht aus Schemaobjekten und deren Beziehungen.

Mappings: Mappings sind Funktionen, die Schemaobjekte in einem Schema mit Schemaobjekten in einem anderen Schema in Beziehung bringen.

3.3.2 Prozessoren

Die Prozessoren zur Verarbeitung der Daten gliedern sich in die Bereiche Transformationsprozessor, Filterungsprozessor, Konstruktionsprozessor und Zugriffsprozessor.

3.3.2.1 Transformationsprozessor

Der Transformationsprozessor dient zur Übersetzung der Befehle von der Quellsprache in die Zielsprache oder transformiert Daten vom Quellformat ins Zielformat. Die Transformationsprozessoren garantieren Daten-Modell-Transparenz, indem sie die Strukturen und Befehle, welche von einem Prozessor benutzt werden, vor anderen Prozessoren verbergen. Um Transformationen ausführen zu können, benötigt der Transformationsprozessor Mappings zwischen den einzelnen Objekten der Schemata.

3.3.2.2 Filterungsprozessor

Der Filterungsprozessor beschränkt die Befehle und Daten, welche zu einem Prozessor weitergeleitet werden, mittels syntaktischer und semantischer Prüfung.

Die **Syntaktische Prüfung** prüft anhand eines „Syntax Check“, ob Befehle syntaktisch korrekt sind. D.h. ob die Zeichenkette eines Befehls einem erwarteten vordefinierten Format entspricht.

Die **Semantische Prüfung** prüft anhand von Prüfregeln, ob Befehle die semantischen Integritätsbedingungen nicht verletzen, modifiziert gegebenenfalls Befehle dahingehend, dass die semantische Integrität gewährleistet ist und prüft, ob die Daten, welche von einem anderen Prozessor generiert wurden, die semantischen Integritätsbedingungen erfüllen.

Die **Zugriffskontrolle** prüft anhand eines Berechtigungskonzepts, ob ein Benutzer die Berechtigung hat Befehle auszuführen bzw. die Daten zu verwenden.

3.3.2.3 Konstruktionsprozessor

Der Konstruktionsprozessor partitioniert und/oder repliziert eine Operation, welche von einem einzelnen Prozessor kommt, in mehrere Operationen, welche von zwei oder mehreren Prozessoren verarbeitet werden. Konstruktionsprozessoren fügen außerdem die Daten, welche von mehreren Prozessoren produziert wurden in einen Datensatz zusammen, welcher von einem Prozessor verarbeitet wird.

3.3.2.4 Zugriffsprozessor

Ein Zugriffsprozessor akzeptiert Befehle und produziert Daten durch Ausführung der Befehle auf der Datenbank. Er akzeptiert Befehle von mehreren Prozessoren und verschachtelt die Ausführung der Befehle.

3.3.3 Schematypen der Referenzarchitektur

Im Nachfolgenden wird die Standard Drei-Level Schemaarchitektur von zentralisierten DBMS und die Fünf-Level Architektur erläutert, welche eine Erweiterung darstellt, um die Anforderungen Verteilung, Autonomie und Heterogenität erfüllen zu können.

3.3.3.1 ANSI/SPARC Drei-Level Schema Architektur

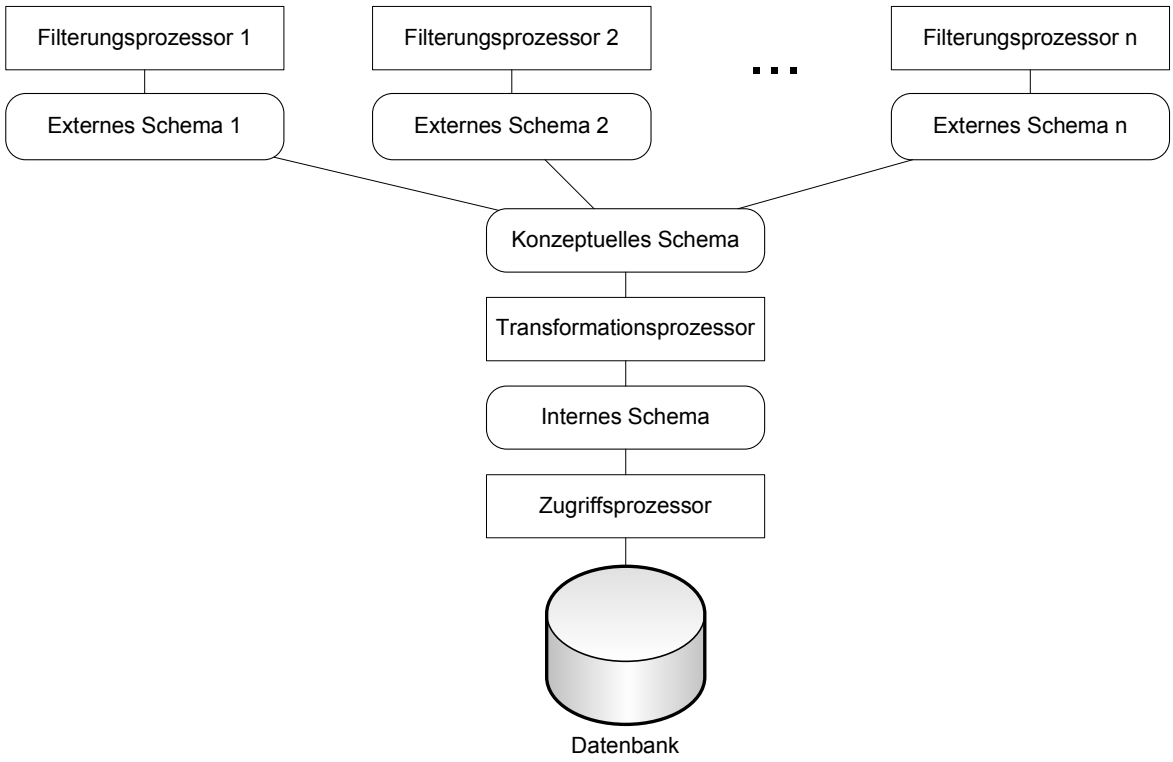

Abbildung 11: Systemarchitektur eines zentralisierten DBMS (nach [ShLa90]

Die drei Level sind konzeptuelles, internes und externes Schema (siehe Abbildung 11). Das konzeptuelle Schema beschreibt die konzeptuellen und logischen Datenstrukturen und die Beziehungen zwischen diesen Strukturen. Das interne Schema beschreibt physische Merkmale der logischen Datenstrukturen im konzeptuellen Schema. Das externe Schema ist ein Teil der Datenbank, welcher für einen bestimmten Benutzer bzw. eine bestimmte Benutzergruppe zugänglich ist, d.h. dass jeder Benutzer bzw. jede Benutzergruppe ihr eigenes externes Schema benötigt. Die drei Levels dienen dazu, dass Schemaänderungen auf unterster Ebene keine direkte Auswirkung auf die Benutzer haben. (vgl. [TsKl78] und [ShLa90]

3.3.3.2 Fünf-Level Schema Architektur für Föderierte Datenbanken

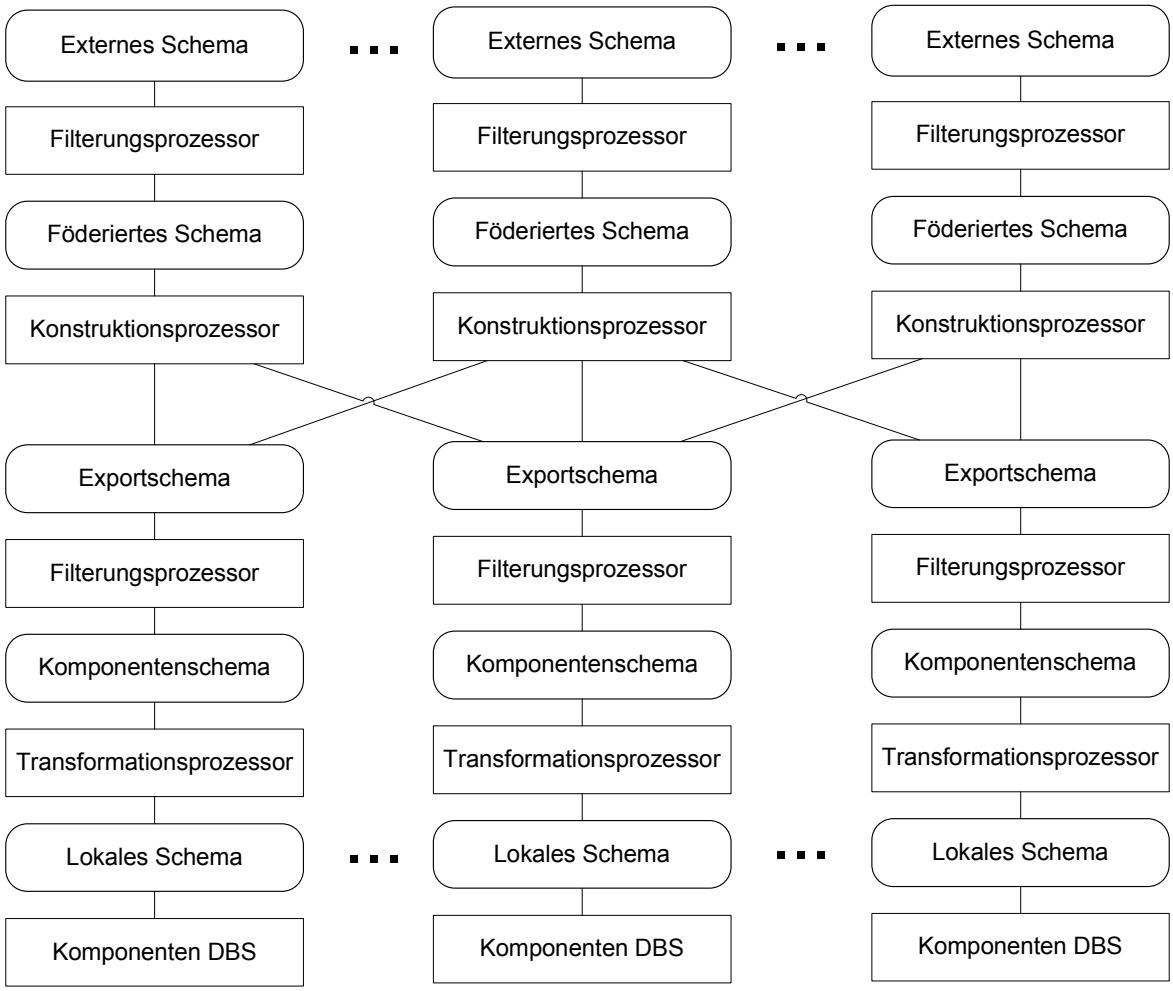

Abbildung 12: System Architektur eines FDBS (nach [ShLa90])

Um Verteilung, Heterogenität und Autonomie zu unterstützen wird das Drei-Level Schema erweitert. Die Fünf-Level Schema Architektur eines FDBS beinhaltet die Ebenen lokales, Komponenten-, Export- und föderiertes Schema (siehe Abbildung 12).

Das **lokale Schema** entspricht dem konzeptuellen Schema des Komponenten DBS.

Das **Komponentenschema** ist die Übersetzung des lokalen Schemas in ein „canonical“ oder „common data model“ (CDM) (vgl. [ShLa90]) genanntes Datenmodell. Das CDM beschreibt die lokalen Schemata, indem es eine einzige Repräsentation verwendet und im lokalen Schema fehlende Semantik im Komponentenschema hinzufügt. Der Prozess der Schemaübersetzung vom lokalen zum Komponentenschema generiert Mappings zwischen den

Komponentenschemaobjekten und den lokalen Schemaobjekten, welche vom Transformationsprozessor verwendet werden. Diese Transformationsprozessoren und das Komponentenschema unterstützen die Heterogenität von FDBS.

Das **Exportschema** repräsentiert eine Teilmenge des Komponentenschemas, welche für das FDBS zur Verfügung steht. Dies erfolgt deshalb, weil nicht alle Daten des Komponentenschemas für die Benutzer der Föderation zur Verfügung stehen sollen. Ein Filterungsprozessor kann verwendet werden, um die Zugriffskontrolle durch Beschränkung der möglichen Operationen auf das Komponentenschema zu gewährleisten. Dadurch wird die Autonomie von FDBS unterstützt.

Ein **föderiertes Schema** ist die Integration von mehreren Export Schemata. Es enthält außerdem die Information zur Datenverteilung, welche beim Integrieren der Exportschemata entsteht. Ein Konstruktionsprozessor transformiert die Befehle auf das föderierte Schema in ein oder mehrere Befehle auf die Exportschemata. Konstruktionsprozessoren und föderierte Schemata unterstützen die Verteilung von FDBS. Für jede Art von föderierten Benutzern (z.B. zentraler Einkauf) gibt es ein föderiertes Schema.

Das **externe Schema** definiert ein Schema für einen Benutzer und/oder eine Applikation bzw. eine Klasse von Benutzern/Applikationen. Ein externes Schema hat folgende Eigenschaften: Anpassung an die bereitgestellte Information; zusätzliche Integritätsbedingungen und Zugriffskontrolle. Ein Filterungsprozessor analysiert die Befehle, welche auf ein externes Schema abgesetzt werden und überprüft die Richtigkeit der Zugriffe per Zugriffskontrolle und Integritätsbedingungen.

[ShLa90]

3.4 Schemaintegration

In diesem Kapitel werden verschiedene Integrationsstrategien, Konflikte, welche bei der Schemaintegration auftreten können, und Konfliktlösungsansätze vorgestellt.

3.4.1 Integrationsstrategien

Bei der Integration von mehr als zwei lokalen Schemata ist eine Integrationsstrategie notwendig, um die unterschiedlichen Schemata zu einem Schema zusammenzuführen. Es können hier drei Arten unterschieden werden: One-Shot Integrationsstrategie, binäre Integrationsstrategie und n-äre Integrationsstrategie (vgl. [Conr97] und [BaLN86]).

3.4.1.1 One-Shot-Integrationsstrategie

Bei der One-Shot-Integrationsstrategie wird versucht alle Schemata in einem Schritt zu integrieren. Abbildung 13 zeigt die Integration von n Quellschemata in ein globales integriertes Schema.

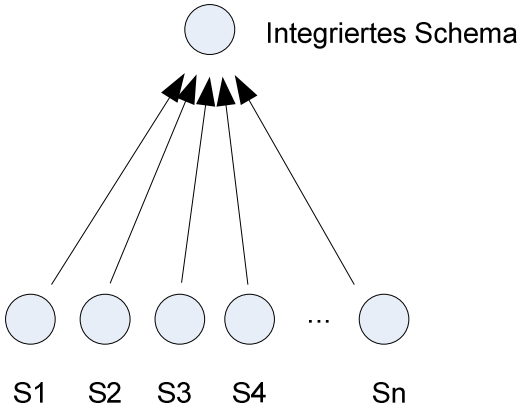

Abbildung 13: One-Shot-Integrationsstrategie (nach [Conr97])

3.4.1.2 Binäre Integrationsstrategie

Bei der Binären Integrationsstrategie werden immer nur genau zwei Schemata pro Integrationsschritt zusammengeführt. Dies wiederholt man solange bis alle Schemata zu einem Schema integriert wurden. Die Integration kann balanciert (siehe Abbildung 14) oder gewichtet (siehe Abbildung 15) erfolgen.

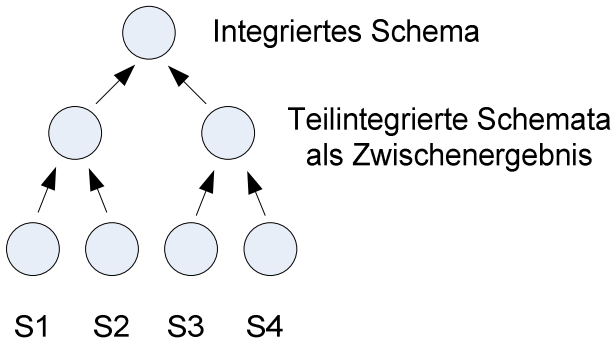

Abbildung 14: Balancierte binäre Integrationsstrategie (nach [Conr97])

Bei der balancierten Integrationsstrategie werden jeweils zwei Schemata auf der selben Stufe für die nächste Ebene zusammengefasst. Daraus ergeben sich teilintegrierte Schemata als Zwischenergebnis. Dieser Vorgang wird solange ausgeführt, bis nur mehr ein integriertes Schema übrig ist.

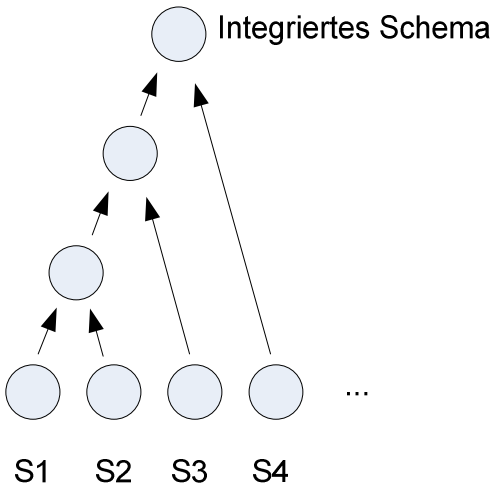

Abbildung 15: Gewichtete binäre Integrationsstrategie (nach [Conr97])

Bei der gewichteten binären Integrationsstrategie werden zu Beginn zwei Schemata zusammengeführt. Das so entstandene Schema wird mit dem nächsten Quellschema zusammengeführt. Dieser Vorgang wird solange fortgeführt bis alle Schemata zu einem integrierten Schema zusammengeführt wurden.

3.4.1.3 n-äre Integrationsstrategie

Bei der n-ären Integrationsstrategie wird im Unterschied zu den oben genannten Strategien bei jedem Integrationsschritt eine beliebige Anzahl von Schemata integriert. Dies ergibt eine Mischung aus One-Shot-Integrationsstrategie und binärer Integrationsstrategie (siehe Abbildung 16).

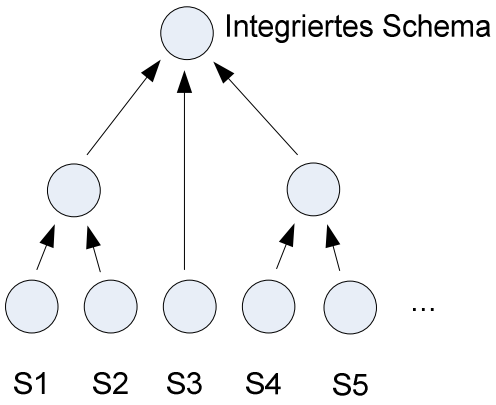

Abbildung 16: n-äre Integrationsstrategie (nach [Conr97])

3.4.2 Integrationsprozess

Der Integrationsprozess lässt sich in vier Phasen unterteilen (vgl. [Conr97], [BaLN86] und [SpPD92]):

1. **Vorintegration:**

In dieser Phase findet die Auswahl der Integrationsstrategie statt. Es können auch Präferenzen für einzelne lokale Schemata definiert werden.

2. **Vergleich der Schemata:**

In dieser Phase wird nach Gemeinsamkeiten und Konflikten (z.B. Namenskonflikt) der diversen Schemata gesucht.

3. **Anpassung der Schemata:**

In dieser Phase wird versucht, die gefundenen Konflikte durch Anpassung der Schemata zu lösen. Die Anpassung wird nicht auf der Ebene der lokalen Schemata durchgeführt, da dies zur Verletzung der Autonomie der Komponentensysteme führen würde.

4. **Zusammenführung und Restrukturierung:**

In der letzten Phase werden die Schemata mit Hilfe der geeigneten Strategie in ein Schema zusammengeführt.

3.4.3 Integrationskonflikte

Bei der Schemaintegration sollten die Kriterien Vollständigkeit, Korrektheit, Minimalität und Verständlichkeit eingehalten werden. Es treten bei der Integration allerdings Konflikte auf, die sich in die vier Gruppen semantische Konflikte, Beschreibungskonflikte, Heterogenitätskonflikte und strukturelle Konflikte einteilen lassen (vgl. [Conr97] und [SpPD92]).

3.4.3.1 Semantische Konflikte

Werden zwei unabhängig voneinander entstandene Schemata zusammengeführt, die die gleichen Objekte beschreiben, tritt eine semantische Überlappung auf. Dabei muss zwischen semantisch äquivalenten, sich einschließenden, überlappenden und disjunkten Objekten unterscheiden werden.

3.4.3.2 Beschreibungskonflikte

Selbst wenn in unterschiedlichen Schemata die gleichen Objekte der realen Welt modelliert werden, kann es zu einer unterschiedlichen Beschreibung der Objekte durch Attribute kommen. Zur Kategorie der Beschreibungskonflikte gehören Benennungskonflikte, Wertebereichskonflikte, Skalierungskonflikte und Konflikte durch Integritätsbedingungen bzw. Manipulationsoperationen.

3.4.3.3 Heterogenitätskonflikte

Bei der Integration eines relationalen und eines objektorientierten Schemas muss berücksichtigt werden, dass im relationalen Schema weniger Modellierungskonzepte angeboten werden. Bei der Integration von heterogenen Datenmodellen kommt es auch immer zu strukturellen Konflikten, da zur Modellierung desselben Sachverhalts unterschiedliche Modellierungskonzepte verwendet werden können.

3.4.3.4 Strukturelle Konflikte

Wird für die zu integrierenden Schemata dasselbe Datenmodell verwendet, kann es immer noch zu Konflikten kommen, wenn zur Beschreibung desselben Sachverhalts unterschiedliche Modellierungskonzepte verwendet werden. Der Modellierer hat z.B. beim objektorientierten Entwurf die Wahl, ob er eine Eigenschaft eines Objekts als wertbasiertes Attribut oder als Referenz auf ein eigenständiges Objekt modelliert.

3.4.3.5 Klassifikation von Konflikten zwischen relationalen Schemata

Alle Schemata sind in relationale Schemata überzuführen, um eine Integration zu erleichtern. Das weniger ausdrucksstarke Datenmodell wird als „kleinster gemeinsamer Nenner“ verwendet, sodass möglichst alles aus den lokalen Schemata abgebildet werden kann. Die nachfolgende Klassifikation orientiert sich an den Schemadefinitionsmöglichkeiten der relationalen Datenbanksprache SQL (vgl. [KiSe91]).

I. Schemakonflikte

A. Tabellen-Tabellen-Konflikte

1. eine Tabelle vs. eine Tabelle

a. Tabellennamenkonflikte

1) verschiedene Namen für gleiche Tabellen

2) gleiche Namen für verschiedene Tabellen

b. Tabellenstrukturkonflikte

1) fehlende Attribute

2) fehlende, aber implizite Attribute

c. Integritätsbedingungskonflikte

2. viele Tabellen vs. viele Tabellen

B. Attribut-Attribut-Konflikte

1. ein Attribut vs. ein Attribut

a. Attributnamenkonflikte

1) verschiedene Namen für gleiche Attribute

2) gleiche Namen für verschiedene Attribute

b. Default-Wert-Konflikte

c. Integritätsbedingungskonflikte

1) Datentypkonflikte

2) Bedingungskonflikte

2. viele Attribut vs. viele Attribute

C. Tabelle-Attribut-Konflikte

II. Datenkonflikte

A. falsche Daten

1. nicht korrekte Einträge

2. veraltete Daten

B. unterschiedliche Repräsentationen

1. verschiedene Ausdrücke

2. verschiedene Einheiten

3. unterschiedliche Genauigkeit

[KiSe91]

Da die Metadaten zu den Schemata für das „Visual Integration Tool“ in relationalen Datenbanken abgelegt sind, werden diese Konflikte in Abschnitt 4 berücksichtigt. Da nur Metadaten relevant sind, können Datenkonflikte für diese Arbeit ignoriert werden. Sie wurden nur zur Vollständigkeit angeführt.

3.4.4 Konfliktlösungsansätze

Die vier wichtigsten Konfliktlösungsansätze sind die Zusicherung, die Integration von Klassenhierarchien, die formalisierte objektorientierte Integration und das generische Integrationsmodell (vgl. [Conr97], [SpPD92] und [SpPa94])

3.4.4.1 Zusicherung

Bei der Zusicherung wird angegeben, welche Bestandteile der zu integrierenden Schemata einander entsprechen bzw. in einer Beziehung zueinander stehen, die für die Integration relevant ist. Die Formulierung der Zusicherungen muss unabhängig vom konkreten Datenmodell in Form von Inter-Schema-Korrespondenzen erfolgen. Einer der grundlegenden Aspekte dieses Ansatzes ist die automatische Auflösung von strukturellen Konflikten und die Berücksichtigung von Beziehungen im Integrationsprozess ohne Veränderungen der Ausgangsschemata.

Generisches Datenmodell (GDM):

Zur Formulierung der Inter-Schema-Korrespondenzen werden einige zentrale Modellierungskonzepte wie Objekte, wertbasierte Attribute und Referenzattribute verwendet, welche zusammen das generische Datenmodell bilden.

Korrespondenz-Zusicherung:

Bei der Korrespondenzzusicherung müssen zwei Arten unterschieden werden, die Zusicherung zwischen Schemaelementen und die Zusicherung von Korrespondenzen zwischen Pfaden und Links in Schemata.

- **Element-und-Attribut-Korrespondenz-Zusicherungen:**

  Diese Art der Zusicherung kann in zwei Typen (Element Korrespondenz Zusicherung und zugehörige Attribut Korrespondenz Zusicherung) unterteilt werden.

  - **Element-Korrespondenz-Zusicherung:**

    Es gibt folgende Möglichkeiten Korrespondenzen durch Zusicherungen zwischen den Schemaelementen X_1 (aus Schema S_1) und X_2 (aus Schema S_2) zu beschreiben:

    - $X_1 \equiv X_2$ (Äquivalenz)
    - $X_1 \supseteq X_2$ (Einschluss)
    - $X_1 \cap X_2$ (Überlappung)
    - $X_1 \neq X_2$ (Disjunktheit)

  - **Zugehörige-Attribut-Korrespondenz-Zusicherung:**

    Zusätzlich zu den Elementkorrespondenzen ist es sinnvoll Korrespondenzen der zugehörigen Attribute der in Beziehung stehenden Schemaelemente mit anzugeben. Hier gibt es wiederum Äquivalenz, Einschluss, Überlappung und Diskunktheit.

- **Pfad-Korrespondenz-Zusicherungen:**

  Eine Berücksichtigung der Beziehungen zwischen Schemaelementen ist neben der direkten Zuordnung von Schemaelementen eines Schemas zu Schemaelementen eines anderen Schemas ebenfalls wichtig. Im GDM gibt es Referenzattribute, mit welchen man Beziehungen zwischen Objekten darstellen kann. Zur Beschreibung eines Pfades verwendet man für das Schema S_1: $X_1 – X_2$ - … - X_n und für das Schema S_2: $Y_1 – Y_2$ - … - Y_n.

  - $X_1 – X_2$ - … - $X_n \equiv Y_1 – Y_2$ - … - Y_n (Pfadäquivalenz)

- $X_1 – X_2 - … - X_n \supseteq Y_1 – Y_2 - … - Y_n$ (Pfadeinschluss)
- $X_1 – X_2 - … - X_n \cap Y_1 – Y_2 - … - Y_n$ (Pfadüberlappung)
- $X_1 – X_2 - … - X_n \neq Y_1 – Y_2 - … - Y_n$ (Pfadausschluss)

Schemaintegration:

Die Integration der Schemata erfolgt mit Hilfe sogenannter Integrationsregeln. Die nachfolgenden Integrationsregeln sind für binäre Integration (siehe Kapitel 3.4.1.2) gültig. Ein zentrales Integrationsprinzip ist, dass immer die weniger eingeschränkte Struktur verwendet wird, wenn Integrationskonflikte auftreten. Nachfolgend werden fünf Integrationsregeln beschrieben (vgl. [Conr97] und [SpPD92]):

- **Integrationsregel 1 (Element ohne Gegenstück):**

  Diese Integrationsregel wird auf Schemaelemente angewendet, die nur in einem der beiden Schemata vorhanden sind. D.h. es gibt im zweiten Schema kein Gegenstück, welches Beziehung über eine Zusicherung beschreibt. Für jedes Element X_1 aus Schema S_1 mit fehlendem Gegenstück im Schema S_2 wird ein X vom selben Typ wie X_1 im integrierten Schema erzeugt. Das Gleiche erfolgt für jeden Link $X_1 – Y_1$ (Verbindung zwischen den Elementen) im Schema S_1 ohne Gegenstück im Schema S_2.

- **Integrationsregel 2 (äquivalente Elemente):**

  Diese Regel beschreibt die Behandlung von äquivalenten Objekttypen und den dazugehörigen Wertattributen. Im integrierten Schema wird ein Objekttyp X eingeführt, welcher den Namen von X_1 übernimmt, bzw. explizit anders gewählt wird. Mit einer „integrate-join Operation“ auf X_1 und X_2 wird die Struktur von X bestimmt. Dafür muss gelten: $X_1 \equiv X_2$ with corresponding attributes: $attcor_1(A_{11}, A_{21})$, …, $attcor_j(A_{1j}, A_{2j})$

- **Integrationsregel 3 (Links zwischen äquivalenten Elementen):**

  Diese Integrationsregel behandelt elementare Links (Attribut- und Beziehungs-Links) zwischen als äquivalent erkannten Schemaelementen. Dabei muss für die direkt miteinander verbundenen Elemente A_1 und B_1 im

Schema S_1 bzw. A_2 und B_2 im Schema S2 folgendes gelten: $A_1 \equiv A_2$, $B_1 \equiv B_2$ und A_1 - $B_1 \equiv A_2 – B_2$. Werden die Elemente A_1 und A_2 zu A und B_1 und B_2 zu B integriert, dann werden die Links A_1 - B_1 und $A_2 – B_2$ zu A – B integriert.

- **Integrationsregel 4 (Integration von Pfaden):**

  Diese Integrationsregel erlaubt die aus mehreren Links zusammengesetzte Integration von Pfaden. Dabei muss für die Elemente A_1, B_1, …, D_1 in Schema S_1 bzw. A_2, B_2, …, D_2 in Schema S_2 mit der Bedingung $A_1 \equiv A_2$ und $D_1 \equiv D_2$ folgendes gelten:

  - $A_1 – D_1 \equiv A_2 – B_2$ - … - D_2 → Pfad im integrierten Schema: A – B_2‘ - … D (B_2‘ ist integriertes Element zu B_2)
  - $A_1 – B_1$ - … - $D_1 \equiv A_2 – B_2$ - … - D_2 → Zwei Pfade im integrierten Schema: A – B_1‘ - … D und A – B_2‘ - … D

- **Integrationsregel 5 (Integration von Objekttyp und Wertattribut):**

  Diese Integrationsregel beschreibt die Integration von Objekttyp und Wertattribut bzw. komplexem Attribut. Für einen Objekttyp X_1 aus dem Schema S1 mit den Wertattributen (A_{11}, …, A_{1j}, B_1, …, B_k) und einem komplexen Attribut eines Elements E_2 aus dem Schema S_2 mit den Wertattributen (A_{21}, …, A_{2j}, C_1, …, C_h) als Komponenten mit einer Korrespondenzzusicherung $X_1 \equiv X_2$ mit zugehörigen Attributkorrespondenzen $attcor_1(A_{11}, A_{21})$, …, $attcor_j(A_{1j}, A_{2j})$ gilt: Ist E das mit E2 zusammenhängende Element im integrierten Schema, dann erhalten wir bei der Integration von X_1 und X_2 einen Objekttyp X und einen Beziehung-Link E – X mit folgenden Bedingungen:

  - Attribut X_2 von E_2 wird in ein Referenzattribut X_2‘ von X transformiert und die Kardinalitäten werden von X_2 auf X_2‘ übernommen
  - X erhält den Namen von X_1,außer es wird explizit ein anderer gewählt
  - X erhält seine Struktur durch eine integrate-join Operation von X_1 und X_2

Die oben angeführten Integrationsregeln sind vom Datenmodell unabhängig. (vgl. Conr97])

3.4.4.2 Integration von Klassenhierarchien

Die Integration von Klassenhierarchien spielt beim föderierten Datenbankentwurf eine große Rolle, weil die Vererbung eines der zentralen Konzepte bei objektorientierten Modellen ist. Die Struktur der vorliegenden Hierarchien in den lokalen Schemata soll auch im integrierten Schema enthalten sein. Dies bedeutet, dass Klassen aus zwei lokalen Schemata nur dann integriert werden können, wenn sie semantisch äquivalent sind. Sind sie nicht äquivalent, aber haben überlappende Objekte, dann muss für jede Klasse eine eigene Klasse im integrierten Schema angelegt werden. Gibt es zwischen den zwei Klassen keine Teilmengenbeziehung, wird mittels „Upward Inheritance“ (Aufwärtsvererbung) eine gemeinsame Oberklasse mit den Eigenschaften beider Klassen erstellt (vgl. [BuFN94], [Conr97] und [GasC95]).

3.4.4.3 Formalisierte objektorientierte Integration

Diese Integrationsmethode basiert auf einem eigenen Objektmodell, in dem es unter anderem folgenden Beziehungen zwischen den Objekttypen geben kann: Teilmengenbeziehung, Überlappung und Disjunktheit. Zu Beginn müssen die zu integrierenden Schemata in ein einheitliches Datenmodell gebracht werden. Der Ansatz basiert auf einer 4-Ebenen-Schema-Architektur mit den Ebenen lokales Schema, lokales Objektschema, globales Schema und globale Sicht (vgl. [Conr97], [RePG95] und [RPRG94]).

Auf die formale Repräsentation von Schemata und deren Integration wird in dieser Arbeit nicht näher eingegangen.

3.4.4.4 Generisches Integrationsmodell (GIM)

Das zentrale Anliegen dieser Integrationsmethode ist die Unabhängigkeit von konkreten Datenmodellen. Diese Unabhängigkeit wird durch eine spezielle Darstellungsform, die GIM-Notation, gewährleistet, mit Hilfe derer man integrierte, aber auch externe Schemata direkt ableiten kann. GIM erlaubt im Gegensatz zu den oben erklärten Ansätzen eine Umstrukturierung der Generalisierungs- und Spezialisierungshierarchien. Durch diese Art der Integration müssen neue Klassen

eingeführt werden, welche das in Kapitel 3.4.4.2 eingeführte Prinzip der „Upward Inheritance“ verwenden (vgl. [Conr97], [Schm95] und [ScSa96]).

GIM-Notation

In einer Notation, welche unabhängig von konkreten Datenmodellen ist, werden die zu integrierenden Schemata zusammen dargestellt. Die Notation setzt sich aus den in Abbildung 17 gezeigten Elementen zusammen. Die Menge von allen vorkommenden Objekten (disjunkte Extensionen) der zu integrierenden Schemata werden horizontal dargestellt. Die Attribute (Intentionen) werden vertikal dargestellt. Mit einem X wird markiert, welche Objekte welche Attribute besitzen. Sind in einem Bereich alle Objekte und Attribute mit einem X verbunden, kann dies als eine Klasse angesehen werden. (vgl. [Conr97] und [ScSa96]).

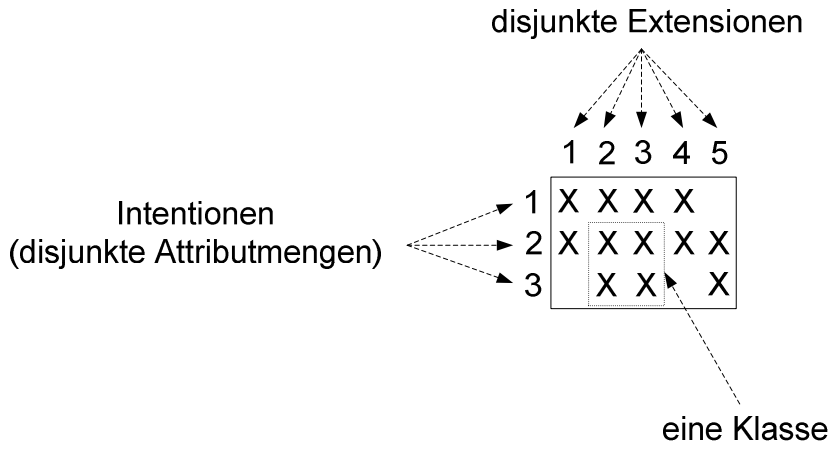

Abbildung 17: Notation von Schemainformation in GIM (nach [Schm95])

Integration mit GIM (vgl. [Conr97] und [Schm95]):

- **Extensionale Analyse:**

  In dieser Phase wird ermittelt, ob es Überlappungen zwischen den Extensionen der in den Schemata angegebenen Klassen gibt. Aufgrund der Analyse wird dann eine Zerlegung in disjunkte Klassen vorgenommen, bis es keine Überlappungen zwischen Klassen mehr gibt.

- **Intentionale Analyse und Zerlegung:**

  In dieser Phase wird für jedes in einem zu integrierenden Schema vorkommende Attribut festgestellt, zu welcher Klasse es für die jeweiligen

Objekte gehört. Gehören mehrere Attribute zu einer Klasse, können sie zu einer Menge zusammengefasst werden, welche als ein einzelner intentionaler Aspekt behandelt wird. Weiters müssen in dieser Phase auftretende Integrationskonflikte (z.B. durch Verwendung von Homonymen und Synonymen) durch Umbenennung behoben werden. Als Abschluss der ersten beiden Phasen wird eine GIM Matrix, auch normalisiertes Schema genannt, gebildet, in welcher alle Klassen und zugehörigen Attribute eingetragen sind.

- **Ableitung eines integrierten Schemas:**

  Aus dem zuvor gebildeten normalisierten Schema (GIM Matrix) wird nun ein integriertes Schema abgeleitet. Die Ableitung erfolgt automatisiert anhand des folgenden Algorithmus.

  - Zuerst wird die GIM Matrix sortiert, damit die am häufigsten ausgefüllten Zeilen und Spalten links oben beginnen.
  - Aus der sortierten GIM Matrix können nur Spezialisierungshierarchien ausgelesen werden. Damit können dann Klassen und Unterklassenbeziehungen gebildet werden.

- **Ableitung externer Schemata:**

  Die Ableitung des externen Schemas erfolgt analog zu der des integrierten Schemas.

3.5 Fazit

In Kapitel 3 wurde erläutert, wie man mehrere Data Warehouses zu einem System föderieren kann. Dabei wurde auf diverse Architekturen und Integrationsmethoden eingegangen. Die hier gewonnen Erkenntnisse fließen in die Entwicklung des „Visual Integration Tool“ ein, welches in Abschnitt 4 näher erläutert wird.

4 Common Warehouse Metamodel

Das „Common Warehouse Metamodel“ (CWM) ist ein von der Object Management Group (OMG) entwickelter Standard zum Austausch von Metadaten für Data Warehousing. Wie der Austausch funktioniert, wird in diesem Kapitel beschrieben. CWM bietet eine allgemeine Sprache zur Beschreibung von Metadaten und die Möglichkeit zum XML-basierten Austausch dieser Metadaten.

Mit Hilfe eines Industriestandards zum Austausch von Metadaten, welcher durch CWM gewährleistet wird, haben die Hersteller die Möglichkeit, kompatible Datenbanken, Tools und Applikationen zu entwickeln. Vor der Einführung des Standards war es notwendig Schnittstellen (Brücken) zwischen allen Beteiligten Systemen zu bauen, um einen Datenaustausch zu gewährleisten. Grundlegende von OMG eingeführte Standards waren UML (Unified Modeling Language) und MOF (Meta Object Facility) 1997 und XMI (XML Metadata Interchange) 1999.

Historische Entwicklung:

Während der Entwicklung von CWM gab es andere Industriestandards. Im Jahre 1993 veröffentlichte die Electronics Information Group ihren Standard CASE Data Interchange Format (CDIF) als Standard für Metadaten, welche von CASE Tools erzeugt wurden. Im Oktober 1995 wurde die Meta Data Coalition (MDC) von führenden IT Firmen gegründet. Im April 1996 veröffentlichte die MDC die Meta Data Interchange Specification (MDIS). Währenddessen entwickelte Microsoft das Open Information Model (OIM). Im September 1998 veröffentlichte die OMG ihren „Request for Proposal (RFP) for a Common Warehouse Meta Data Interchange“. Im Dezember 1998 ist Microsoft der MDC beigetreten und übergab OIM der MDC, damit OIM weiter als offener Standard weiterentwickelt wurde. Im Juli 1999 wurde OIM 1.0 veröffentlicht. Bei der Veröffentlichung von CWM im September 1999 zeigte sich eine Inkompatibilität vom OIM und CWM. Wegen der Probleme, die mehrere konkurrierende Standards mit sich bringen, unterstützte CWM bei Überschneidungen von CWM und OIM alle Möglichkeiten von OIM. Dies führte im Juni 2000 zu einer Veröffentlichung der zweiten Version von CWM durch die OMG. Die MDC Mitglieder stimmten im September 2000 dafür, OIM nicht weiter zu unterstützen und gaben CWM den Vorzug (vgl. [PCTM02] und [PCTM03]).

4.1 Information Supply Chain

Das typische Data Warehouse bzw. die typische Geschäftsanalyseumgebung wird als Information Supply Chain (ISC) beschrieben. Dabei wird die Produktion von Information als eigenständiger Geschäftsprozess angesehen. Normalerweise zeigt eine Supply Chain die Herstellung und Distribution von diversen Gütern, die in einzelne Phasen (z.B. Beschaffung, Produktion, Vertrieb) aufgeteilt ist. Abbildung 18 zeigt die typische ISC, welche den gesamten Informationsfluss von der Datenquelle bis hin zur Informationsverwertung für das Gut Information darstellt (vgl. [PCTM02] und [PCTM03]).

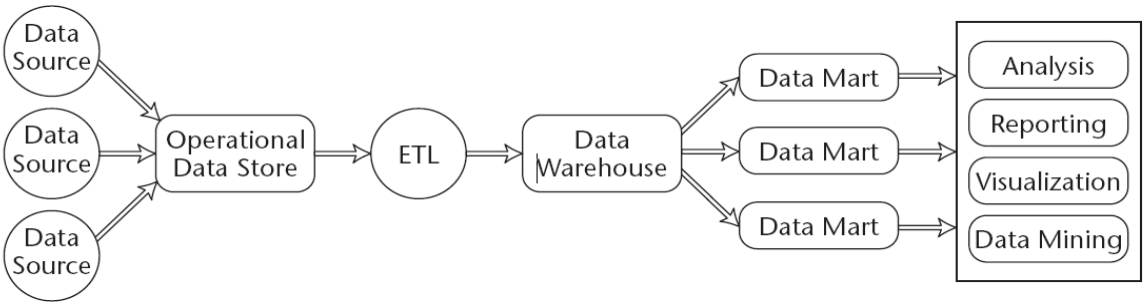

Abbildung 18: Information Supply Chain (Quelle [PCTM03])

Zuerst werden die Rohdaten aus den Datenquellen ausgelesen und im Operational Data Store bereit gestellt. Mit Hilfe des ETL (Extraction, Transformation and Loading) Prozesses werden die Daten für das Data Warehouse aufbereitet. Aus dem Data Warehouse können mehrere Data Marts für nachfolgende Analysen gebildet werden. Die Hauptproblematik dabei ist, dass die Tools für die diversen Phasen oft von unterschiedlichen Herstellern stammen und der Austausch der Metadaten zwischen den Phasen oft mit sehr hohen Kosten verbunden ist. Um die einzelnen Phasen miteinander zu verbinden, werden Brücken zwischen den Phasen eingerichtet (siehe Abbildung 19).

(vgl. [PCTM02] und [PCTM03])

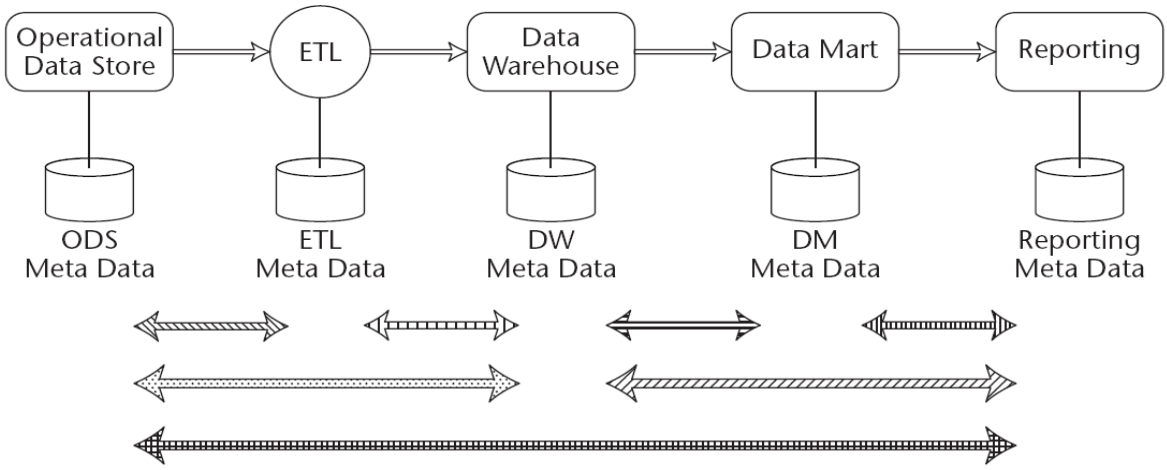

Abbildung 19: Metadatenintegration mit Point-to-Point Brücken (Quelle [PCTM03])

Diese Point-to-Point Verbindung mittels Brücken ist komplex und deshalb mit hohen Kosten verbunden, da für jedes zusätzliche Element Brücken zu allen anderen errichtet werden müssen. Eine Lösung dieser Problematik ist die Einführung eines zentralen Metadatenrepository, wie in Abbildung 20 gezeigt. Hierbei muss nur mehr eine Brücke pro Phase zum Repository erstellt werden und nicht zu jeder anderen Phase. Das Repository übernimmt die Aufgabe der Metadatenspeicherung, die Kontrolle und stellt die Metadaten zur Verfügung (vgl. [PCTM02] und [PCTM03]).

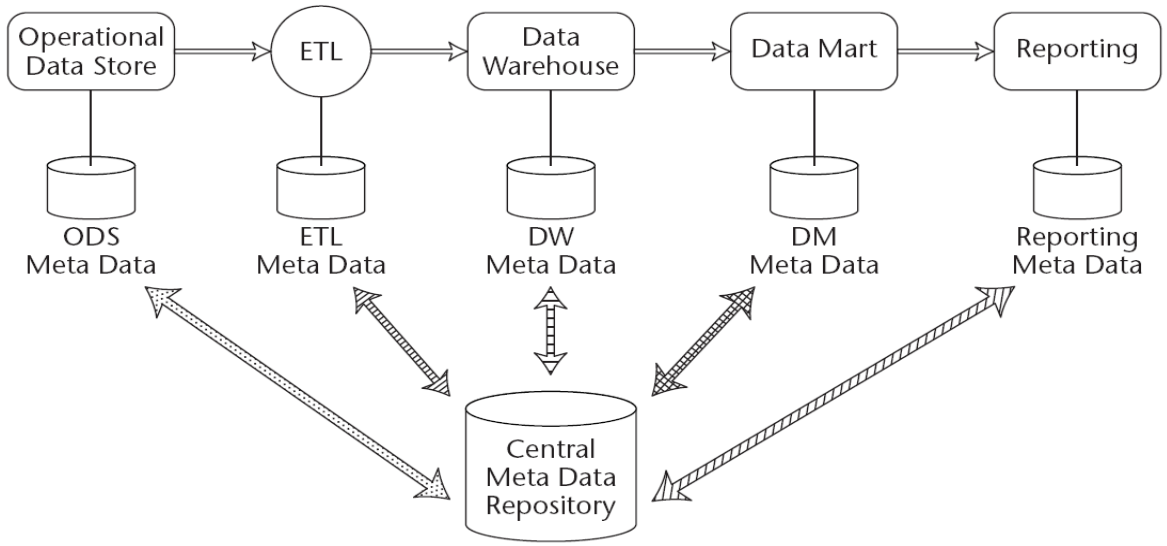

Abbildung 20: Metadaten Integration mit zentralisiertem Repository (Quelle [PCTM03])

CWM versucht basierend auf der Modellierung der Data Warehousing-Infrastruktur und mittels der Model Driven Architektur (MDA) eine Lösung zu finden, die die Anzahl der zu bildenden Brücken reduziert. CWM bietet eine kosteneffektive

Integrationsmöglichkeit mittels zentralem Metadatenrepository (vgl. [PCTM02] und [PCTM03]).

4.2 Modellbasierter Ansatz

Abbildung 21 zeigt die Verwendung des CWM Metamodells bei einer Point-to-Point-Metadatenarchitektur. Zwischen den einzelnen Phasen müssen keine teuren Brücken installiert werden, da diese alle CWM konform sind und somit ein Metadatenaustausch möglich ist. Zusätzlich gibt es noch die Variante, die CWM Metadaten in einem Metadatenrepository zu speichern (vgl. [PCTM02] und [PCTM03]).

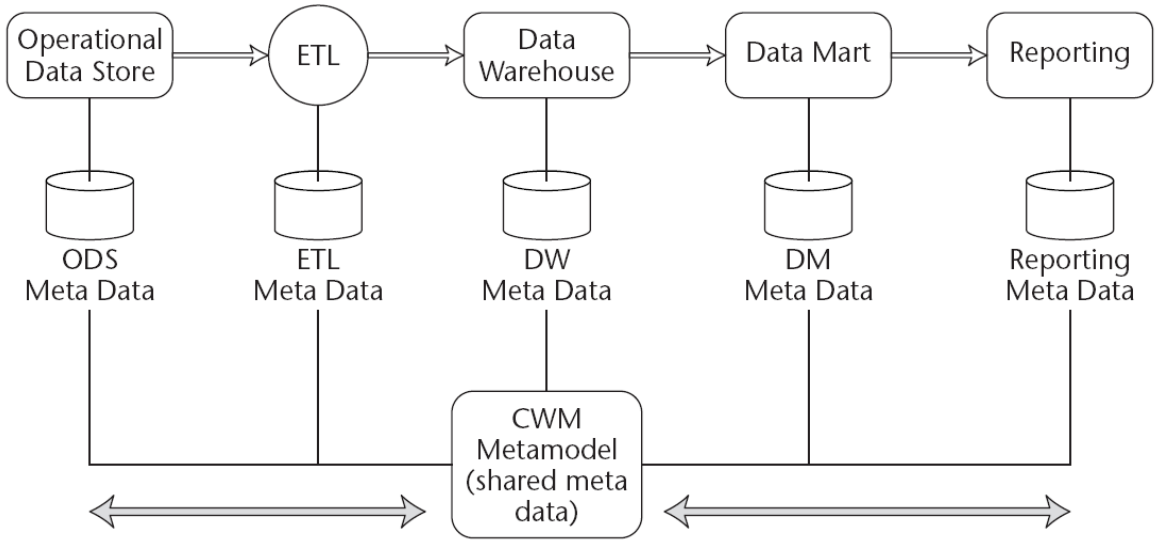

Abbildung 21: Modellbasierte Point-to-Point-Metadatenarchitektur (Quelle [PCTM03])

Eine komplett modellbasierte Metadatenintegrationslösung besteht normalerweise aus folgenden Komponenten [PCTM03]:

1. Eine formale Sprache, mit welcher es möglich ist, für gemeinsame plattformunabhängige Modelle Metadaten zu spezifizieren
2. Ein allgemeines Metamodell, welches den Problembereich beschreibt
3. Ein allgemeines Austauschformat zum Austausch von gemeinsamen Metadaten
4. Ein allgemeines Programmierinterface für den Metadatenzugriff
5. Standardmechanismen zum Erweitern der Modelle
6. Standardmechanismen zum Erweitern des Metamodells
7. Softwareadapter, die den produktmäßigen Metadatenimport und -export erleichtern

8. Ein zentrales Metadatenrepository
9. Eine globale Metadatenmanagementstrategie
10. Eine globale technische Metadatenarchitektur

4.3 Foundation Technologies

Das Ziel von CWM ist es, das Austauschformat von Metadaten für Data Warehousing und Geschäftsprozessanalyse zu standardisieren. Die Standardisierung erfolgt in drei Schritten. Zuerst muss das Modell für die gemeinsamen CWM Metadaten mit UML (Unified Modelling Language) definiert werden. Anschließend wird die Spezifikation für das CWM Austauschformat in XML (Extensible Markup Language) generiert. Mittels MOF (Meta Object Facility) werden die Daten verwaltet und ausgetauscht. Als letzter Schritt wird die Spezifikation der Programmiersprachen API in CORBA IDL (Interface Definition Language) generiert. Als Implementierungsplattform hat sich Java durchgesetzt (vgl. [PCTM02] und [PCTM03]).

4.3.1 Extensible Markup Language (XML)

XML ist eine Teilmenge von SGML (Standard Generalized Markup Language). XML beschreibt eine Klasse von Objekten, welche XML Dokumente genannt werden und beschreibt teilweise das Verhalten von Computerprogrammen, welche diese verarbeiten (vgl. [PCTM02], [PCTM03] und [W3CR10]).

4.3.2 Meta Object Facility (MOF)

MOF ist ein modellgetriebenes Objektframework für die Spezifikation, die Konstruktion, die Verwaltung, den Austausch und die Integration von Daten in Softwaresystemen. Das MOF benutzt eine Vier-Ebenen Metadatenarchitektur, die sogenannte OMG Meta Data Architecture. Die unterste Ebene M0 beinhaltet Objektdaten aus modellierten Systemen (Warehousedaten). Die darüberliegende Ebene M1 beinhaltet das Model mit Metadaten, welches als UML Model (Warehousemetadaten) dargestellt wird. M2 beinhaltet das Metamodel mit Meta-Metadaten, welche als UML Metamodel (CWM Metamodel) dargestellt werden. Die oberste Schicht, M3, beinhaltet das Meta-Metamodel, welche als MOF Model dargestellt wird. MOF M3 wird basierend auf sich selbst definiert (Rekursionsanker). CWM 1.0 basiert auf MOF 1.3, welches von OMG im September 1999 übernommen wurde. (vgl. [PCTM02] und [PCTM03]).

4.3.3 XML Meta Data Interchange (XMI)

XMI ist eine XML Sprache zum Austausch von Metadaten in Softwaresystemen. XMI integriert die drei Standards UML, MOF und XML. UML definiert eine umfangreiche, objektorientierte Modellierungssprache, welche durch eine graphische Notation unterstützt wird. XML bietet ein universelles Format zum Austausch von Metadaten (und Daten). MOF definiert ein erweiterbares Framework zur Definition von Metadaten unter der Verwendung von UML und bietet Werkzeuge mit Interfaces zur Erstellung, zum Updaten, zum Zugriff und zum Navigieren von Metadaten. XML erlaubt MOF Metadaten den Austausch über Streams oder Dateien mit einem auf XML basierten Standardformat. CWM 1.0 basiert auf XMI 1.1, welches im Oktober 1999 von OMG übernommen wurde (vgl. [PCTM02] und [PCTM03]).

4.4 Architektur

Der Kern von CWM ist eine Menge von Erweiterungen der OMG Metamodellarchitektur (MOF), welche diese an die Bedürfnisse der Domäne Data Warehousing und Geschäftsanalyse anpassen. CWM hat eine modulare Architektur, die auf einer objektorientierten Basis von MOF aufbaut. Abbildung 22 zeigt die einzelnen Pakete des CWM Metamodells. Dieses besteht aus 21 Paketen, welche in fünf Schichten unterteilt sind. Die einzelnen Komponenten bauen teilweise aufeinander auf. So ist z.B. das Core Paket Teil aller anderen Pakete. Die oberen Schichten greifen immer auf darunterliegende zurück, d.h. die Superklassen der unteren Schicht vererben Funktionalität auf die darüber liegenden. (vgl. [PCTM02] und [PCTM03]).

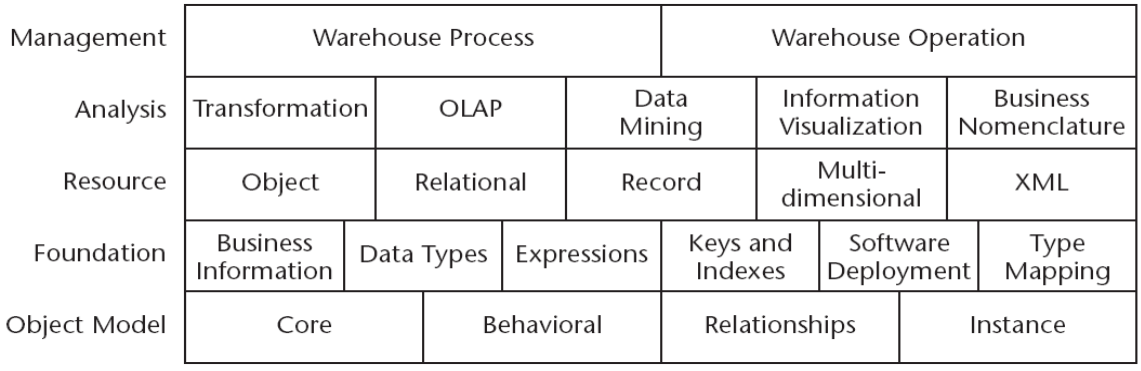

Abbildung 22: CWM Metamodel Packages (Quelle: [PCTM03])

4.5 Fazit

CWM bietet einen Standard im Data Warehousing Bereich, welcher als Grundlage für die Entwicklung des „Visual Integration Tools“ und des damit verbundenen Metadaten Repository (siehe Abbildung 1) verwendet wird. Das Metadaten Repository basiert auf den in Abbildung 22 vorgestellten CWM Metamodel Packages, welche in Abschnitt 4 näher erläutert werden.

Abschnitt 3
Ansätze der visuellen Datenintegration

5 Ansätze der visuellen Datenintegration

In diesem Kapitel werden Ansätze zur Visualisierung der Datenintegration näher vorgestellt, welche als Anregung für die Entwicklung des „Visual Integration Tools“ dienen. Weiters wird auf das Produkt MapForce eingegangen, um ein Beispiel eines kommerziellen Produkts zur Datenintegration vorzustellen.

5.1 Multi-dimensionale UML Package Diagrams

Der Ansatz Multi-dimensionaler UML-Package-Diagrams befasst sich mit multidimensionalen (MD) Modellen für Data Warehousing, welche mit Hilfe von UML abgebildet werden. Dabei werden nicht nur UML Klassendiagramme, sondern auch zusätzliche UML Erweiterungen verwendet, die in einem UML-Profil zusammen gefasst sind. Diese Erweiterungen beinhalten Stereotypes zur Darstellung von Paketen für Fakten, Dimensionen und ganzer Star Schemata (siehe Abbildung 23). Diese Pakete dienen zur grafischen Repräsentation von MD Modellen und beinhalten Constraints (z.B. darf ein Star Package nur ein Fact Package beinhalten). [LuTI02]

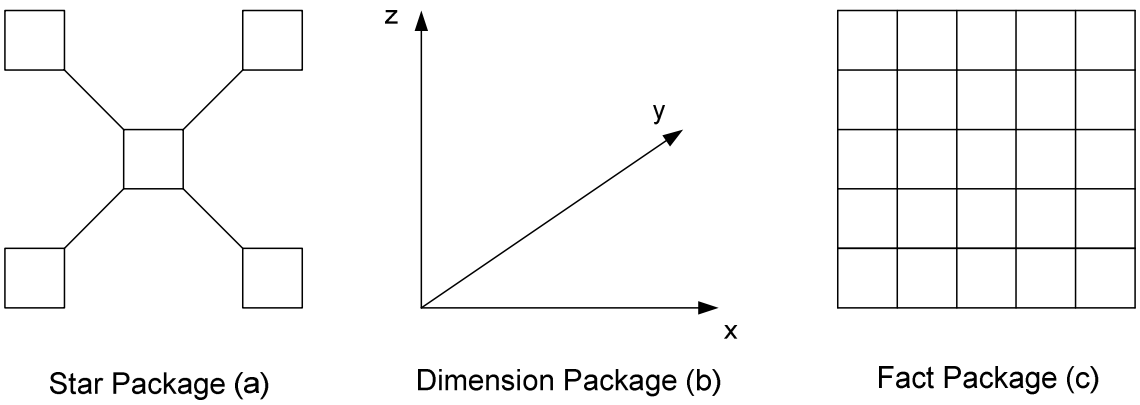

Abbildung 23: Stereotype Icons (nach [LuTI02])

Der Designprozess kann in drei Abstraktionslevels unterteilt werden. Diese Schachtelung der Pakete kann mit einer Klassenhierarchie bei der objektorientierten Programmierung verglichen werden.

Es gilt folgende Beziehung zwischen den Objekten:

- 1 Star-Schema beinhaltet n Dimension und 1 Fakt
- 1 Dimension beinhaltet n Dimension Level

Abbildung 24 zeigt ein Beispiel für die Schachtelung eines Star-Schemas. Star-Schema 1 Paket beinhaltet zwei Dimension Pakete und ein Fakt Paket. Das Dimension 2 Paket beinhaltet mehrere Dimension Level.

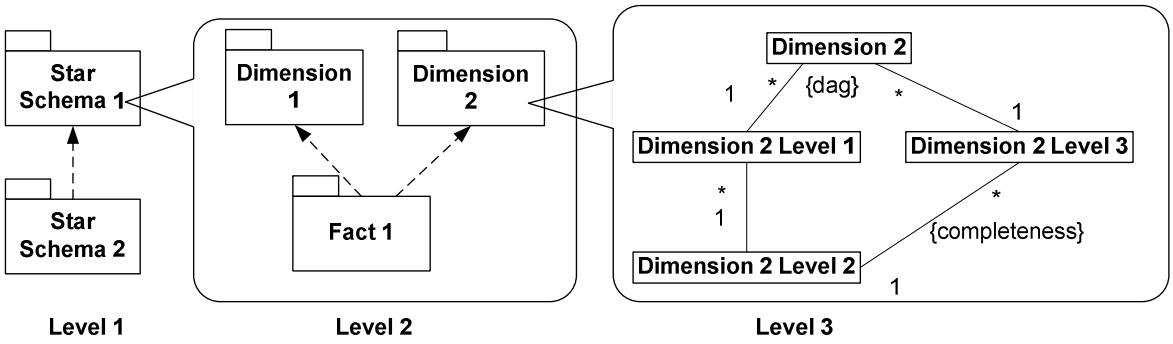

Abbildung 24: Die drei Ebenen des MD Modells (nach [LuTI02])

Level 1 – Modelldefinition:

Bei UML-Package-Diagrammen werden wie beim VIT nur Star-Schemata berücksichtigt. Ein Paket repräsentiert ein Star Schema eines konzeptuellen MD Modells. Eine Abhängigkeit zwischen zwei Paketen auf diesem Level bedeutet, dass die Star Schemata mindestens eine Dimension gemeinsam haben (siehe Abbildung 25).

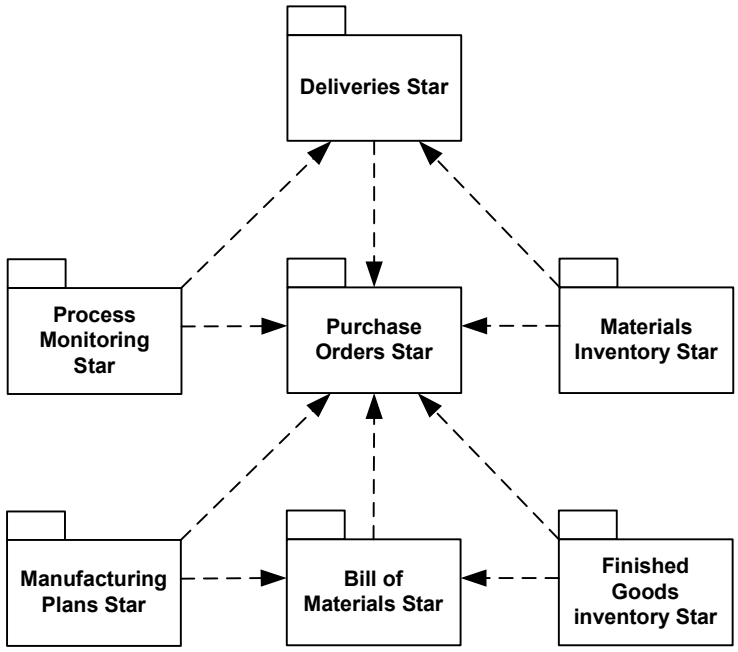

Abbildung 25: Level 1: Star Schemata (nach [LuTI02])

Level 2 – Star Schema Definition:

Im zweiten Level erfolgt die Darstellung der Dimensionen und Fakten (siehe Abbildung 26). Ein Paket repräsentiert ein Fakt oder eine Dimension eines Star Schemas.

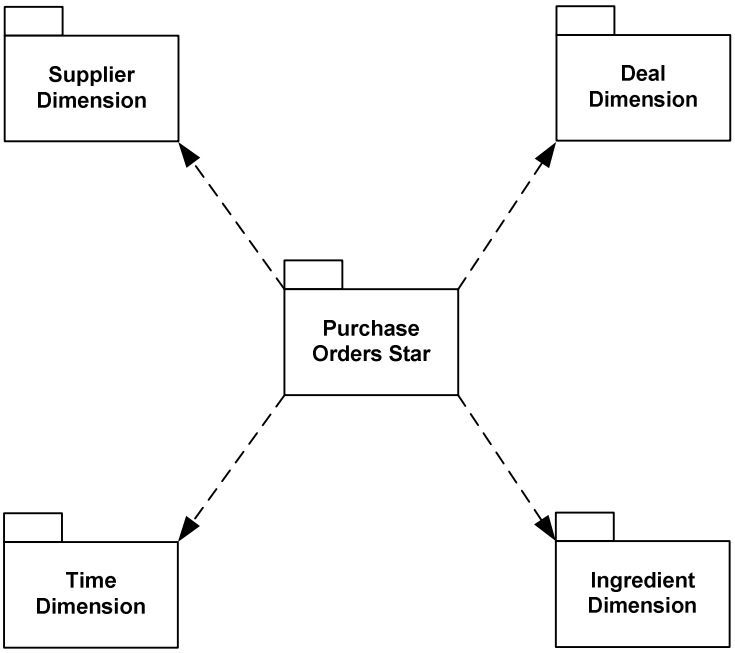

Abbildung 26: Level 2: Fakt Schema (nach [LuTI02])

Level 3 – Dimension/Fakt Definition:

Das Paket beinhaltet eine Menge von Klassen, welche das Fakt Paket und die Hierarchielevel eines Dimension Pakets repräsentieren (siehe Abbildung 27). Im dritten Level wurde die Supplier Dimension aus Abbildung 26 verwendet.

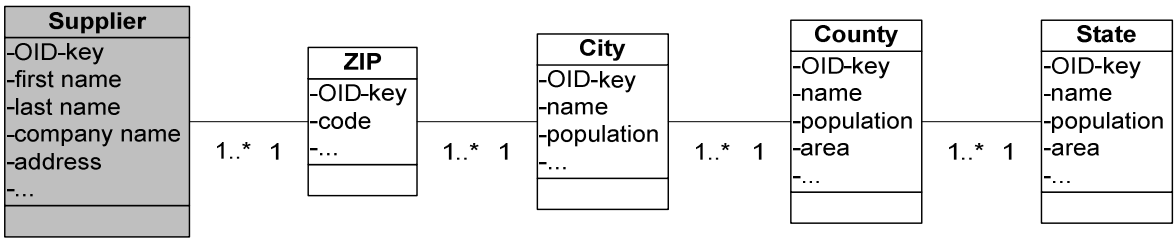

Abbildung 27: Level 3: Dimension (nach [LuTI02])

Das UML Package Diagram wurde als Plugin für Rational Rose realisiert (vgl. [LuTI02]). In Kapitel 6.2 wird erläutert, ob Rational Rose auch für das VIT in Frage kommt.

5.1.1 Vor- und Nachteile

Vorteile:

- Reduzierung der Komplexität von UML Diagrammen
- Übersichtlichkeit aufgrund der Gruppierung in Packages für Star-Schemata, Fakten, Dimensionen

Nachteile:

- Fehlende Information über Anzahl der einzelnen Elemente innerhalb eines Packages (z.B. Dimension)

5.2 Data Mapping Diagrams

Beim Data Warehousing sind ETL Prozesse für die Extraktion von Daten aus heterogenen operationalen Datenquellen, ihre Transformation und das Laden ins Data Warehouse (DW) verantwortlich. Die Data Mapping Diagrams, welche in diesem Kapitel beschrieben werden, bieten ein Framework für das Design von „DW back-stage“ auf sehr niedriger Granularität, welches Transformationsregeln auf Attributlevel beinhalten muss. Dieses Framework ermöglicht die Modellierung von Beziehungen zwischen Quelle und Ziel auf unterschiedlichem Detail-Level. Der Ansatz dieser Methode basiert auf der Erweiterung von UML (Unified Modeling Language) mit dem „MD-Profil“ (siehe 5.1) und erlaubt damit das hinein und hinaus zoomen in multi-dimensonalen UML Diagrammen eines Star-Schemas. [LuVT04]

Die „Data Mapping Diagrams“ bieten die Möglichkeit Mappings zwischen unterschiedlichen Data Warehouse Schemata grafisch darzustellen. Bei der Darstellung werden unterschiedliche Stufen unterschieden. Das Framework lässt sich in fünf Phasen, drei Abstraktions-Levels und unterschiedliche Diagramme einteilen [LuVT04]:

Phasen

Die Phasen beschreiben die einzelnen Vorgänge, welche von der Quelle bis zum Ziel durchlaufen werden:

- Quelle: definiert die Datenquellen eines DWs, wie OLTP System, externe Datenquellen, etc.
- Integration: definiert die Mappings zwischen Datenquelle und DW
- Data Warehouse: definiert die DW Struktur
- Anpassung: definiert die Mappings zwischen DW und der Client Struktur
- Client: definiert spezielle Strukturen, wie Data Marts oder OLAP Anwendungen, welche von Clients benutzt werden um auf das DW zuzugreifen

Abstraktions-Levels

- Konzeptuell: definiert das DW aus einer konzeptuellen Sicht
- Logisch: definiert logische Aspekte eines DW, wie die Definition des ETL Prozesses
- Physisch: definiert die physischen Aspekte eines DW, wie die Speicherung der logischen Strukturen auf unterschiedlichen Datenträgern oder die Konfiguration der Datenbank Server, welche das DW unterstützen

Diagramme

Jede Phase und jedes Abstraktions-Level benötigt eigene Modellierungsformalismen. Die Data Mapping Diagrams unterstützen 15 Diagrammarten, welche hier allerdings nicht näher erläutert werden (vgl. [LuVT04]).

Abbildung 28 zeigt die unterschiedlichen Mapping Levels und ihr Zusammenspiel.

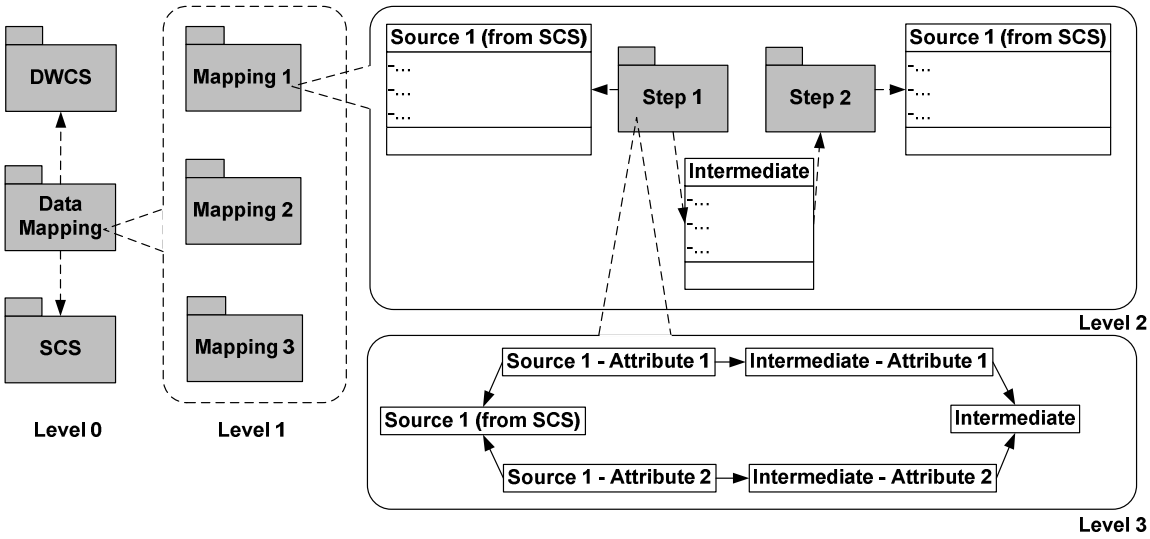

Abbildung 28: Data mapping levels (nach [LuVT04])

Database Level (Level 0):

In diesem Level wird jedes Schema des DW (Datenquelle auf konzeptuellen Level (SCS - source conceptual schema) und konzeptuelles Schema des DW (DWCS – data warehouse conceptual schema)) als ein Paket repräsentiert. Die Mappings zwischen diversen Schemata sind als Mappingpaket modelliert, in welchem alle anderen Mappings gekapselt sind.

Dataflow Level (Level 1):

Dieser Level beschreibt die Beziehung der Daten zwischen individuellen Quelltabellen von involvierten Schemata zu Zielen im DW.

Table Level (Level 2):

Wo Mappingdiagramme des Dataflow Levels Beziehungen zwischen Quelle und Ziel als ein einzelnes Paket beschreiben, beschreibt das Diagramm auf dem Table Level alle Zwischentransformationen.

Attribute Level (Level 3):

In diesem Level werden alle Inter-Attribute-Mappings dargestellt. Dieses Diagramm zeigt die Mappings zwischen Quelle und „Intermediate“ bzw. „Intermediate“ und Ziel.

[LuVT04]

5.2.1 Vor- und Nachteile

Vorteile:

- Theoretischer Ansatz zum Mappen von Data Warehouse Schemata
- Trennung unterschiedlicher Stufen der Granularität

Nachteile:

- Unübersichtlichkeit bei mehreren Quell - Data Warehouses
- Fehlende Information über Anzahl der einzelnen Elemente innerhalb eines „Levels“ (z.B. Dimension)

5.3 MapForce

MapForce bietet ein graphisches Interface, welches den Austausch von Daten zwischen Content-Modellen und Formaten vereinfacht. Die Daten können ebenfalls manipuliert werden, indem Transformationen verwendet werden, welche die Ausgabe verändern. Bei der Verwendung von MapForce werden die Quell- und Zielinformationen (Metadaten) geöffnet, Datenverarbeitungsfunktionen von den erweiterbaren Bibliotheken hinzugefügt und Transformationen mittels drag&drop erstellt. Danach kann die transformierte Ausgabe sofort angesehen bzw. gespeichert werden. MapForce bietet außerdem die Möglichkeit, Transformations-Stylesheets oder Programmcode zu generieren, welcher für serverseitige Datenintegration und Webservice-Applikationen verwendet werden kann. [Alto07]

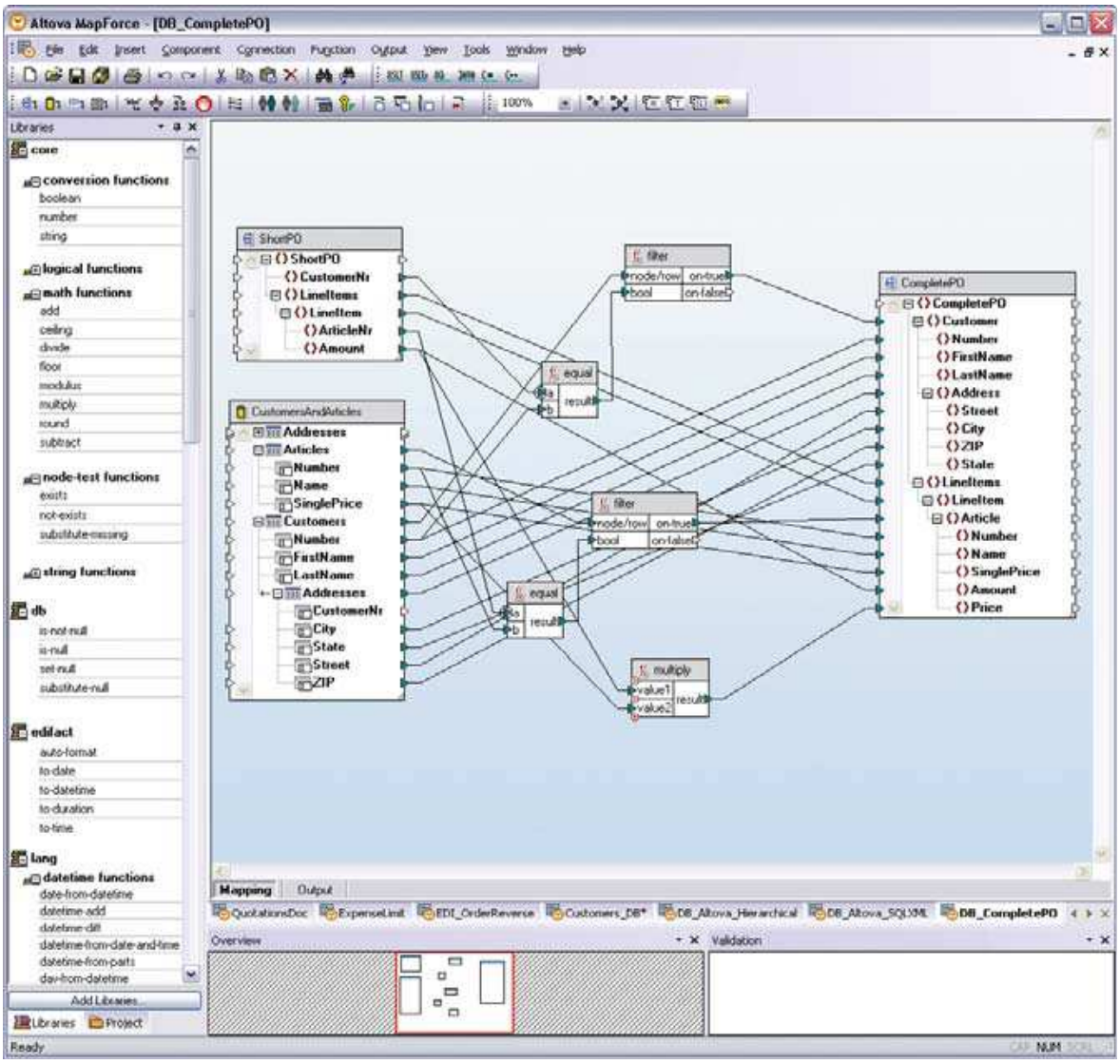

Abbildung 29: MapForce Mapping (Quelle [Alto07])

5.3.1 Vor- und Nachteile

Vorteile:

- Ausgereiftes Tool zum Erstellen von Mappings
- Anbindung unterschiedlichster Datenquellen
- Operationen beim Mapping durchführbar (z.B. Umbenennen)

Nachteile:

- Unübersichtlichkeit bei komplexen Datenstrukturen
- Kein Data Warehouse spezifisches Mapping
- Kostenpflichtig

5.4 Global Schema Architect

Das Tool Global Schema Architect von Lorenz Maislinger [Mais09] verfolgt einen ähnlichen Ansatz wie das VIT. Das Tool wurde ebenfalls als Teil eines gesamten föderierten Systems (vgl. Abbildung 1) entwickelt und dient zur Definition von Mappings zwischen lokalen Schemata und globalem Schema. Der Fokus beim Global Schema Architect liegt aber in der Anbindung von Data Marts (vgl. Kapitel 2.3). Das Tool basiert auf einem Plugin für Eclipse, über welches die Darstellung der Schemata und die Durchführung des Mapping-Vorgangs erfolgt.

Das Tool gliedert die Funktionen in folgende Bereiche [Mais09]:

- Erzeugen eines neuen globalen Schemas
- Importieren eines Datamarts
- Erstellen der Import-Mappings
- Erstellen der Global-Mappings
- Export der SQL-MDi Datei
- Export der Metadaten

5.4.1 Vor- und Nachteile

Vorteile:

- Data Mart spezifisches Mapping
- SQL-MDi Export

Nachteile:

- Unterstützt nur Oracle Datenbanken
- Keine Darstellung von globalem Schema und lokalen Schemata in einem Editor

5.5 Fazit

Die in diesem Kapitel beschriebenen Ansätze zeigen Möglichkeiten, wie man das Mapping für das visuelle Integrationstool (VIT) darstellen kann.

Die multi-dimensionalen UML Package Diagrams bieten eine reine Darstellungsmöglichkeit von Star-Schemata mittels UML bzw. Erweiterungen von

UML Diagrammen. Sie beinhalten keinerlei Information bzgl. Mappings zwischen zwei Schemata. Für die Entwicklung des VIT wird dieser Ansatz als Input für eine Darstellungsmöglichkeit der unterschiedlichen Schemata verwendet. Der Nachteil der UML Package Diagrams ist allerdings die fehlende Information über die Komplexität innerhalb einzelner Packages. Dadurch wird ein Vergleich eines lokalen und globalen Schemas, welcher mittels VIT durchgeführt wird, erschwert. Die Gruppierung von Dimensionen und Fakten bei der grafischen Darstellung wurde bei der Implementierung des VIT berücksichtigt.

Die Data Mapping Diagrams haben eindeutig einen Fokus auf der Darstellung von Mappings zwischen Data Warehouse Schemata. Dadurch kann kein direkter Vergleich mit den UML Package Diagrams angestellt werden. Auch bei den Data Mapping Diagrams ist die fehlende Information über die Komplexität des einzelnen Levels als Nachteil anzuführen. Als Input für die Entwicklung des VIT dient die Darstellung der Mappings zwischen Data Warehouse Schemata.

Mapforce ist im Vergleich zu den oben genannten wissenschaftlichen Ansätzen ein kommerzielles Produkt. Der Fokus von Mapforce liegt auf dem Mapping zwischen zwei Schemata. Es erlaubt die Darstellung von Operationen beim Mapping zwischen zwei Schemata. Im Gegensatz zu den anderen beiden Ansätzen liegt der Fokus bei Mapforce nicht auf Schemata im Data Warehousing Bereich, sondern nur auf dem Mapping zwischen zwei beliebigen Schemata. Dadurch bietet Mapforce leider keine zufriedenstellende Darstellung von Fakten und Dimensionen. Die Art und Weise, wie in einem kommerziellen Produkt die Darstellung von Mappings umgesetzt wird und wie die Anbindung von Quellsystemen realisiert werden kann wurde bei der Implementierung des VIT berücksichtigt.

Der Global Schema Architect verfolgt im Vergleich zu den anderen vorgestellten Ansätzen und Tools das gleiche Ziel wie das VIT. Es handelt sich um die Integration von unterschiedlichen autonomen Datenquellen, welche mittels eines Mapping-Vorgangs in ein globales Schema integriert werden auf welche mittels SQL-MDi Abfragen zugegriffen wird. Der Unterschied zum VIT ist die Art der Implementierung auf welche in Kapitel 6.3 näher eingegangen wird.

Die oben vorgestellten Ansätze bzw. Tools bieten eine Grundlage für das visuelle Integrationstool, welches im Rahmen dieser Diplomarbeit entwickelt wurde. Die

nachfolgenden Kapitel beschreiben die Planungsphase, die Architektur und die Realisierung des Projekts.

Abschnitt 4

Visuelles Integrationstool

6 Überblick – Visuelles Integrationstool (VIT)

In diesem Kapitel wird zuerst auf die Anforderungen, welche an das VIT gestellt wurden eingegangen. Nachfolgend wir die Lösungsidee erörtert, wie diese Anforderungen umgesetzt werden können. Am Ende dieses Kapitels erfolgt die Einordnung und Abgrenzung zu den zuvor vorgestellten Ansätzen.

6.1 Anforderungen

Um die Zielsetzung, welche in Kapitel 1 beschrieben wurde, zu erfüllen, wird im Rahmen dieser Diplomarbeit ein visuelles Integrations-Tool erstellt, mit welchem man mehrere lokale Data Warehouse Schemata zu einem globalen Schema zusammen führen kann.

Die grundlegenden Anforderungen sind:

- **Auslesen der Quellschemata**

  Funktionalität: Die Schemata werden mit Hilfe von Datenbank-Tools ausgelesen und in der internen Repräsentationsform gespeichert.

  Eingabewert: Datenbank-Metadaten

  Ausgabewert: interne Repräsentationsform

  Fehlerbehandlung: Falls keine Verbindung zum Server hergestellt werden kann, wird eine Fehlermeldung ausgegeben.

- **Grafische Darstellung der Schemata**

  Funktionalität: Die interne Repräsentation der logischen Schemata wird in ein UML Model transformiert und grafisch ausgegeben. Die UML Diagramme stehen dann für die Mapping-Funktion zur Verfügung.

  Eingabewert: interne Repräsentation des Schemas

  Ausgabewert: UML Diagramm

  Fehlerbehandlung: -

- **Erstellen von Mappings zwischen lokalem und globalem Schema**

  Funktionalität: Das Mapping zwischen zwei Schemata erfolgt grafisch anhand der UML Diagramme der logischen Schemata. Es werden Verbindungen zwischen den Dimensionen bzw. Fakten manuell erstellt.

Der Mapping-Vorgang wird mit Hilfe eines Assistenten durchgeführt, mit welchem man die Mappings zwischen lokalem und globalem Schema erstellt. Dabei müssen Attributkonflikte (z.B. Währungskonflikte) zwischen den verschiedenen Schemata berücksichtigt werden.

Eingabewert: logische Schemata

Ausgabewert: interne Repräsentation des Metamodells

Fehlerbehandlung: Bei Verknüpfung von zwei Dimensionen bzw. Fakten ohne semantischen Zusammenhang wird eine Fehlermeldung ausgegeben.

- **Exportieren der Zieldaten**

  Funktionalität: Das erstellte Metamodell wird im Repository gespeichert.

  Eingabewert: interne Repräsentation des Metamodells

  Ausgabewert: gespeicherte Daten im Repository

  Fehlerbehandlung: Bei Schreibfehlern wird eine Fehlermeldung ausgegeben.

In den nächsten Kapiteln wird zuerst die Lösungsidee beschrieben, wie die Anforderungen umgesetzt werden, und in weiterer Folge auf das Design und die Implementierung des VIT eingegangen.

6.2 Lösungsidee

Die Wahl der Programmiersprache fiel aufgrund von Literaturrecherchen und vorhandener Ressourcen (ausschlaggebendes Kriterium) auf die Programmiersprache Java (vgl. [PCTM03]). Java ist die meist verwendete Programmiersprache für Tools, die den CWM Standard verwenden. Weiters bietet Java eine Plattformunabhängigkeit. Es wurde überlegt, die Entwicklung als Plugin für bestehende UML Editoren, wie z.B. Rational Rose bei den UML Package Diagrams (vgl. Kapitel 5.1), oder als Erweiterung von Open Source UML Editoren zu entwickeln. Aus lizenztechnischen Gründen wurde dann die Entscheidung zu Gunsten von Open Source UML Editoren getroffen. In die engere Wahl kamen UML Editoren, die als Eclipse Plugin entwickelt wurden. Die bestehenden Eclipse UML Editoren setzen auf GEF (Graphical Editing Framework [EGEF10]) auf. Die analysierten Editoren (DB Schema Viewer [DBSc07], Blueprint Software Modeller

[Blue07], Eclipse UML & Eclipse UML Studio Edition [EUML07], useitGenerator [Usei07], ArgoUML [Argo07] und AmaterasUML [Amat07]) bieten eine umfassende Funktionalität für diverse UML Diagramme. Leider fehlte den oben genannten Editoren, welche auf Eclipse basieren, die Funktionalität dass zwei Instanzen der Editoren miteinander kommunizieren können, was die Anzeige von zwei Schemata und die Interaktion von zwei Instanzen unmöglich macht. Weiters fehlten den Editoren entsprechende Layout-Funktionen, die für Star-Schemata (vgl. Kapitel 2.5.2.1) nötig sind. Der Nachteil der nicht Eclipse basierten Editoren (z.B. Rational Rose) ist, dass die Einarbeitung und Anpassung länger dauert als eine Eigenentwicklung. Diese Punkte führten zu der Entscheidung das VIT, im Gegensatz zum Global Schema Architect (vgl. Kapitel 5.4), als Standalone-Lösung zu implementieren.

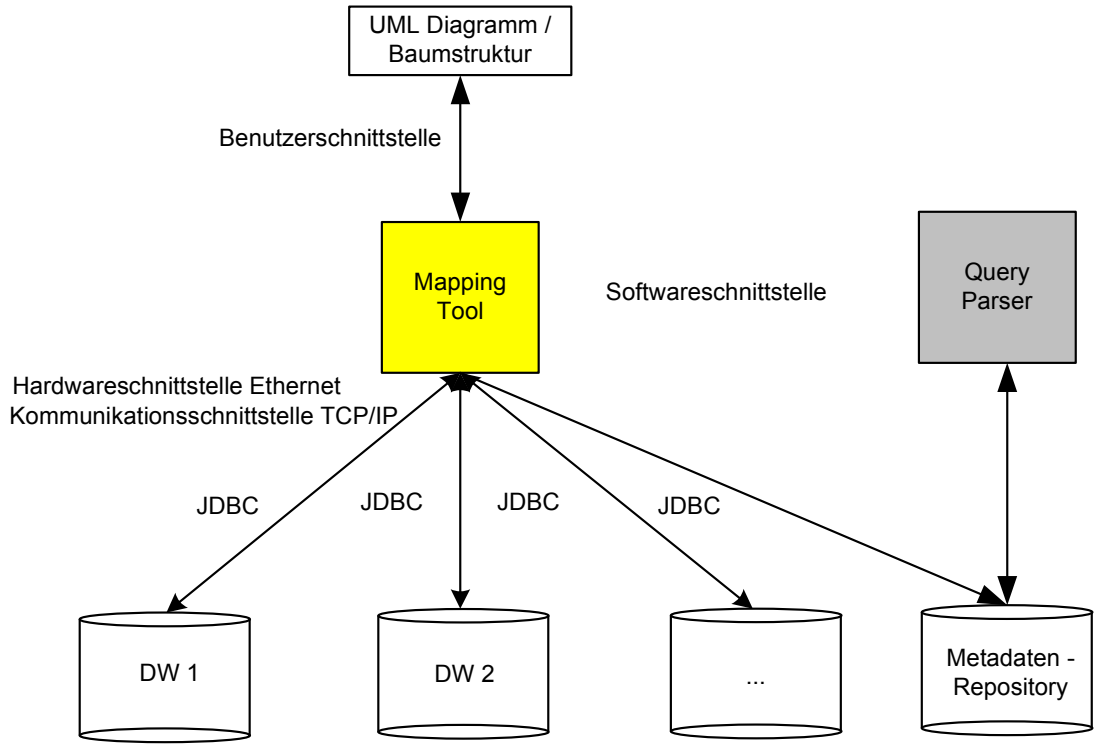

Abbildung 30: Schnittstellenmodell

Abbildung 30 zeigt das Schnittstellenmodell des VIT. Über eine grafische Benutzerschnittstelle werden die Schemata als UML Diagramm dargestellt. Zur Navigation zwischen den Schemata wird eine Baumstruktur verwendet. Das Mapping Tool greift auf die Data Warehouses DW1, DW2, usw. zu und liest die Metadaten aus. Der Zugriff erfolgt über datenbankspezifische JDBC Treiber.

Mittels Mapping Tool werden die Mappings, welche nachfolgend näher beschrieben werden, erstellt und anschließend im Metadaten-Repository gespeichert, welches als Schnittstelle zum Query Parser ([Brunn08]) dient. Der Zugriff auf das Metadaten-Repository erfolgt ebenfalls über JDBC Treiber. Die Metadaten werden in einer relationalen Datenbank abgelegt. Der Query Parser liest die gespeicherten Metadaten aus und verwendet diese für die nachfolgende Bearbeitung.

Das VIT gliedert sich in die zwei Packages „DBManager“ und „VIT“. Die Aufgabe des DBManger Packages ist die Sicherstellung des Zugriffs auf die relationalen Datenbanken, aus denen die Metadaten ausgelesen werden müssen bzw. in die sie im Falle des Metadaten-Repository geschrieben werden müssen. Die Aufgabe des VIT Packages ist die Speicherung der Metadaten, die Transformation der Metadaten, die Darstellung der Metadaten und die Erstellung der Mappings zwischen lokalem und globalem Schema.

6.3 Einordnung und Abgrenzung

Das VIT wurde als Standalone-Lösung entwickelt und grenzt sich damit vom in Kapitel 5.4 vorgestellten Eclipse Plugin, dem Global Schema Architect, durch die Art der Implementierung ab. Weiters liegt beim VIT der Fokus auf der Einbindung von Data Warehouses im Vergleich zu Data Marts beim Global Schema Architect.

Die beiden in Kapitel 5.4 vorgestellten Ansätze „UML Package Diagrams“ und „Data Mapping Diagrams“ verfolgen einen UML-basierten Ansatz zur Darstellung von Schemata und Mappings. Diese beiden Ansätze bilden, wie in Kapitel 5.4 beschrieben, die Grundlage für die Darstellung im VIT, da sie die Thematik behandeln, wie Star-Schemata und deren Mappings in Editoren dargestellt werden können.

Bei der Verwaltung und Transformation der einzelnen Schemata wird die in Kapitel 3.3.3.2 vorgestellte Fünf-Level-Schema-Architektur als Referenzarchitektur verwendet. Dabei wird darauf geachtet, dass zwischen den einzelnen Schemata ein Prozessor für die Transformation/Filterung verwendet wird, um die Schemata sauber voneinander abzugrenzen.

Für die Integration der lokalen Schemata zu einem globalen Schema wird die in Kapitel 3.4.1.1 vorgestellte One-Shot-Integrationsstrategie verwendet. Dies

bedeutet, dass jedes lokale Schema direkt auf das globale Schema gemappt wird und es keine teilintegrierten Schemata gibt. Der Integrationsprozess verläuft anschließend so wie in Kapitel 3.4.2 vorgestellt.

Mit dem VIT wird versucht die in Kapitel 3.4.3 vorgestellten Integrationskonflikte zu beheben. Semantische Konflikte, Beschreibungskonflikte und strukturelle Konflikte können mit dem VIT behoben werden. Heterogenitätskonflikte zwischen relationalen und objektorientierten Schemata können nicht behoben werden, da das VIT für Star-Schemata in relationalen Datenbanken konzipiert wurde.

Die in Kapitel 3.4.4 vorgestellten Konfliktlösungsansätze werden bei der Entwicklung des VIT nur zum Teil berücksichtigt. Da das VIT nur die Integration von relationalen Modellen unterstützt, können die Integration von Klassenhierarchien, die formalisierte objektorientierte Integration und das generische Integrationsmodell nicht berücksichtigt werden. Diese Punkte wurden aber auf Grund der Vollständigkeit in dieser Arbeit erläutert. Das Konzept der Zusicherung wurde aber bei der Umsetzung des VIT berücksichtigt. Die vorgestellten Integrationsregeln wurden mit der in Kapitel 7.1.4 vorgestellten Algebra realisiert.

7 Design des VIT

In diesem Kapitel wird auf das Design des VIT, die einzelnen Funktionen und den generellen Ablauf eingegangen.

Anhand des Use Case Diagramms (siehe Abbildung 31) wird erläutert, welche Funktionen vom Integrationstool übernommen werden müssen und welche beteiligten Systeme bzw. Personen es gibt.

Die Akteure in diesem Use Case Diagramm sind:

- Modellierer
- Integrationstool
- Datenbank/Repository

Abbildung 31 zeigt den kompletten Funktionsumfang des VIT. Für das Auslesen der Quellschemata ist eine Verwaltung der Quellsysteme notwendig. Der Modellierer muss die Möglichkeit haben neue Quellsysteme hinzuzufügen oder bestehende zu ändern oder zu löschen. Beim Auslesen muss das Integrationstool

eine Verbindung zur Datenbank aufbauen, das physische Schema auslesen und speichern. Der Modellierer kann das ausgelesene physische Schema nicht verändern. Für die Darstellung der Schemata müssen die physischen Schemata der Quellsysteme zuerst in logische Schemata konvertiert werden. Diese logischen Schemata werden als UML Diagramme dargestellt. Der komplette Aufbau des grafischen Editors und die Layout-Funktion der Star-Schemata wurden eigens entwickelt. Um Abfragen auf ein föderiertes Data Warehouse zu ermöglichen, ist es notwendig ein gemeinsames globales Schema zu haben, auf welches Abfragen erfolgen können. Um diese Abfragen an die diversen Quellschemata weiterzuleiten, ist es notwendig Mappings zwischen globalem Schema und Quellschema zu erstellen. Beim VIT wird die One-Shot-Integrationsstrategie (vgl. Kapitel 3.4.1.1) verwendet. Es wird versucht jedes Quellschema direkt in einem Schritt auf das globale Schema zu mappen. Die Erstellung der Mappings erfolgt über einen so genannten Mapping Assistenten, mit dessen Hilfe Mappings zwischen Fakten und Dimensionen der zu mappenden Schemata erstellt werden können. Nach Integration aller vorhandenen Quellschemata werden die Metadaten und Mappings im Metadatenrepository gespeichert, welches als Schnittstelle zum Query Parser dient (vgl. Kapitel 6.2).

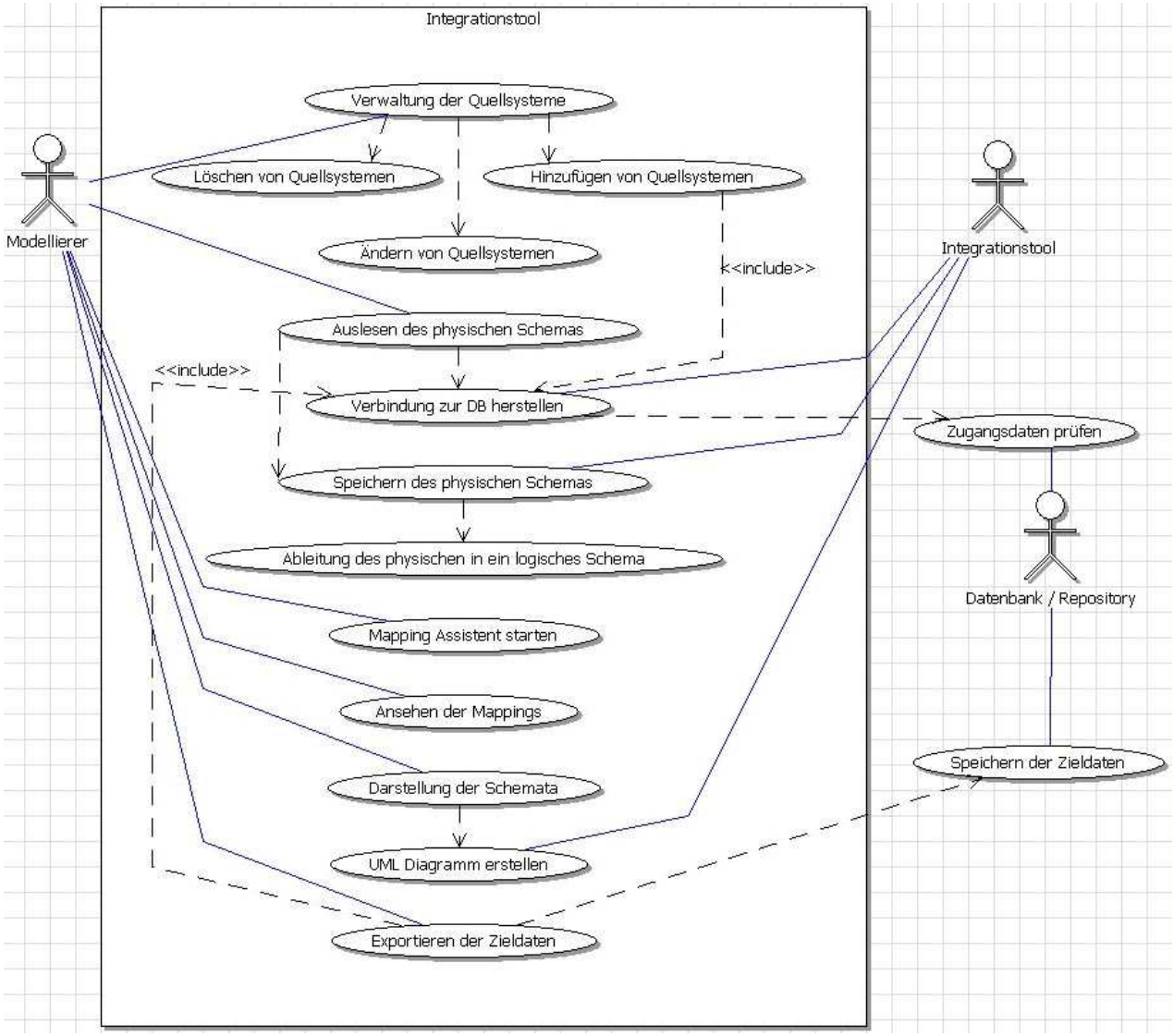

Abbildung 31: Use Case Diagramm

7.1 Funktionen des VIT

In diesem Kapitel werden die diversen Funktionen des VIT anhand von Sequenzdiagrammen näher beschrieben. Eine technische Beschreibung der dahinterliegenden Architektur erfolgt in Kapitel 8.

Die Darstellung der Sequenzdiagramme wurde in die drei Bereiche Benutzer (Modellierer), Integrationstool und Datenbank gegliedert, um eine einheitliche Darstellung zu gewährleisten.

7.1.1 Quellsystem verwalten

Zum Auslesen der Quellschemata und zum Exportieren der Zielschemata müssen relationale Datenbanken angebunden werden. Die Verwaltung dieser

Quellsysteme erfolgt über die Softwarekomponente „DBManager“. Die Quellsystemverwaltung umfasst die Funktionalitäten Hinzufügen, Ändern und Löschen von Quellsystemen. Abbildung 32 zeigt schematisch den Ablauf beim Hinzufügen von Quellsystemen. Der Benutzer startet die Funktion „Hinzufügen von Quellsystemen“. Das Integrationstool öffnet eine Eingabemaske für die Zugangsdaten (DB Metadaten) (siehe Abbildung 33). Nach Eingabe der Zugangsdaten kann der Benutzer die Verbindung testen. Dabei versucht das Integrationstool auf die definierte Datenbank mittels JDBC (Java Database Connectivity) Treiber zuzugreifen. Nach dem Verbindungstest erfolgt die Ausgabe einer Erfolgs- bzw. Fehlermeldung. Der Benutzer hat weiters die Möglichkeit die Zugangsdaten zu speichern, um sie für das anschließende Auslesen des physischen Schemas bzw. das Exportieren der Zieldaten zu verwenden.

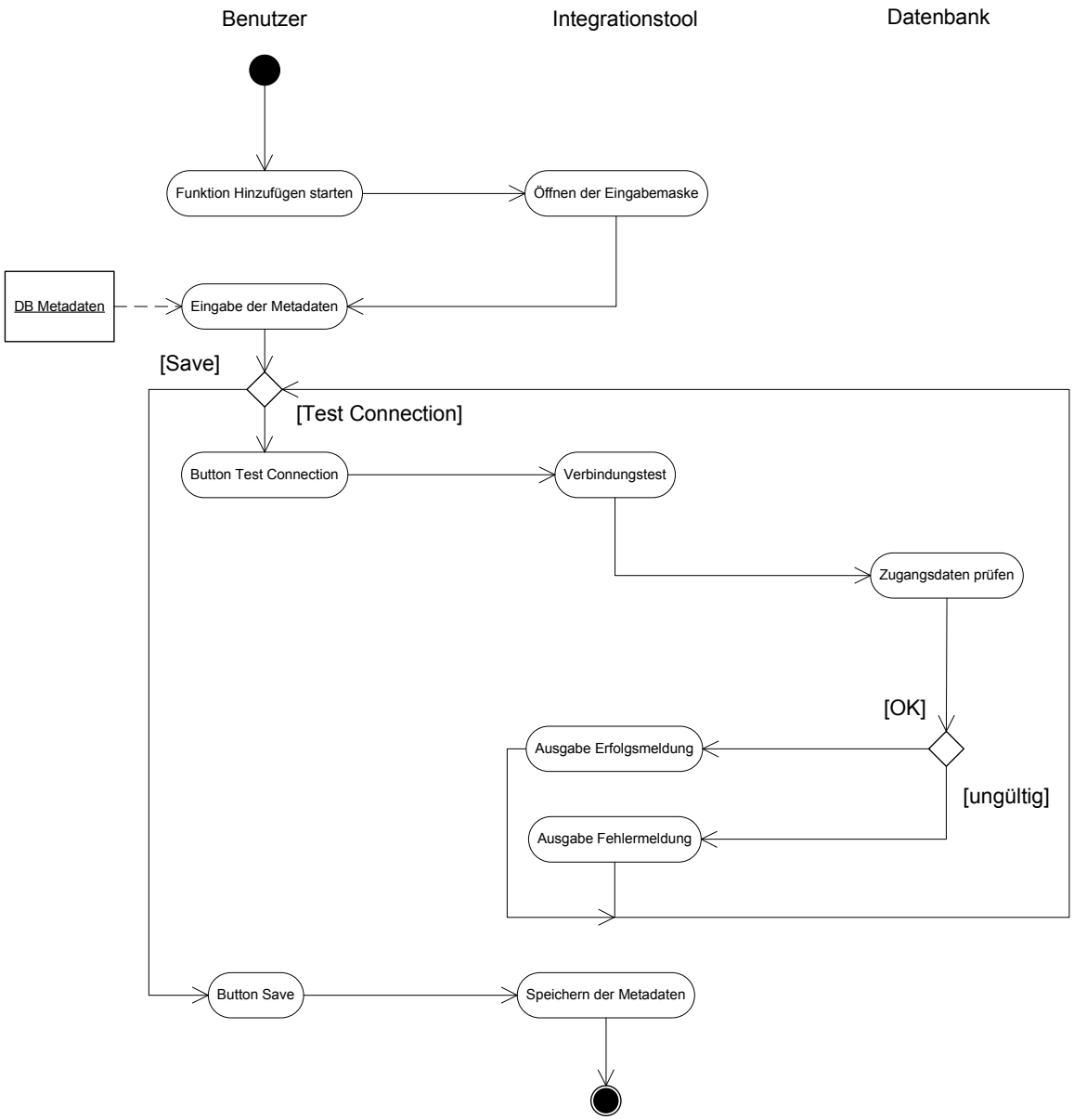

Abbildung 32: Quellsystem hinzufügen

Abbildung 33 zeigt die Eingabemaske zur Anbindung neuer Quellsysteme. Nach Eingabe der Zugangsdaten kann ein Verbindungstest durchgeführt werden und/oder es können die Zugangsdaten gespeichert werden. Derzeit ist eine Anbindung von MS SQL Server, Oracle und Sybase möglich.

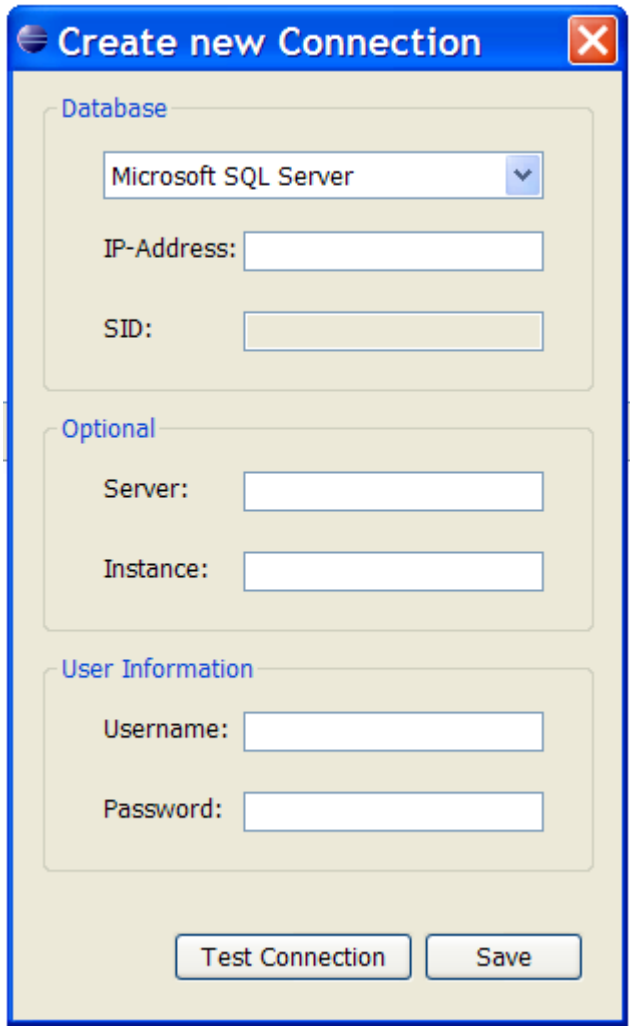

Abbildung 33: Eingabemaske DBManger

7.1.2 Auslesen eines physischen Schemas

Zum Auslesen eines physischen Schemas wird auf ein zuvor angebundenes Quellsystem (vgl. Kapitel 7.1.1) zugegriffen. Abbildung 34 zeigt schematisch den Ablauf beim Auslesen eines physischen Schemas. Es wird aus einer Liste der angebunden Quellsysteme das gewünschte System ausgewählt (siehe Abbildung 35). Anschließend kann definiert werden, ob es sich bei dem Schema um das globale Schema handelt bzw. ob es sich dabei um ein anzubindendes Quell Data Warehouse handelt. Bei der Auswahl des „Load Schema“ Buttons wird eine Verbindung zur Datenbank hergestellt und das physische Datenbankschema ausgelesen. Anschließend wird das Schema mittels eines Transformationsprozessors (vgl. Kapitel 3.3.2.1) in der internen Repräsentationsform gespeichert, welche für die Darstellung der Schemata notwendig ist.

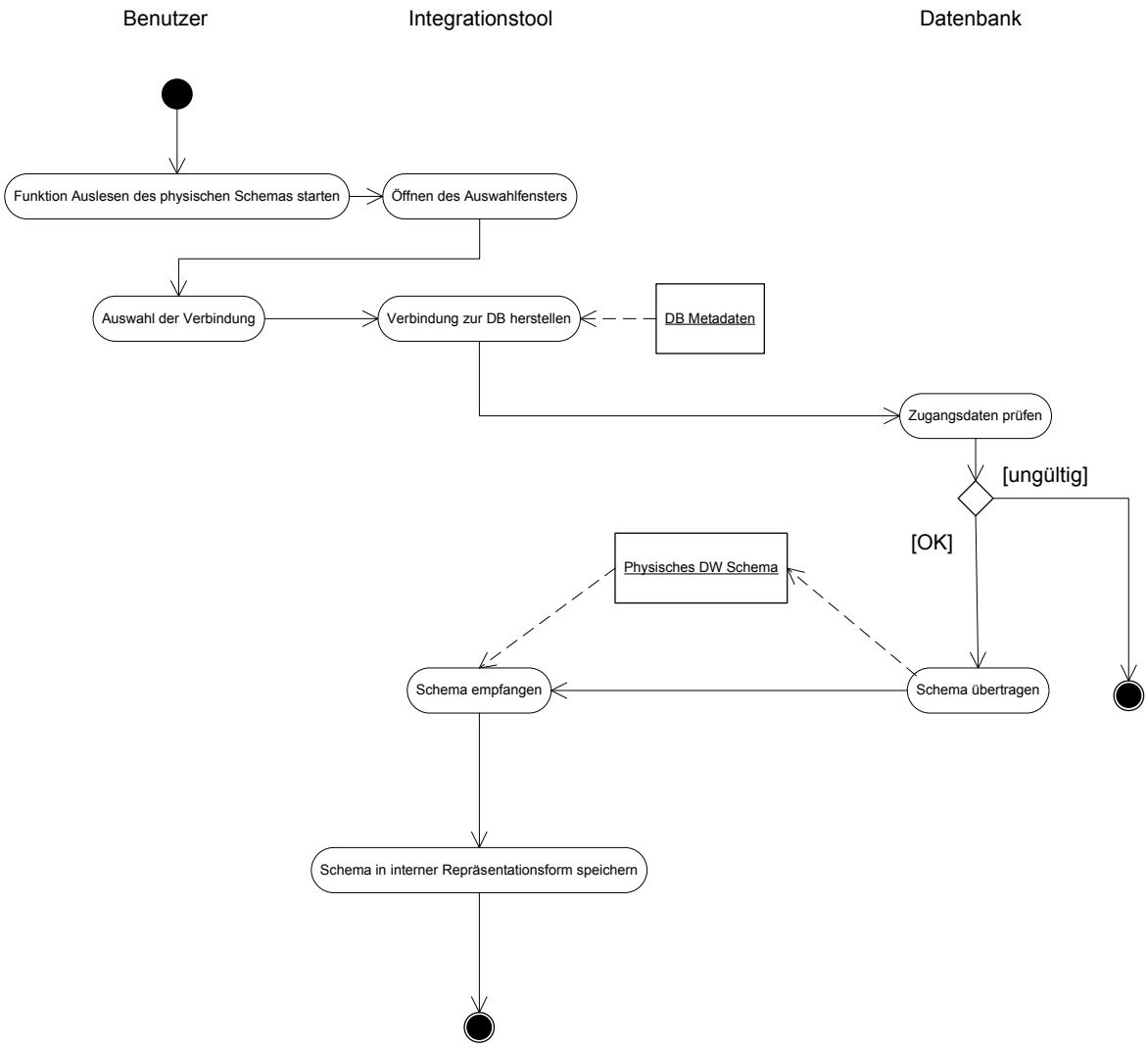

Abbildung 34: Auslesen eines physischen Schemas

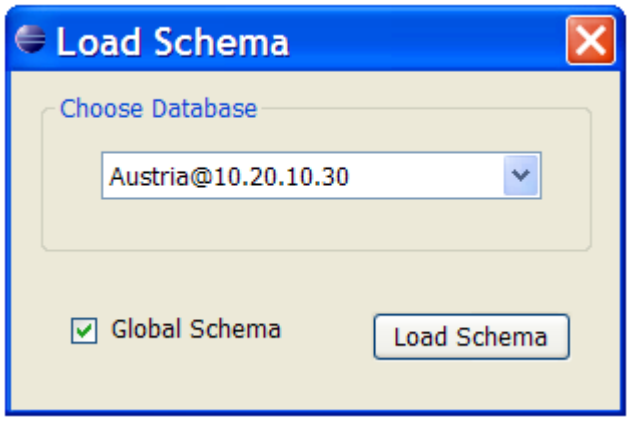

Abbildung 35: Physiches Schema laden

7.1.3 Darstellen eines logischen Schemas

Die Darstellung der Schemata erfolgt auf zwei Arten. Einerseits wird das Schema zur Übersicht als Baumstruktur und andererseits als UML Diagramm dargestellt. Abbildung 36 zeigt die Oberfläche des VIT. Im linken Bereich werden alle Schemata (Globales Schema und Quell Schemata) in einem Baum angezeigt. Der rechte Teil gliedert sich in die zwei Bereiche „Globales Schema“ (oben) und „Quellschemata“ (unten), welche die UML Diagramme enthalten. Die einzelnen Quellschemata können über Registerblätter ausgewählt werden.

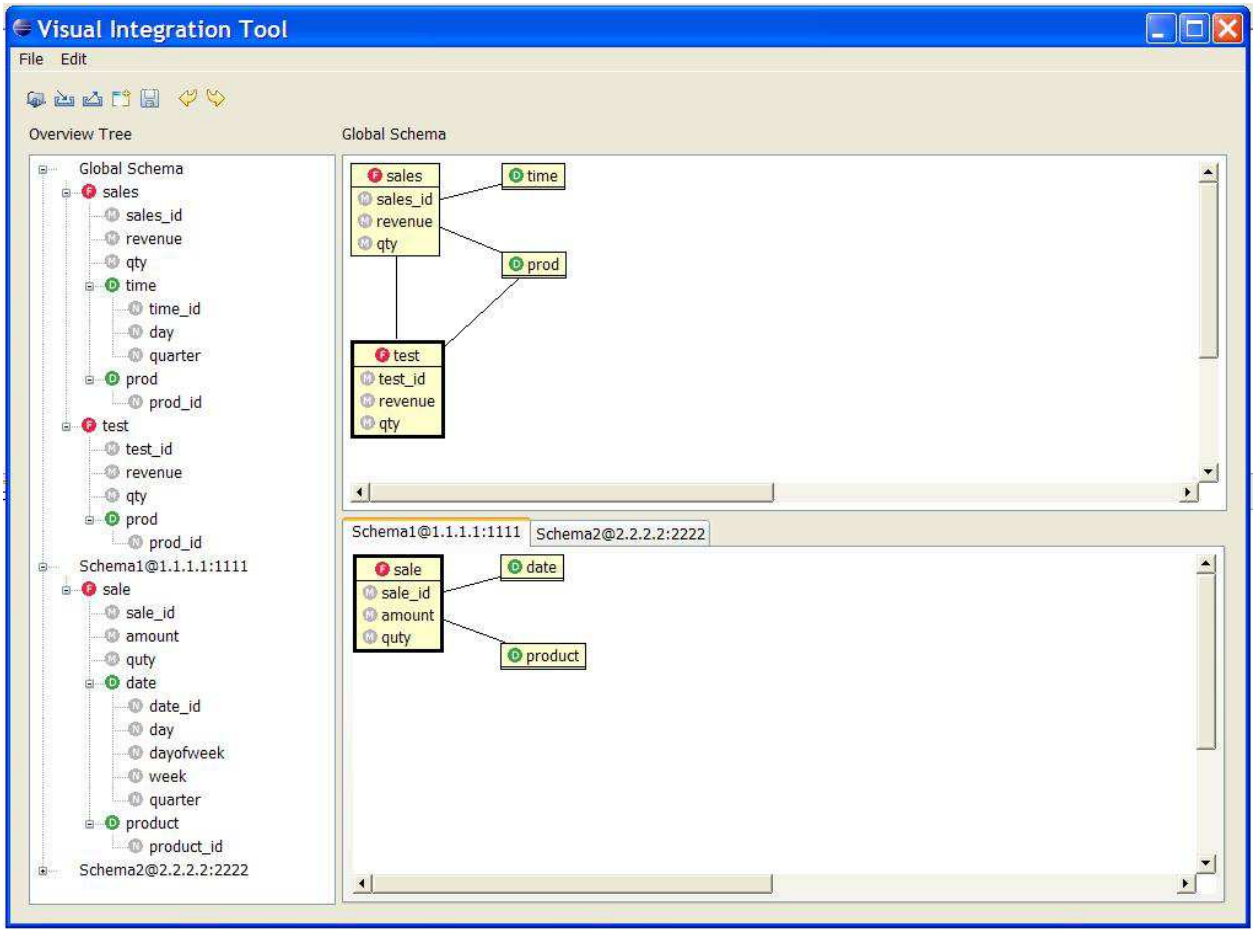

Abbildung 36: Visual Integration Tool

7.1.3.1 Baumstruktur

Die Übersicht auf der linken Seite in Abbildung 36 (Overview Tree) erfolgt mit Hilfe eines Baumes, in dem die geladenen Schemata dargestellt werden. Das oberste Schema ist immer das globale Schema. Abbildung 37 zeigt ein Beispiel für die Baumstruktur für ein globales Schema.

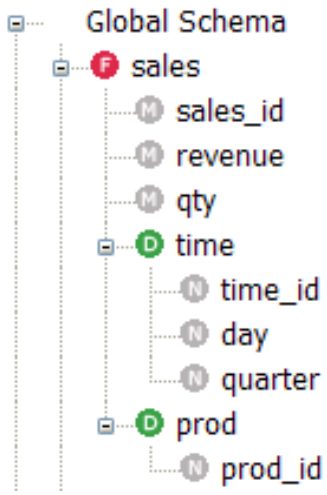

Abbildung 37: Baumstruktur

Der Baum gliedert sich in folgende Bereiche:

- Schema
  - Fact
    - Measure
    - Dimension
      - Dimensionales Attribut
      - Nichtdimensionales Attribut

Die Baumstruktur dient zur übersichtlichen Darstellung aller Schemata. Da beim Import aus der relationalen Datenbank nicht festgestellt werden kann, ob Levelattribute dimensional oder nichtdimensional sind, kann dies über das Kontextmenü definiert werden (siehe Abbildung 38). Weiters kann für lokale Schemata definiert werden, ob eine Dimension benötigt wird oder nicht.

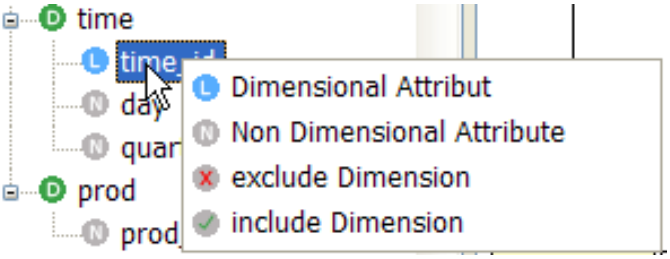

Abbildung 38: Kontext Menü

7.1.3.2 UML Diagramm

Die Übersicht als UML Diagramm teilt sich in den Bereich für das globale Schema und den Bereich für die lokalen Schemata auf (siehe Abbildung 36). Der untere Bereich für die lokalen Schemata ist mittels Registerblättern für die einzelnen lokalen Schemata aufgeteilt.

Für die Darstellung der UML Schemata wurde das Draw2D Framework verwendet, welches auch die Basis für das GEF (Graphical Editing Framework [EGEF10]) darstellt. Mit diesem Framework können die Assoziationen zwischen den einzelnen Objekten (Fakten und Dimensionen) dargestellt werden. Das Framework übernimmt die Berechnung und Darstellung der Linien nach Veränderungen des Layouts.

Abbildung 39 zeigt die Darstellung der Fakten (mit Merkmalen) und Dimensionen (mit dimensionalen Attributen) mit ihren Assoziationen als UML Diagramm.

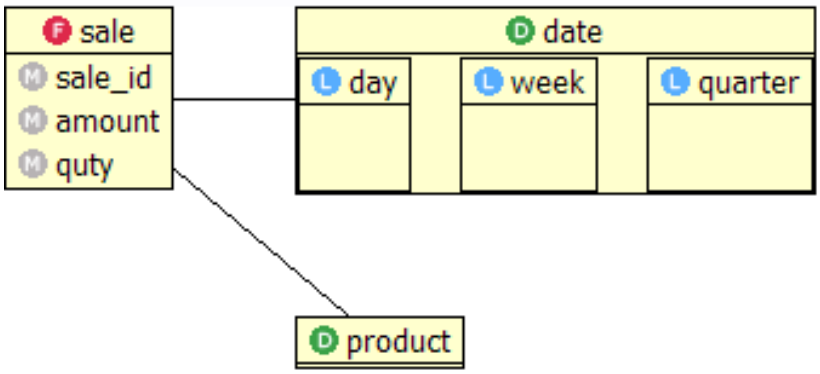

Abbildung 39: UML Diagramm

Das UML Diagramm gliedert sich in folgende Komponenten:

- Fakt Element
- Dimension Element
- Level Element
- Assoziation

Für das Layout der UML Elemente wurde ein eigener Algorithmus implementiert. Zuerst wurde überlegt, ob auf bestehende Algorithmen (z.B. Spring Layout) zurückgegriffen werden sollte, doch diese wurden aufgrund der Unübersichtlichkeit

verworfen. Nachfolgend wird der Algorithmus in Beschreibungssprache dargestellt:

```
x=0, yFact=0, yDim=0
for(all facts) {
        drawFact(curFact, x, yFact)
        x = 50 + curFact.getWidth()
        for(all dimensions of fact) {
                drawDimension(curDimension, x, yDim
                yDim = yDim + 50 + curDimension.getHeight()
                if(yDim > yFact) {
                        yFact = yDim
                }
        }
        yFact = yFact + 50 +curFact.getHeight()
}
```

Abbildung 40 zeigt die Darstellung mehrerer Fakten und Dimensionen.

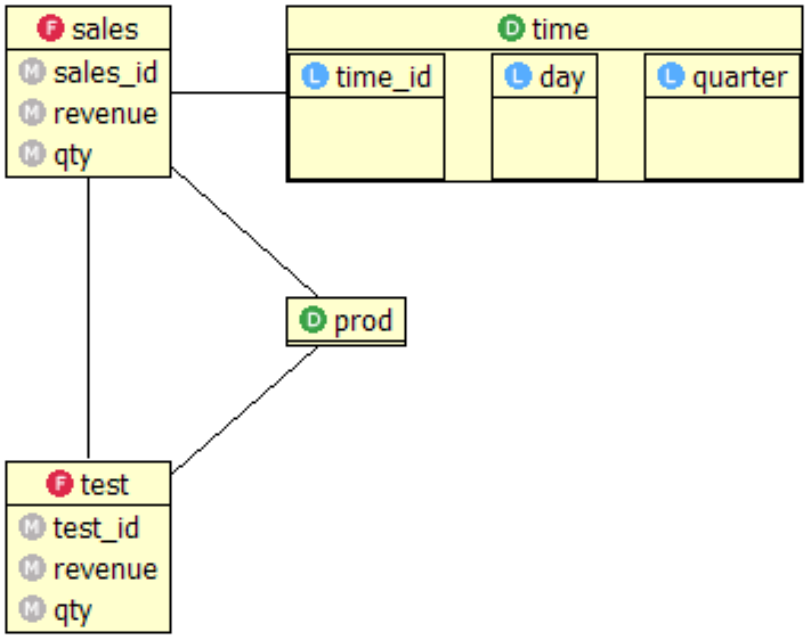

Abbildung 40: Layout

Besteht eine Beziehung zwischen den einzelnen Elementen, wird eine Assoziation über eine Verbindungslinie dargestellt. Die zu einer Dimension gehörenden dimensionalen Attribute werden unsortiert in einem Dimensionselement dargestellt

(siehe Abbildung 41). Nichtdimensionale Attribute werden nur in der Baumstruktur und nicht im UML Diagramm dargestellt.

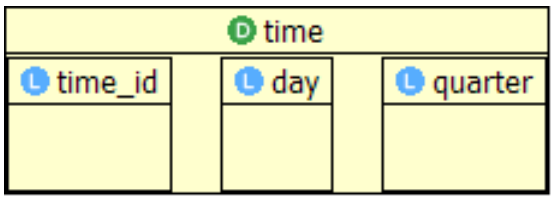

Abbildung 41: Dimension mit Level

7.1.4 Erstellen von Mappings

Beim Mapping werden Dimensionen und Fakten eines lokalen Data Warehouses mit dem globalen Schema gemappt. Fakten können nur auf Fakten und Dimensionen nur auf Dimensionen gemappt werden. Abbildung 43 zeigt schematisch den Ablauf des Mapping-Vorgangs. Zuerst muss sowohl ein Element im globalen Schema als auch ein Element im lokalen Schema markiert werden. Auf Dimensionen und Fakten im UML Diagramm kann ein Fokus per Mausklick gesetzt werden, mit Hilfe dessen das Element markiert wird. Abbildung 42 zeigt ein Fakt, auf das ein Fokus gesetzt wurde.

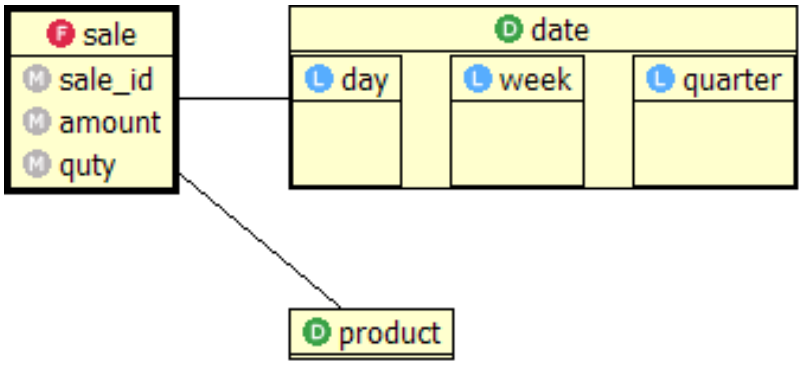

Abbildung 42: Fokus auf Fakt

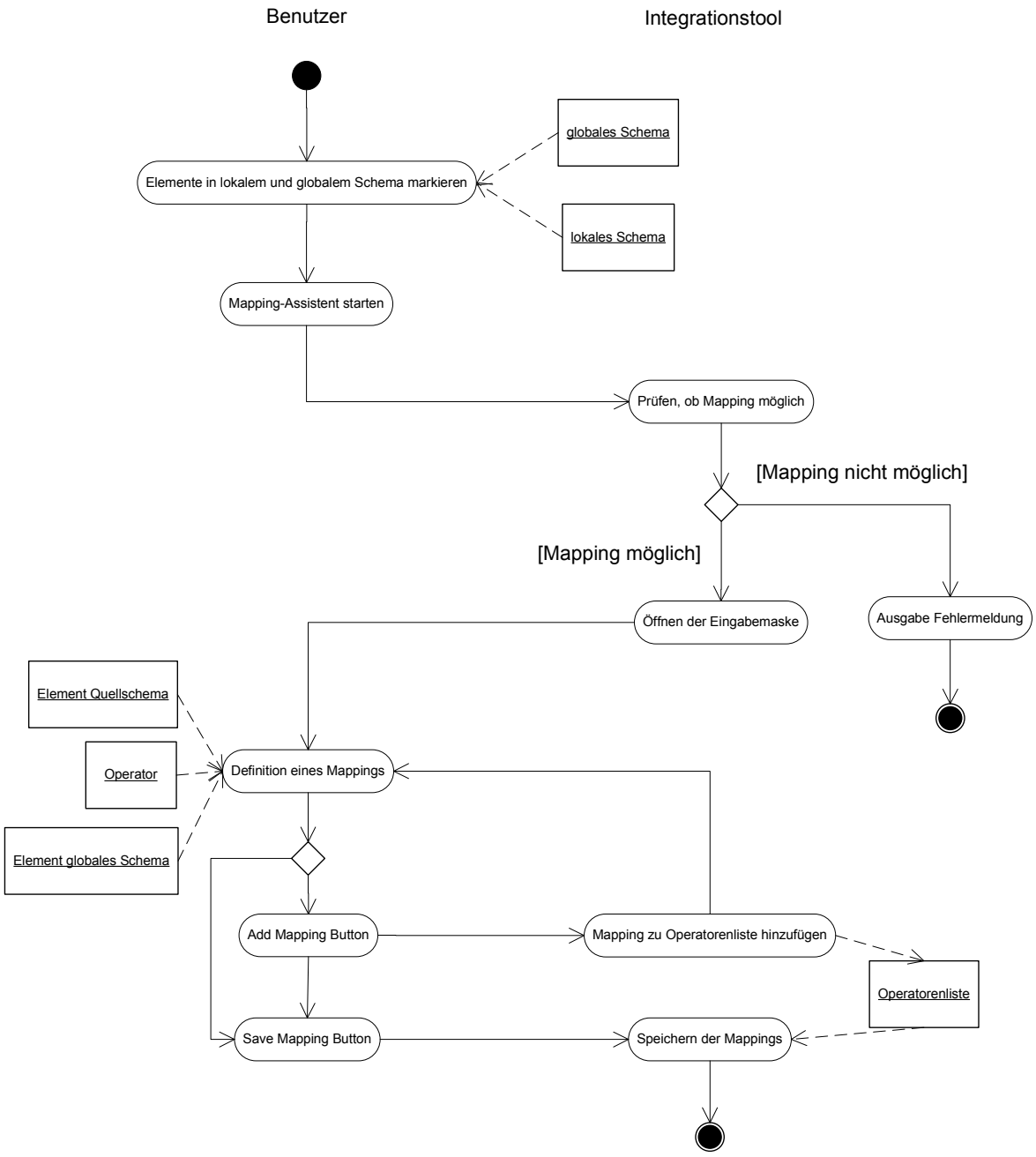

Abbildung 43: Ablauf Mapping-Vorgang

Nachdem die entsprechenden Dimensionen bzw. Fakten im globalen und im Quellschema ausgewählt wurden, kann der Mapping-Assistent über das Menü gestartet werden. Dieser dient zur Unterstützung des Mapping-Vorgangs. Es wird geprüft, ob ein Mapping möglich ist (Fakt-Fakt bzw. Dimension-Dimension). Abbildung 44 zeigt den Aufbau des Mapping-Assistenten. Dieser gliedert sich in zwei Baumdarstellungen für „Quellschema“ (Source Schema) und „globales Schema“ (Global Schema), ein Drop-Down-Menü für die Mapping-Operatoren, welche zuvor beschrieben wurden, ein Feld für Parameter zu den Operatoren und

einen Bereich für die bereits definierten Mappings. Um ein Mapping zu definieren, wird ein Merkmal (bzw. Dimension) im Quellschema und ein Merkmal (bzw. Dimension) im globalen Schema ausgewählt. Anschließend wird ein Operator, und falls notwendig Parameter, definiert.

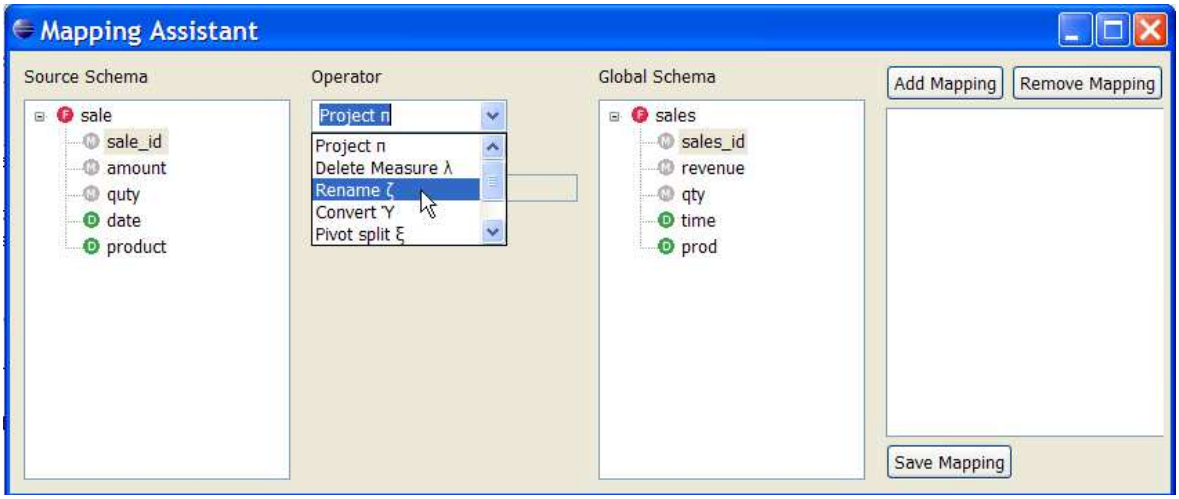

Abbildung 44: Mapping-Assistent

Über den Button „Add Mapping“ wird das Mapping zur Operatorenliste hinzugefügt (siehe Abbildung 45). Mit dem Button „Remove Mapping“ können bereits definierte Mappings wieder von der Liste entfernt werden. Über den Button „Save Mapping“ werden die definierten Mappings gespeichert.

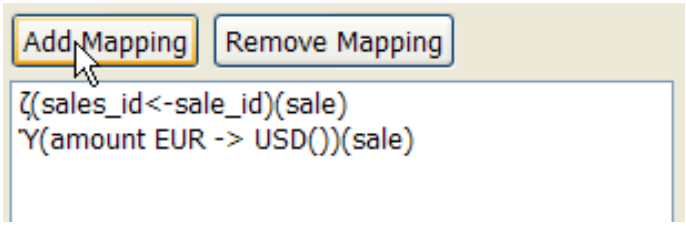

Abbildung 45: Operatoren Liste

Es wird zwischen dem Mapping von Dimensionen und dem Mapping von Fakten unterschieden. Folgende Operatoren, welche in [Brun08] nachzulesen sind, können verwendet werden:

Dimension Algebra:

- Rename ζ – mit diesem Operator können Namen von Attributen umbenannt werden

- Convert ϒ - mit diesem Operator kann die Domäne eines dimensionalen Attributs oder eines nichtdimensionalen Attributs einer Klassifikationsstufe geändert werden, indem eine Funktion angewendet wird
- Delete level α - mit diesem Operator kann eine Klassifikationsstufe einer Dimension aus allen Hierarchien, in denen sie verwendet wird, gelöscht werden

Fact Algebra:

- Project π - dieser Operator reduziert die Dimensionalität, indem eine Projektion auf ein oder mehrere dimensionale Attribute erfolgt
- Delete Measure λ - dieser Operator berechnet eine Faktmenge, in welcher die Anzahl der Kenngrößen-Attribute durch Projektion auf ein oder mehrere Kenngrößen-Attribute reduziert wurde
- Rename ζ - mit diesem Operator können Namen von Attributen geändert werden
- Convert ϒ - mit diesem Operator kann die Domäne eines Kenngrößen-Attributs geändert werden, indem eine Funktion angewendet wird
- Pivot ξ / χ - mit den beiden Pivot-Varianten können Schema-Instanz-Konflikte aufgelöst werden
- Enrich dimension ε - dieser Operator erhöht die Dimensionalität, indem ein zusätzliches dimensionales Attribut mit fixem Wert eingeführt wird

7.1.5 Darstellen von Mappings

Wurden bereits Mappings zwischen Quellschema und globalem Schema definiert, werden diese färbig in der Baumstruktur und im UML Diagramm dargestellt (siehe Abbildung 46).

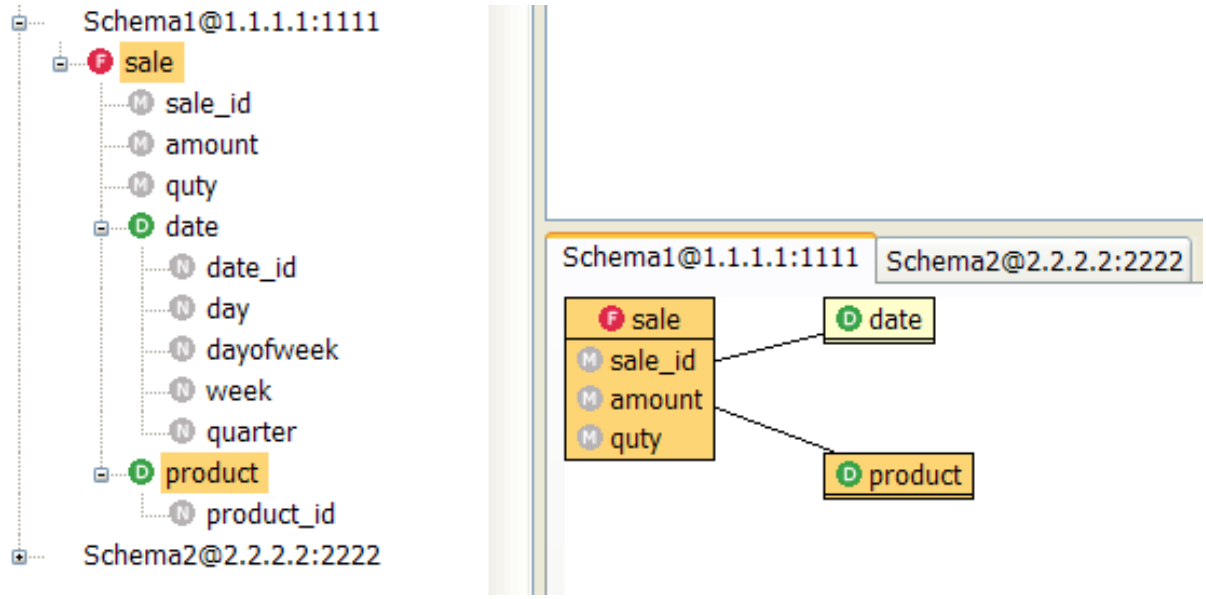

Abbildung 46: Markierte Mappings

Über Doppelklick auf ein färbig markiertes Element im UML Diagramm werden die definierten Mapping-Operatoren in einem Fenster ausgegeben (siehe Abbildung 47).

Abbildung 47: Definierte Mappings

Die Darstellung der Mapping Operatoren kann bei einer Erweiterung des VIT weiter entwickelt werden.

7.1.6 Exportieren der Zieldaten

Nach der Definition aller Mappings zwischen Quellschema und globalem Schema können die Zieldaten exportiert werden. Dafür muss eine Verbindung zum Metadaten-Repository definiert werden (vgl. 7.1.1), in welchem die Zieldaten gespeichert werden sollen. Abbildung 48 zeigt die schematische Darstellung dieses Export-Vorganges.

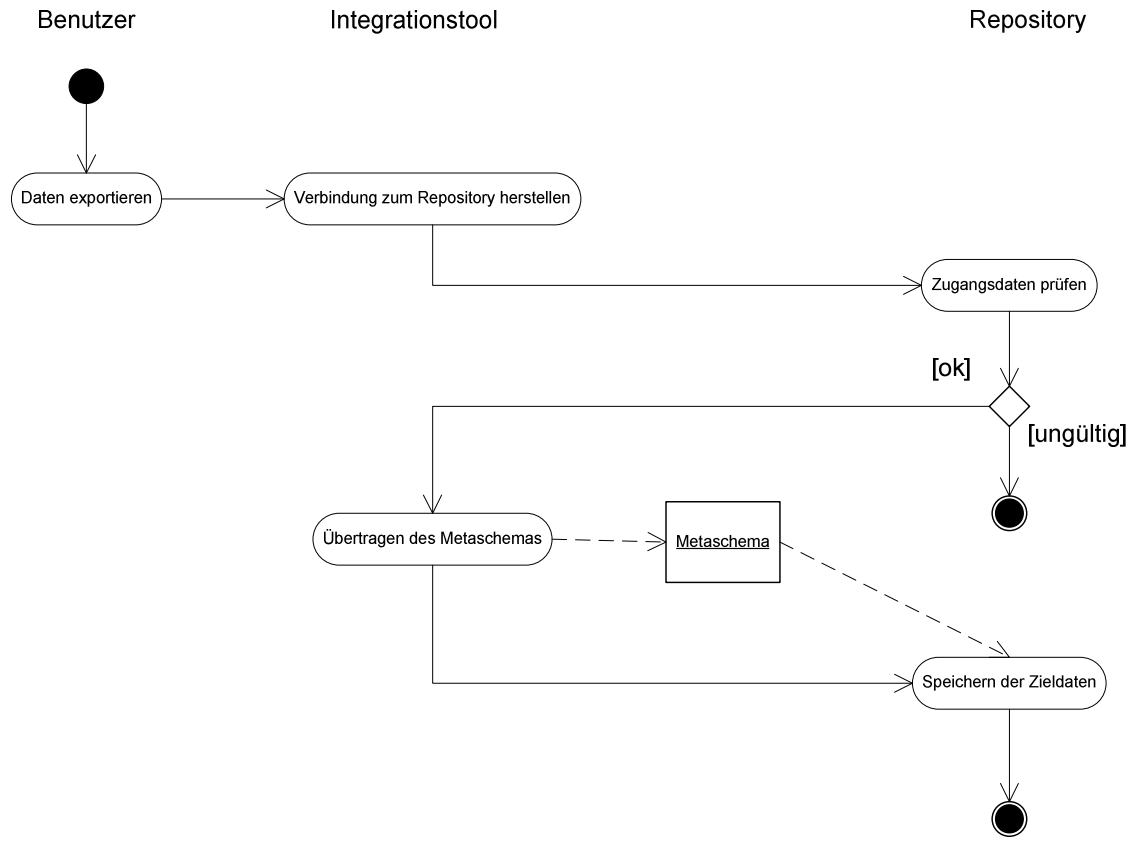

Abbildung 48: Exportieren der Zieldaten

7.1.6.1 Repository

Das Metadaten-Repository wurde CWM-konform entworfen und dient zum Austausch der Metadaten zwischen VIT und Query Parser, welcher in [Brun08] näher erläutert wird. Auf die Definition des Repositories und der zugehörigen Klassen wird im Zuge dieser Arbeit nicht näher eingegangen, da diese schon in [Brun08] ausgiebig erläutert wurde. (vgl. Kapitel 4)

7.2 Genereller Ablauf aus Sicht des Modellierers

In diesem Kapitel wird der generelle Ablauf bzw. die Bedienung des VIT aus Sicht des Modellierers anhand eines Beispiels erläutert. Dabei wird nicht auf die technischen Abläufe im Hintergrund (vgl. Kapitel 7.1), sondern nur auf die Tätigkeiten des Benutzers (Modellierers) eingegangen.

Beispiel:

Eine Firma besteht aus einer Muttergesellschaft in Österreich und einer neu gekauften Tochtergesellschaft in Tschechien. Beide Firmen haben bereits ihr eigenes Data Warehouse im Einsatz. Für eine konzernweite Analyse der Verkaufszahlen sollen die beiden Data Warehouses föderiert werden.

Folgende Schemata werden verwendet:

- Schema1: Data Warehouse in Österreich
- Schema2: Data Warehouse in Tschechien
- Global Schema: Föderiertes Schema

7.2.1 Auslesen der Metadaten

Als erster Schritt müssen die Quellsysteme angebunden werden. Dafür wird wie in Kapitel 7.1.1 vorgegangen. Mit dem Button „Create New Connection“  wird die Eingabemaske zur Eingabe der Datenbankverbindungsdaten geöffnet. Dieser Vorgang muss für alle Quellsysteme durchgeführt werden.

7.2.2 Auswahl des lokalen Schemas

Wenn alle Quellsysteme erfolgreich angebunden wurden, können die Metadaten aus den Quellsystemen geladen werden (vgl. Kapitel 7.1.2). Mit dem Button „Load Schema from Database“  wird die Eingabemaske zum Auswählen der zu ladenden Quellsysteme geöffnet. Die Quellsysteme werden über eine Drop-Down-Box selektiert. Dieser Vorgang muss für alle zu ladenden Schemata durchgeführt werden. Wenn es sich bei dem Schema um das globale Schema handelt, muss zusätzlich noch das Feld „Global Schema“ angehakt werden.

7.2.3 Bearbeiten von Dimensionen

Nachdem alle benötigten Schemata aus den Quellsystemen geladen wurden, kann mit der Zuordnung begonnen werden. Im ersten Schritt werden die Attribute der Dimensionen über den Übersichtsbaum als dimensional oder nichtdimensional definiert (vgl. Abbildung 38).

Abbildung 49, Abbildung 50 und Abbildung 51 zeigen die drei bearbeiteten Schemata.

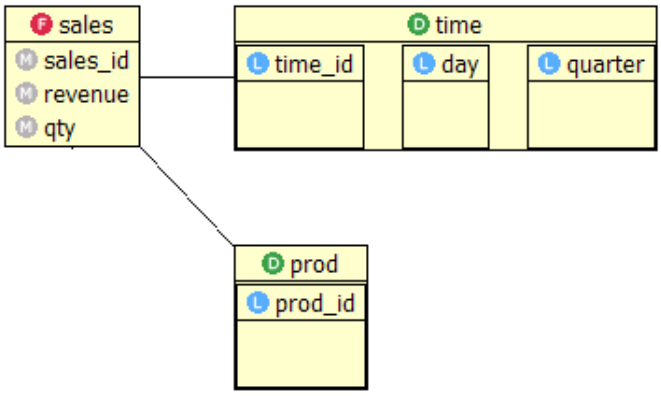

Abbildung 49: Global Schema

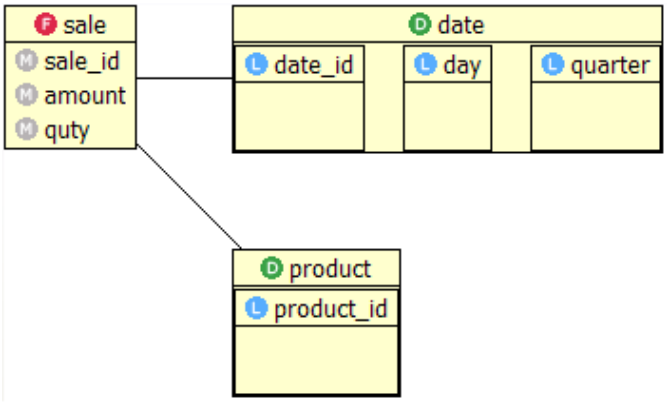

Abbildung 50: Schema1

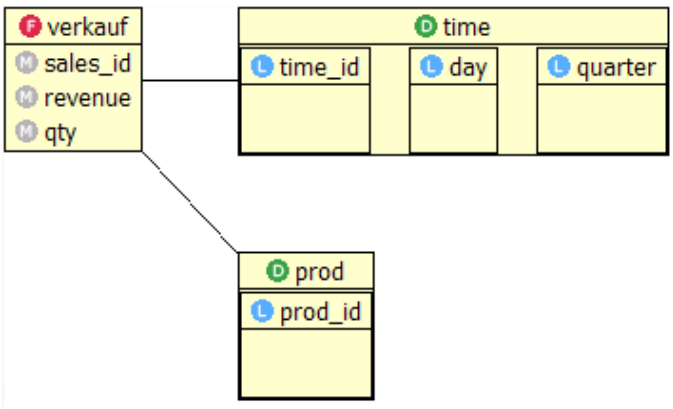

Abbildung 51: Schema2

7.2.4 Erstellen von Mappings

Nachdem die Bearbeitung der Dimensionen der ausgelesenen Schemata abgeschlossen ist, kann mit dem Mapping der Dimensionen und Fakten begonnen werden. Die Mappings werden immer zwischen einem Quellschema und dem globalen Schema definiert. Um ein Mapping zu erstellen, muss immer ein Element (Fakt oder Dimension) im Quellschema und im globalen Schema markiert werden (vgl. Kapitel 7.1.4). Mit dem Button „Start Mapping Assistent“ wird der Mapping-Assistent gestartet.

Anschließend werden die Mappings, wie in Kapitel 7.1.4 beschrieben, definiert. Abbildung 52 zeigt ein Mapping zwischen der Dimension „product“ von Schema1 und der Dimension „prod“ des globalen Schemas.

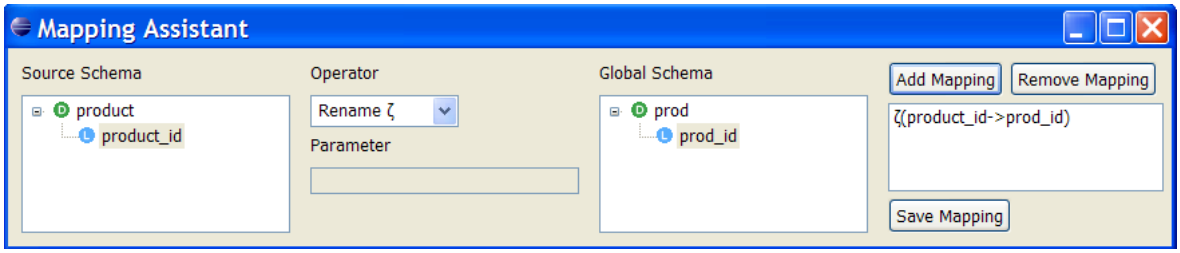

Abbildung 52: Mapping

Der Mapping-Vorgang muss für alle Fakten und Dimensionen für die Quell-Data Warehouses (Schema1 und Schema2) durchgeführt werden.

7.2.5 Exportieren der Zieldaten

Nachdem der Mapping-Vorgang abgeschlossen wurde, können die Zieldaten exportiert werden (vgl. Kapitel 7.1.6). Mit dem Button „Export Schema to Database“ wird eine Auswahlmaske zur Selektion einer angebundenen Datenbank geöffnet. Mit dem Button „Export Schema“ werden die Daten exportiert.

Mit dem Export der Zieldaten ist der Prozess abgeschlossen.

8 Implementierung

In diesem Kapitel wird die technische Implementierung des VIT näher beschrieben. Als Implementierungssprache wurde die Programmiersprache Java verwendet (vgl. Kapitel 6.2).

Das „Visual Integration Tool“ gliedert sich in zwei Packages:

- DBManager
- VIT

Aufgrund der Wiederverwendbarkeit der einzelnen Komponenten wurde eine Trennung in zwei Packages vorgenommen. Der DBManager kümmert sich um die Verwaltung der angebundenen Datenbanken und das VIT kümmert sich um die restlichen in Kapitel 7.1 beschriebenen Funktionen.

Als Architekturmuster wurde das Model-View-Controller-Prinzip (MVC) verwendet. Beim MVC wird darauf geachtet, dass die Datenhaltung (Model), die Programmlogik (Controller) und die Darstellung (View) so gekapselt sind, dass diese Komponenten später einfach erweitert bzw. ausgetauscht werden können. (vgl. [Reen03])

Die Klassen des CWM Standards (vgl. Kapitel 4) wurde nur für die Schnittstelle zum Query Parser (vgl. [Brun08] verwendet. Eine genaue Definition der einzelnen Klassen ist (wie in Kapitel 7.1.6.1) in [Brun08] nachzulesen.

8.1 DBManager

Die Aufgabe des DBManager ist die Verwaltung der Datenbankzugriffsdaten. Derzeit sind folgende Datenbanken eingebunden: Oracle, Sybase und MS SQL.

Für die Unterstützung von zusätzlichen Datenbanken muss eine weitere Subklasse von DBDataSet erstellt werden, welche den JDBC Treiber für die neue Datenbank verwendet (siehe

Abbildung 53).

In der abstrakten Klasse DBDataSet befinden sich alle benötigten Methoden zur Verbindungserstellung und zum Speichern der Zugangsdaten. In den Subklassen werden die entsprechenden JDBC Treiber verwendet, mit denen man auf die Datenbank zugreifen kann.

Streng nach dem MVC-Prinzip sind Model (DBDataSet), Controller (DBConnectionController) und View (DBDataSetView) getrennt.

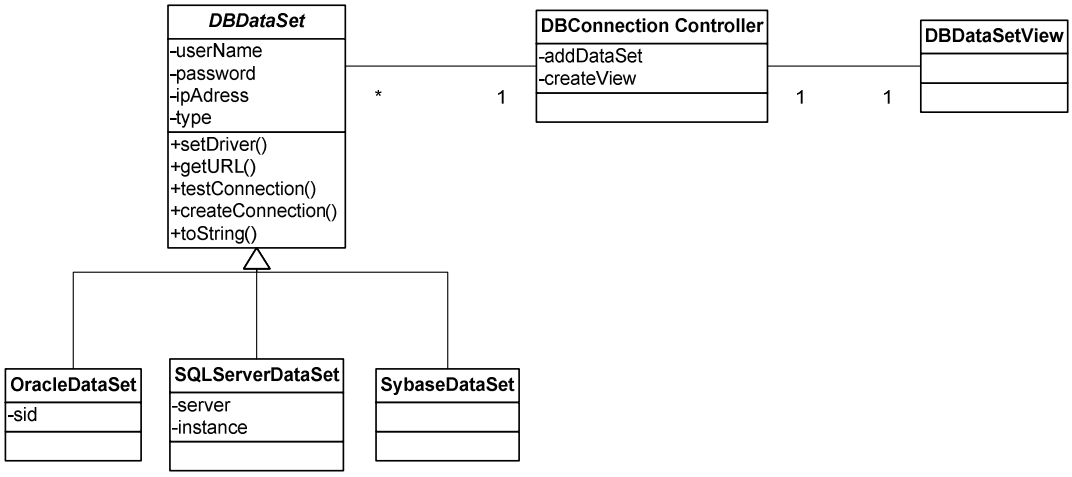

Abbildung 53: DBManager

8.2 VIT (Visual Integration Tool)

Das VIT Package ist das Haupt-Package, welches die Funktionalität zur Transformation der Schemata, zur Darstellung der Schemata und zur Erstellung von Mappings enthält.

Der WorkbenchController ist die Kernkomponente des VIT. Er steuert die Hauptaufgaben, welche vom Visual Integration Tool gelöst werden müssen. Teilaufgaben werden an die anderen Controller weitergeleitet. Wie auch beim DBManager gibt es eine strenge Trennung zwischen Model, Controller und View nach dem MVC-Prinzip.

Nachfolgend werden die einzelnen Komponenten näher beschrieben.

8.2.1 Controller

Beim Design der Controller wurde auf die in Kapitel 3.3.3 vorgestellten Referenzarchitekturen Rücksicht genommen.

Das VIT Package beinhaltet folgende Controller:

- WorkbenchController
- TransformationController
- FigureController
- MappingAssistent

8.2.1.1 WorkbenchController

Der WorkbenchController übernimmt die Aufgabe der Verwaltung der Workbench-Oberfläche, dem zentralen Fenster des VIT. Der WorkbenchController ist zuständig für die Verwaltung der Menüfunktionen, wie Anlegen einer neuen Datenbankverbindung, Laden eines Schemas aus einem lokalen Data Warehouse, Starten des Mapping-Assistenten und Exportieren der Daten in das Repository. Zusätzlich dient er dabei als zentrale Schnittstelle zu den anderen Controllern des VIT. Der WorkbenchController liest die Daten eines angebundenen Quellsystems aus und speichert die relationale Struktur des Schemas. Der WorkbenchController übernimmt auch die Aufgabe der Positionsbestimmung der Elemente des UML Diagramms (vgl. Kapitel 7.1.3.2). Beim Export der Zieldaten werden die SQL Statements für das Update oder die Erstellung der Datensätze im Repository generiert.

8.2.1.2 TransformationController

Der TransformationController übernimmt die Umwandlung der relationalen Schemata in ein dimensionales Schema. Dies entspricht den in Kapitel 3.3.3.2 vorgestellten Prozessoren. Dabei werden dem TransformationController die Daten, welche aus den relationalen Datenbanken der lokalen Data Warehouses ausgelesen wurden, übergeben. Anschließend wird das relationale Datenbankschema in ein dimensionales Schema mit Fakten und Dimensionen umgewandelt. Zusätzlich übernimmt der TransformationController den Aufbau der Struktur für die Baumübersicht (vgl. Kapitel 7.1.3.1).

8.2.1.3 FigureController

Der FigureController übernimmt die Verwaltung der Figure-Elemente (Fakt, Dimension) für das UML Diagramm und deren Assoziationen. Er enthält die Funktionen für die Positionierung der Figure-Elemente im Zeichenbereich und die Funktion zum Selektieren von Elementen für den Mapping-Vorgang.

8.2.1.4 MappingAssistant

Mit dem MappingAssistant können Mappings definiert werden (vgl. Kapitel 7.1.4), welche dann zu den Schemata gespeichert werden können. Der MappingAssistent hat eine Schnittstelle zum FigureController und bekommt von diesem die selektierten Elemente geliefert. Der MappingAssistent hat zusätzlich eine

Schnittstelle zum Transformationscontroller von dem er die Daten der selektierten Elemente bekommt.

8.2.2 Model

Die Data Warehouse Schemata werden auf zwei Arten gespeichert, einerseits relational, wie sie aus dem Data Warehouse ausgelesen werden, andererseits dimensional, wie sie für das Mapping benötigt werden.

Die Daten in den lokalen Data Warehouses liegen im Star-Schema vor (siehe Kapitel 2.5.2.1). Diese werden beim Importvorgang in ein relationales Datenmodell (siehe

Abbildung 54) gespeichert.

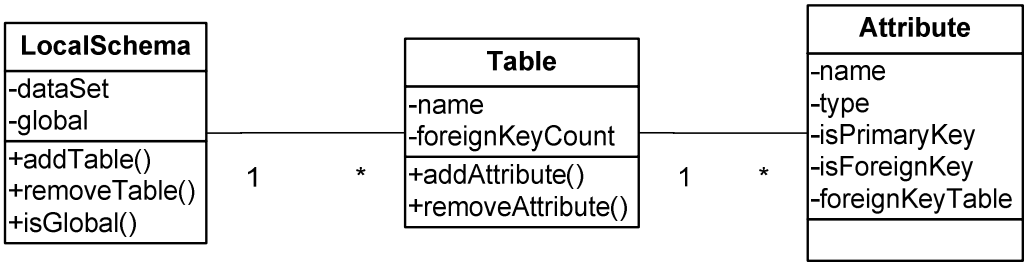

Abbildung 54: Relationales Schema

Anschließend wird das relationale Modell mit Hilfe des Transformationsprozessors in ein dimensionales Modell transformiert, welches für die Darstellung der Schemata und das Mapping benötigt wird (siehe Abbildung 55). Dabei wird das ausgelesene relationale Schema in ein Komponentenschema mit Fakten und deren Merkmalen und Dimensionen mit deren Levelattributen transformiert und als Baumstruktur (vgl. Kapitel 7.1.3.1) gespeichert. Für die Darstellung als UML Diagramm (vgl. Kapitel 7.1.3.2) wird die Baumstruktur in ein UML Schema transformiert.

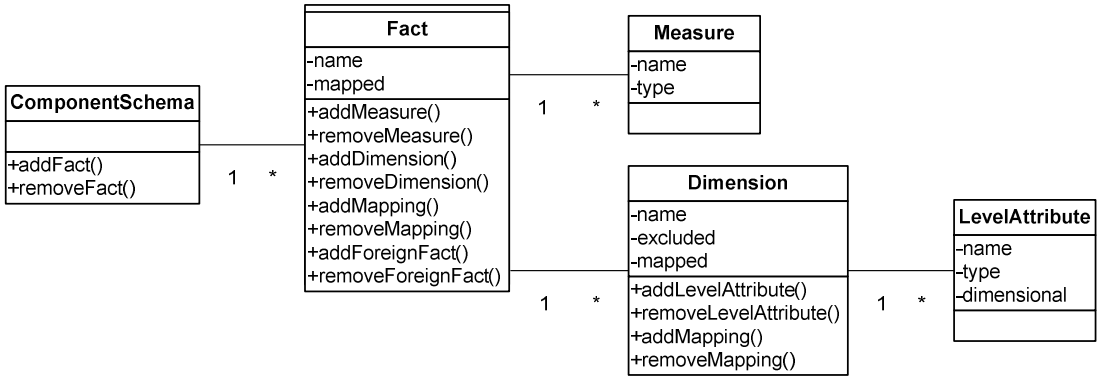

Abbildung 55: Dimensionales Schema

8.2.3 View

Die jeweiligen Controller haben eine eigene View-Klasse (LoadSchemaView, WorkbenchView, TransformationView und MappingAssistentView) zugeordnet.

Der LoadSchemaView dient zur Darstellung der Eingabemaske für neue Datenbankverbindungen (siehe Abbildung 33). Der WorkbenchView dient zur Darstellung der gesamten Oberfläche des VIT (siehe Abbildung 36). Der TransformationView dient zur Darstellung und Manipulation der Baumstruktur (siehe Abbildung 37). Der MappingAssistentView dient zur Darstellung der Eingabemaske des Mapping-Assistenten (siehe Abbildung 44). Für die grafische Oberfläche wurden die SWT (The Standard Widget Toolkit) Libraries für Eclipse verwendet (vgl. [TSWT11]). Die Darstellung der UML Diagramme erfolgt mit Hilfe der Draw2D Komponente des GEF Frameworks und dessen Libraries (vgl. [EGEF10]).

9 Fazit und Ausblick

Diese Arbeit hat gezeigt, wie die Integration von autonomen Data Warehouses in ein föderiertes Data Warehouse System realisiert werden kann. Dabei wurde das Visual Integration Tool (VIT) als Teil eines Gesamtsystems (siehe Abbildung 1) entwickelt. Es wurden die theoretischen Hintergründe zu Data Warehousing und zu föderierten Systemen und die verschiedenen Formen der Quellschemata (vgl. Kapitel 2.5.2) erläutert. Das Problem bei der Integration von autonomen Data Warehouses zu einem föderierten Data Warehouse System ist die Unterschiedlichkeit der Quellschemata. Zur Lösung dieses Problems wurden verschiedene Integrationsstrategien und Referenzarchitekturen vorgestellt (vgl. Kapitel 3.4). Die dabei auftretenden relationalen Konflikte wurden in dieser Arbeit erläutert (vgl. Kapitel 3.4.3).

Es wurde der Data Warehouse Standard CWM (vgl. Kapitel 4) vorgestellt und die Einhaltung des Standards für die Schnittstelle zum SQL-MDi Query Parser (vgl. [Brunn08]) über das Metadaten-Repository gewährleistet. Weiters wurden verschiedene Ansätze zur visuellen Datenintegration vorgestellt, welche eine Schemaintegration mit Hilfe eines grafischen Tools ermöglichen sollen (vgl. Kapitel 5).

Im letzten Abschnitt, der sich mit dem VIT beschäftigt, wird erläutert, wie die Realisierung der Anforderungen an das Tool erfolgt ist. Das VIT wurde aufgrund von Empfehlungen in der Literatur und vorhandenen Ressourcen in der Programmiersprache Java realisiert (vgl. Kapitel 6.2). Das VIT gliedert sich in zwei Komponenten. Die erste Komponente (DBManager) dient zur Anbindung von Quellsystemen, aus welchen die Schemata der autonomen Data Warehouses gelesen werden. Derzeit werden die Datenbanken Oracle, Sybase und MS SQL unterstützt. Die zweite Komponente (VIT) dient zur Darstellung der Schemata (Star-Schema) und zum Erstellen von Mappings zwischen globalem Schema und Schemata der autonomen Data Warehouses. Für die Darstellung der Schemata wurden einerseits UML Diagramme und andererseits eine Baumstruktur zur Übersicht verwendet (vgl. Kapitel 7.1.3). Zur Integration der autonomen Data Warehouses wird eine One-Shot-Integrationsstrategie (vgl. Kapitel 3.4.1.1) verwendet. Für die Mappings zwischen globalem Schema und Schemata der

autonomen Data Warehouses werden von Brunneder definierte Operatoren zur Lösung von Integrationskonflikten verwendet (vgl. [Brun08]).

Nachfolgend wird auf die Erweiterungsmöglichkeiten und die nicht behandelten Themen des Visual Integration Tools (VIT) eingegangen.

Derzeit wird beim Auslesen der Schemata nur das Star-Schema (vgl. Kapitel 2.5.2.1) berücksichtigt. Es besteht natürlich die Möglichkeit, dass die Schemata als Snowflake-Schema (vgl. Kapitel 2.5.2.2) vorliegen. Dieser Bereich wäre eine Möglichkeit, das VIT zu erweitern, um somit ein breiteres Spektrum an Importmöglichkeiten abzudecken. Damit wäre auch die Integration von Data Warehouses in ein föderiertes Data Warehouse möglich, deren Schemata als Snowflake-Schema vorliegen.

Eine weitere Möglichkeit zur Erweiterung bietet der DBManager, welcher dank der strukturierten Gliederung eine Anbindung zusätzlicher Datenbanken ermöglicht, um so mehrere Quellsysteme zu verwalten.

Bei der Darstellung der Schemata als UML Diagramme wäre eine Implementierung weiterer Layoutalgorithmen möglich. Damit wäre bei einer größeren Anzahl an Fakten und Dimensionen eine bessere Übersichtlichkeit und Verwaltung der UML Elemente möglich.

Diese Arbeit geht nur auf die Integration von Data Warehouse Schemata, nicht aber auf die Verwaltung von Data Marts (vgl. Kapitel 2.3), ein. Dies wäre ein Bereich, in dem sich die Forschung weiterentwickeln könnte.

Diese Arbeit hat gezeigt, dass es im Bereich der visuellen Integrationstools für föderierte Data Warehouses noch Forschungspotenzial gibt. Der derzeitige Markt ist großteils wissenschaftlicher Natur und es gibt wenige kommerzielle Tools (vgl. Kapitel 5). In der heutigen Zeit ist aufgrund der Globalisierung und des Zusammenschlusses mehrerer Firmen eine globale Analyse der Daten von hoher Wichtigkeit. Dabei ist es wichtig, dass man beim Zusammenschluss mehrerer Firmen das Rad nicht jedes Mal neu erfinden muss, sondern auf bereits existierende Data Warehouses aufbauen kann und diese zu einem gemeinsamen Data Warehouse föderiert. Schnell adaptierbare Systeme helfen kurzfristig auf Änderungen der Marktsituation reagieren zu können.

10 Literaturverzeichnis

[Abdu09] *Abdulghani A.:* Computation of OLAP Data Cubes, IGI Global, Quantiva, 2009

[Alto07] *Altova GmbH:* MapForce Data Sheet, 2007,

URL: http://www.altova.com/documents/MapForcedatasheet.pdf

[Stand 17.12.07]

[Amat07] AmaterasUML, URL:

http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=AmaterasUML

[Stand 10.01.07]

[Argo07] Argo UML, URL: http://argouml.tigris.org/

[Stand 10.01.07]

[BaLN86] *Batini C., Lenzerini M., Navathe S.B.:* A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, 1986

[BeSc06] *Berger S, Schrefl M.:* Extended Syntax Definition for SQL MDi, Linz, 2006

[Blue07] Blueprint Software Modeler,
URL: http://www.atportunity.com/WebContent/english/home/home.html
[Stand 10.01.07]

[Brun08] *Brunneder W.:* Entwicklung eines Parser für SQL-MDi, eine multidimensionale Abfragesprache für Föderierte Data Warehouse-Systeme, Institut für Wirtschaftsinformatik – Data & Knowledge Engineering, Linz, 2008

[BuFN94] *Busse, Fankhauser, Neuhold:* Federated Schema in ODMG, Integrated Publication and Information Systems Institute, GMD IPSI, Darmstadt, Germany, 1994

[Claus98] *Clausen N.:* OLAP-Multidimensionale Datenbanken, Addison-Wesley-Longman, Bonn, 1998

[CoCS93] *Codd E. F., Codd S. B., Salley:* Providing OLAP to User-Analysts: An IT Mandate; Codd & Associates; 1993 URL: http://dev.hyperion.com/resource_library/white_papers/providing_olap_to_user_analysts.pdf
[Stand 10.01.07]

[Conr97] *Conrad S.:* Föderierte Datenbanksysteme: Konzepte der Datenintegration; Springer; Berlin/Heidelberg/New York; 1997

[CWD407] CWD4all, URL: http://www.cwd4all.com/

[Stand 10.01.07]

[CWMI07] CWM Interfaces including JMI,

URL: http://dev.hyperion.com/download/code_library/cwm_javainterfaces.cfm

[Stand 10.01.01]

[DBEx07] DBExplorer,
URL: http://www.diligent-it.com/products/dbexplorer/index_dbexplorer.htm
[Stand 10.01.07]

[DBSc07] DB Schema Viewer URL: http://dbschemaviewer.sourceforge.net/
[Stand 10.01.07]

[DeEa05] *Dehne, Eavis, Rau-Chaplin:* Parallel Querying of ROLAP Cubes in the Presence of Hierarchies, ACM 1595931627/05/0011, 2005

[EGEF10] Eclipse Graphical Editing Framework
URL: http://www.eclipse.org/gef/
[Stand 06.06.2010]

[ESQL07] Eclipse SQL Explorer, URL: http://eclipsesql.sourceforge.net/
[Stand 10.01.07]

[EUML07] Eclipse UML & Eclipse UML Studio Edition,
URL: http://www.omondo.com/
[Stand 10.01.07]

[GaSC95] Garca-*Solaco, Saltor, Castellanos:* A Structure Based Schema Integration Methodology, IEEE Computer Society Press, 1995

[GoMa98] *Golfarelli, Maio, Rizzi:* The Dimensional Fact Model: A Conceptual Model for Data Warehouses, Bologna, 1998

[InHa94] *Inmon, Hackathorn:* Using the Data Warehouse; John Wiley & Sons; New York; 1994

[KiSe91] *Kim W., Seo J.:* Classifying Schematic and Data Heterogeneity in Multidatavase Systems, IEEE Computer, 1991

[Lust02] *Lusti M.:* Data Warehousing und Data Mining: Eine Einführung in entscheidungsunterstützende Systeme; 2. Auflage; Springer; Berlin/Heidelberg/New York; 2002

[LuTI02] *Luján-Mora, Trujillo, Il-Yeol Song:* Multidimensional Modeling with UML Package Diagrams, ER, 2002

[LuVT04] *Luján-Mora, Vassiliadis, Trujillo:* Data Mapping Diagrams for Data Warehouse Design with UML, ER, 2004

[Mais09] *Maislinger, L.:* Grafisches Werkzeug zur Integration von Data Marts, Institut für Wirtschaftsinformatik - Data & Knowledge Engineering, 2009

[Manh08] *Manhart K.:* BI-Methoden (Teil 1): Ad-hoc Analysen mit OLAP, Tecchannel, 2008

URL: http://www.tecchannel.de/server/sql/1751285/index2.html

[Stand 18.09.08]

[ÖzVa91] *Öszu, Valduriez:* Distributed Database Systems: Where Are We Now?, IEEE Computer, 1991

[PCTM02] *Poole, Chang, Tolbert, Mellor:* Common Warehouse Metamodel; Johny Wiley & Sons, Inc., New York; 2002

[PCTM03] *Poole, Chang, Tolbert, Mellor:* Common Warehouse Metamodel Developer's Guide; Wiley Publishing, Inc.; Indianapolis, Indiana; 2003

[Rahm11] *Rahm E:* Architektur von Data Warehouse Systemen, Universität Leipzig

URL: http://dbs.uni-leipzig.de/file/dw-kap2.pdf

[Stand: 20.02.2011]

[Reen03] *Reenskaug T.M.H.:* The Model-View-Controller (MVC), University of Oslo, 2003

[RePG95] *Reddy, Prasad, Gupta:* Formulating Global Integrity Constraints During Derivation of Global Schema, Data & Knowledge Engineering, 1995

[Ritc04] *Ritchey T.:* Strategic Decision Support using Computerised Morphological Analysis, Department for Technology Foresight and Assessment, Sweden, 2004

[Ross08] *Rossgatterer T.:* Entwicklung eines Query Prozessors in einem Föderierten Data Warehouse System, Institut für Wirtschaftsinformatik – Data & Knowledge Engineering, Linz, 2008

[RPRG94] Reddy *M.P., Prasad, P.G. Reddy P.G., Gupta:* A Methodology for Integration of Heterogeneous Databases, IEEE Transactions on Knowledge and Data Engineering, 1994

[Schm95] *Schmitt I.:* Flexible Integration and Derivation of Heterogeneous Schemata in Federated Database Systems, Preprint 10, Fakultät für Informatik , Universität Magdeburg, 1995

[ScSa96] *Schmitt, Saake:* Integration of Inheritance Trees as Part of View Generation for Database Federations, Proc. of 15th Int. Conf. on Conceptual Modelling, Springer Verlag, 1996

[ShLa90] *Sheth A. P, Larson J. A.:* Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv,* 1990
URL: http://doi.acm.org/10.1145/96602.96604
[Stand 10.01.07]

[Silv08] *Silvers F.:* Building and Maintaining a Data Warehouse, CRC Press, Boca Raton, 2008

[SpPa94] *Spaccapietra, Parent:* View Integration: A Step Forward in Solving Structural Conflicts, IEEE Transactions on Knowledge and Data Engineering, 1994

[SpPD92] *Spaccapietra S., Parent C., Dupont Y.:* Model Independent Assertions for Integration of Heterogeneous Schemas, VLDB Journal, 1992

[TsKl78] *Tsichritzis, Klug:* The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems, Information Systems, 1978

[TSWT11] SWT: The Standard Widget Toolkit, URL: http://www.eclipse.org/swt/
[Stand 24.06.2011]

[Usei07] useitGenerator, URL: http://useitgenerator.sourceforge.net/index.html
[Stand 10.01.07]

[W3CR10] W3C Recommendation Extensible Markup Language (XML)
URL: http://www.w3.org/TR/REC-xml/

11 Abbildungsverzeichnis

12 Tabellenverzeichnis