JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Sozial und Wirtschaftswissenschaftliche
Fakultät

Integration von Data Marts in ein globales Data Warehouse mit hetero-homogenem Schema

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften
(Mag.rer.soc.oec)

im Diplomstudium

WIRTSCHAFTSINFORMATIK

Eingereicht von:
Daniel Sierninger

Angefertigt am:
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

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

Mitwirkung:
Mag. Christoph Schütz

Linz, Juni 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit mit dem Titel „Integration von Data Marts in ein globales Data Warehouse mit hetero-homogenem Schema“ selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz im Mai 2011 Daniel Sierninger

Zusammenfassung

Mit zunehmender Verbreitung der Informationstechnologie fallen immer größere Datenmengen an, die oftmals in autonomen, heterogenen und häufig auch verteilten Datenquellen gespeichert sind. Da diese Daten oftmals über einzelne Abteilungen oder Unternehmen verteilt gespeichert sind, müssen sie in weiterer Folge auf Unternehmens- bzw. Konzernebene wieder in ein gemeinsames Schema zusammengeführt werden. Dies ist Aufgabe der Daten-Integration. Die Daten-Integration kombiniert die Daten verschiedener Datenquellen und stellt dem Benutzer eine einheitliche Sicht darauf zur Verfügung. Ein Problem dabei ist jedoch, dass die Quellen zumeist autonom voneinander sind. Die Daten der einzelnen Quellen weisen daher in den meisten Fällen Heterogenitäten in Form von unterschiedlichen Datenmodellen, Abfragesprachen und Schemata auf.

Data Warehouses integrieren Daten aus verschiedenen, operativen Datenquellen und stellen sie dem Benutzer für Analysezwecke beziehungsweise als Entscheidungshilfe zur Verfügung. Data Marts sind ein kleiner Teil eines Data Warehouses. Sie beinhalten nur jenen Teil der Daten, der von einer bestimmten Abteilung oder einem bestimmten Funktionsbereich benötigt wird. Bei der Integration von Data Marts in ein globales Data Warehouse können bei den Daten Heterogenitäten unter anderem in Bezug auf die Namensgebung, die dimensionalen und nicht-dimensionalen Attribute der Dimensionen und hinsichtlich der Kennzahlen der Cubes auftreten. Zur Abbildung der Heterogenitäten haben [Neumayr et al. 2010] das Konzept der hetero-homogenen Dimensionen und Cubes eingeführt. Hetero-homogene Hierarchien sind in Bezug auf die Sub-Hierarchien und Sub-Cubes heterogen und homogen hinsichtlich eines minimalen, gemeinsamen Schemas, das von allen Sub-Hierarchien und Sub-Cubes geteilt wird [Neumayr et al. 2010].

Diese Arbeit stellt eine Erweiterung der Prototyp-Implementierung von [Schütz 2010] um die Möglichkeit zur Integration von Data Marts in ein globales Data Warehouse mit hetero-homogenem Schema vor. Dabei wird zur Speicherung der Daten sowohl für das Data Warehouse als auch für den Data Mart eine objekt-relationale Datenbank von Oracle verwendet. Zur Abbildung der Heterogenitäten werden Mappings zur Verfügung gestellt, die vom Benutzer vor dem Import des Data Marts zu erstellen sind. Die Integration der Daten ist semi-automatisch. Zwar ist der Importvorgang selbst automatisch, jedoch sind vom Benutzer vorher die Mappings zu erstellen und der Importvorgang zu starten. Zudem muss der Benutzer bei Konflikten und Fehlern eingreifen.

Abstract

The rise of information technology in the last few years has been causing larger amounts of data to be processed in corporate databases and information systems. This data is often stored in autonomous, heterogeneous and distributed sources and has to be integrated again, to enable the users to work with the data. This process is called data integration. Data integration combines data from various sources and provides the user a mediated, global view on it. A big problem in data integration is that the sources are often autonomous, leading to structural heterogeneities like different data models, query languages and different schemas.

Data Warehouses integrate data from different, operative data sources and provide it to the user for analysis and as decision support. Data Marts are extracts of data warehouses which contain the part of the data that is needed in a certain division of a company.

Data Integration has to cope with heterogeneities in the data, i.e. in respect to the naming, the dimensional and non-dimensional attributes in dimensions and the facts of a cube. To model heterogeneities like different non-dimensional attributes or diverse facts. [Neumayr et al. 2010] introduced hetero-homogenous hierarchies and cubes. They adapted the M-Objects and M-Relationships and introduced M-Cubes to accommodate structural and semantic heterogeneities. Hetero-homogenous hierarchies are homogenous because they share a minimal, common schema. They are heterogeneous with respect to their sub-hierarchies.

This thesis shows a prototype implementation which integrates Data Marts in a global Data Warehouse with hetero-homogeneous schema. The system uses an Oracle object-relational database to store the data of the Data Marts and the Data Warehouse, and provides mappings to describe the heterogeneities between the global and the local schema. These mappings have to be defined by the user, before the Data Mart can be integrated. The integration process is semi-automatic. The user has to define the mappings and to start the process manually. The import of the dimensions and the cube itself is done automatically.

Inhaltsverzeichnis

1. Einleitung ... 1

1.1. Aufgabenstellung & Zielsetzung ... 3

1.2. Aufbau der Arbeit ... 5

2. State of the art ... 6

2.1. Data Warehouses ... 6

2.2. Objekt-relationale Datenbanken ... 13

2.3. Integration von Daten ... 15

3. Data Warehouses mit hetero-homogenen Hierarchien und Cubes ... 22

3.1. Motivation ... 22

3.2. Hetero-homogene Dimensionen ... 23

3.3. Hetero-homogene Cubes ... 26

3.4. Prototyp eines hetero-homogenen Data Warehouses ... 30

4. Integrationsprozess ... 36

4.1. Problemstellung - Beispielszenario ... 37

4.2. Vorgehensweise ... 41

4.3. Definition von Mappings ... 43

4.3.1. Mapping Übersicht ... 43

4.3.2. Definition der Mappings ... 47

4.4. Import von Dimensionen ... 53

4.5. Import eines M-Cube ... 54

4.6. Mapping Beispiel ... 55

4.7. Dimensionsimport - Sonderfälle ... 59

4.7.1. Zusätzliche Levels ... 60

4.7.2. Fehlende Levels ... 63

4.7.3. Zusätzliche Attribute ... 65

4.7.4. Äquivalente Objekte ... 65

4.8. M-Cube-Import - Sonderfälle ... 68

4.8.1. Zusätzliche Dimensionen ... 69

4.8.2. Fehlende Dimensionen ... 71

4.8.3. Zusätzliche Measures ... 73

4.8.4. Unterschiedliche Granularität der Kennzahlen ... 73

4.8.5. Äquivalente M-Relationships ... 74

4.8.6. Pivoting ... 77

4.8.7. Unpivoting 79
4.9. Einschränkungen des Importprozesses 81
5. Implementierung des Importprozesses 83
5.1. Speicherung der Mappings 83
5.1.1. Mappings beim M-Object 84
5.1.2. Mappings beim M-Cube 87
5.1.3. Mappings bei den M-Relationships 89
5.2. Import der Dimensionen 92
5.2.1. Importieren eines M-Objects 94
5.2.2. Setzen der Attributwerte 98
5.3. Import des M-Cubes 100
5.4. Automatisch abgeleitete Mappings 104
5.4.1. M-Object-Mappings 104
5.4.2. M-Relationship-Mappings 105
5.4.3. Mappings aus Parametern der Import Methode 105
5.4.4. Sonstige Mappings 106
5.5. Anpassen der Koordinaten 107
5.6. Hinzufügen von Levels zum globalen Schema 107
5.7. Hilfsmethoden 109
6. Ergebnisse 110

Abbildungsverzeichnis

Abbildung 1: Data Warehouse Architektur [Goeken 2006] 8

Abbildung 2: Multidimensionales Datenmodell [Marx Gómez et al. 2009] 13

Abbildung 3: Architektur eines Daten-Integrationssystems [Levy 2000] 17

Abbildung 4: Hetero-homogene Produktdimension [Neumayr et al. 2010] 25

Abbildung 5: Data Warehouse mit M-Relationships [Neumayr et al. 2010] 27

Abbildung 6: Konkretisierung von M-Relationships [Neumayr et al. 2010] 29

Abbildung 7: M-Object und Dimensions Architektur [Schütz 2010], Änderungen nach [Schütz 2011] 32

Abbildung 8: M-Cube und M-Relationship Architektur [Schütz 2010], Änderungen nach [Schütz 2011] 34

Abbildung 9: Problemstellung - Produktdimension 38

Abbildung 10: Problemstellung - Dimension Ort 38

Abbildung 11: Problemstellung - Dimension Zeit 39

Abbildung 12: Problemstellung - M-Cubes 40

Abbildung 13: Mappings Problemstellung – Produkt Dimension 55

Abbildung 14: Mappings Problemstellung – Zusätzliche Attribute 57

Abbildung 15: Problemstellung - Importierte Dimension Musik 58

Abbildung 16: Ortsdimension - Ausgangspunkt 60

Abbildung 17: Ortsdimension - Mappings 61

Abbildung 18: Importierte Ortsdimension 62

Abbildung 19: Ortsdimension mit fehlenden Levels 64

Abbildung 20: Importierte Ortsdimension mit fehlenden Levels 64

Abbildung 21: M-Object Mapping Beispiel - Ausgangspunkt 66

Abbildung 22: M-Object-Mapping Beispiel - Importierte Dimension 68

Abbildung 23: M-Cube mit mehr Dimensionen 69

Abbildung 24: Data Mart mit weniger Dimensionen 71

Abbildung 25: M-Cube mit äquivalenten M-Relationships 75

Abbildung 26: Pivoting Beispiel 78

Abbildung 27: Unpivoting Beispiel 80

Abbildung 28: mobject_ty 85

Abbildung 29: mcube_*_ty 88

Abbildung 30: mrel_*_ty 90

Tabellenverzeichnis

Tabelle 1: M-Object Mappings 47

Tabelle 2: M-Cube Mappings 49

Tabelle 3: M-Relationship Mappings 50

Tabelle 4: M-Object-Tabelle mit Level-Mappings 86

Tabelle 5: M-Object-Tabelle mit Attribut-Mappings 86

Tabelle 6: M-Object-Tabelle mit M-Object-Mappings 87

Tabelle 7: M-Cube Tabelle mit Dimension-Mappings 88

Tabelle 8: M-Cube Tabelle mit Hidden-Dimension-Mappings 89

Tabelle 9: M-Relationship-Tabelle mit Measure-Mapping 91

Tabelle 10: M-Relationship-Tabelle mit M-Relationship-Mappings 91

Tabelle 11: M-Relationship-Tabelle mit Pivot Mappings 92

Tabelle 12: M-Relationship-Tabelle mit Unpivot Mappings 92

1. Einleitung

Aufgrund des rasanten Aufstiegs und der damit verbundenen immer größer werdenden Bedeutung der Informationstechnologie hat man es mit immer größeren Datenmengen zu tun, die vielfach in autonomen, heterogenen und oftmals auch verteilten Datenquellen gespeichert sind. Die Integration von Daten ist dadurch zu einem wichtigen Thema in verteilten Datenbanken, Informationssystemen, Data Warehouses und Web Data Management geworden. [Calì et al. 2004]

Aufgabe der Daten-Integration ist es, die Daten verschiedener Datenquellen zu kombinieren und dem Benutzer eine einheitliche Sicht darauf zur Verfügung zu stellen [Lenzerini 2002, S. 233]. Die einheitliche Sicht auf die integrierten Daten wird im Allgemeinen als globales Schema bezeichnet [Calì et al. 2004], [Levy 2000].

Ein Problem bei der Integration von Daten ist, dass die Quellen zumeist autonom voneinander sind, das heißt dass sie bereits vorher unabhängig voneinander existieren [Levy 2000]. Die Daten der einzelnen Quellen weisen daher in den meisten Fällen Heterogenitäten auf, wobei zwischen strukturellen und semantischen Heterogenitäten unterschieden werden kann [Koch 2001]. Strukturelle Heterogenitäten beziehen sich unter anderem auf das Datenmodell, die Abfragesprache und die Protokolle. Semantische Heterogenitäten betreffen Unterschiede zwischen den Schemata [Koch 2001].

Die vorliegende Arbeit beschäftigt sich mit der Problematik der Integration von Daten aus verschiedenen Data Warehouses beziehungsweise Data Marts in ein globales Data Warehouse. Data Warehouses integrieren bereits Daten aus verschiedenen, operativen Datenquellen und stellen diese Daten dem Benutzer für Analysezwecke beziehungsweise als Entscheidungshilfe zur Verfügung. Der Zweck des Data Warehousing besteht laut [Goeken 2006] „in der bedarfsgerechten Informationsversorgung von Entscheidungsträgern, also darin, entscheidungsrelevante Daten in aufbereiteter Form kompakt, konsistent und leicht zugänglich zur Verfügung zu stellen“ [Goeken 2006, S. 22]. Um die Analysierbarkeit zu erhöhen stellen Data Warehouses Fakten und Kennzahlen zur Verfügung. Ein Fakt ist eine numerische Messgröße. Es handelt sich dabei um eine Basiskennzahl, die einen betriebswirtschaftlichen Sachverhalt aus der realen Welt beschreibt [Golfarelli et al. 1998, S. 218]. Kennzahlen sind verdichtete numerische Kenngrößen, die aus den Fakten abgeleitet werden und diese aus verschiedenen Sichten, auch Dimensionen genannt, beschreiben. Meist werden Kennzahlen aus Fakten durch Anwendung arithmetischer Operationen errechnet [Marx Gómez et al. 2009]. So wären etwa der Umsatz oder die verkaufte Menge Kennzahlen.

Data Warehouses müssen riesige Mengen an Daten verarbeiten können. Hierfür werden die Daten mit Hilfe von Dimensionen speziell in einem Cube aggregiert. Dimensionen sind diskrete Attribute, die die Subjekte der betriebswirtschaftlichen Sachverhalte beschreiben. Dimensionen sind eine spezielle Sicht auf Fakten und Kennzahlen [Golfarelli et al. 1998, S. 218]. Sie geben ihnen somit eine Bedeutung [Marx Gómez et al. 2009]. Andernfalls wären Fakten und Kennzahlen lediglich Zahlen. Typische Dimensionen für die Kennzahl Umsatz wären Zeit, Ort und Produkt [Golfarelli et al. 1998, S. 218].
Ein Schwerpunkt der Arbeit ist die Integration von Daten von Data Marts. Ein Data Mart ist ein struktureller und/oder inhaltlicher Extrakt eines Data Warehouses. Er beinhaltet also nur einen kleinen Teil der Daten, der auf die Bedürfnisse einer bestimmten Abteilung oder eines bestimmten Funktionsbereiches zugeschnitten ist [Goeken 2006].

Das zugrunde liegende Data Warehouse beziehungsweise die verwendeten Data Marts sind nach dem hetero-homogenen Konzept von [Neumayr et al. 2010] organisiert. [Neumayr et al. 2010] haben hetero-homogene Hierarchien eingeführt. Dabei handelt es sich um Hierarchien, die ein minimales, gemeinsames Schema teilen, wo die Sub-Hierarchien und Sub-Cubes allerdings spezialisiert werden und somit heterogen in Bezug auf das globale Schema sein können [Neumayr et al. 2010]. Es können daher heterogene Daten unter einem gemeinsamen, homogenen Schema abgebildet werden. Dies erlaubt Heterogenitäten in den Daten, wie zusätzliche nicht-dimensionale Attribute oder zusätzliche Kennzahlen, abzubilden. Für die Modellierung von hetero-homogenen Hierarchien verwenden [Neumayr et al. 2010] anstatt eines normalen Cubes einen Multilevel Cube. Ein Multilevel Cube, kurz M-Cube, besteht aus einer beliebigen Anzahl von Dimension und wird mittels M-Relationships modelliert [Neumayr et al. 2010]. M-Relationships beschreiben Beziehungen von M-Objects auf verschiedenen Abstraktionsebenen.

Ein Data Warehouse kann auf zwei verschiedene Wege gebildet werden: Top-Down oder Bottom-Up. Der Top-Down Ansatz von W. H. Inmon geht davon aus, dass aus den operativen Daten des Unternehmens ein unternehmensweit, einheitliches Data Warehouse gebildet wird und daraus wiederum Data Marts für die einzelnen Abteilungen. Der Bottom-Up Ansatz von R. Kimball verwendet mehrere operative Datenbanken. Auf Basis dieser Datenbanken werden anschließend die Data Marts für die einzelnen Geschäftsbereiche beziehungsweise Abteilungen gebildet und erst aus diesen ein integriertes Data Warehouse [Breslin 2004].

Im Zuge dieser Arbeit soll ein Data Warehouse Bottom-Up erstellt werden, in dem einzelne bereits vorher existierende Data Marts integriert werden. Hierfür wird eine Erweiterung der Prototyp-Implementierung von [Schütz 2010] zur Verfügung gestellt, mit Hilfe dessen ein lokaler Data Mart in ein globales Data Warehouse mit hetero-homogenem Schema integriert werden kann. Ausgangspunkt dabei ist ein in einer objekt-relationalen Oracle Datenbank organisiertes Data Warehouse Management System von [Schütz 2011], welches sowohl die Daten des globalen Data Warehouses als auch die der zu importierenden Data Marts enthält. Die Prototyp-Implementierung eines Data Warehouse Management Systems basiert auf dem Konzept der hetero-homogenen Hierarchien von [Neumayr et al. 2010], welches es erlaubt Heterogenitäten innerhalb einer Hierarchie beziehungsweise eines Cubes abzubilden. Bei der Erweiterung der Prototyp-Implementierung von [Schütz 2010] wurde jedoch nicht das Originalsystem, sondern eine bereits weiter entwickelte Version von [Schütz 2011] verwendet.
Die Integration der Data Marts in das globale Data Warehouse erfolgt semi-automatisch. Zu Beginn muss der Benutzer die Heterogenitäten in den Daten mit den ihm zur Verfügung gestellten Mapping-Typen abbilden. Anschließend erfolgt der Import der Dimensionen und des Cubes. Der Importvorgang selbst erfolgt dabei automatisch. Die Zielsetzung und die konkrete Aufgabenstellung wird in Kapitel 1.1 nochmals genauer vorgestellt. Anschließend wird in Kapitel 1.2 der Aufbau der Arbeit beschrieben.

1.1. Aufgabenstellung & Zielsetzung

Ziel der Arbeit ist es, ein System zur Verfügung zu stellen, mit Hilfe dessen ein lokaler Data Mart in ein globales Data Warehouse mit hetero-homogenem Schema integriert werden kann. Ausgangspunkt dabei ist der Prototyp eines Data Warehouses in einer objekt-relationalen Oracle Datenbank auf Basis des Data Warehouse Management Systems von [Schütz 2010]. Sowohl das globale Data Warehouse als auch alle zu importierenden Data Marts werden mit Hilfe dieses Prototyps erstellt. Dieser soll um die Möglichkeit zur Integration von Data Marts in ein globales Data Warehouse mit hetero-homogenem Schema erweitert werden. Der Prototyp von [Schütz 2010] basiert auf dem Konzept der hetero-homogenen Hierarchien von [Neumayr et al. 2010], welches es erlaubt, Heterogenitäten innerhalb einer Hierarchie beziehungsweise eines Cubes abzubilden. Im Zuge dieser Arbeit wurde jedoch nicht der ursprüngliche Prototyp, sondern die weiter entwickelte Version [HH-DW] von [Schütz 2011] verwendet.

Aufgabe dieser Arbeit war es, den Prototypen von [Schütz 2010] um die Möglichkeit zur Integration von Data Marts in ein globales Data Warehouse mit hetero-homogenem Schema zu erweitern. Voraussetzung dabei ist, dass der Data Mart und das Data Warehouse dasselbe Datenmodell haben. Der verwendete Data Mart ist dabei autonom, d.h. er kann folgende Heterogenitäten aufweisen:

- Unterschiedliche Namensgebung hinsichtlich Level-, Attribut-, Objekt-, Dimensions- und Kennzahl-Namen.
- Der Data Mart kann zusätzliche nicht-dimensionale Attribute einführen.
- Dimensionen des Data Marts können zusätzliche Levels haben, es können aber auch Levels des globalen Schemas fehlen.
- Der lokale M-Cube kann zusätzliche Kennzahlen einführen.
- Der lokale M-Cube kann andere Dimensionen haben.
- Es kann Schema-Instanz-Konflikte geben, die mit dem Pivot-Operator aufgelöst werden müssen (nähere Informationen zum Pivoting findet man in Kapitel 4.3.1.8).
- Die Kennzahlen des Data Marts und des globalen Data Warehouses können unterschiedliche Granularitäten haben.
- Die Kennzahlen des Data Marts können eine andere Einheit haben.

Zur Überbrückung der oben aufgelisteten, möglichen Schema-Heterogenitäten sind entsprechende Mappings zur Verfügung zu stellen, die, bevor der Import gestartet werden kann, vom Benutzer händisch zu erstellen sind.
Für den Import wurde eine Prototyp-Implementierung erstellt, welche den Integrationsprozess semi-automatisch abwickelt. Dabei ist der Importvorgang an sich zwar automatisch, die Mappings sind jedoch vorher vom Benutzer zu erstellen. Zudem muss der Benutzer bei Konflikten und Fehlern eingreifen. Die Daten des Data Marts werden dabei, entgegen herkömmlicher Daten-Integrationssysteme, auch physisch in das globale Data Warehouse integriert. Ein Query Rewriting bei OLAP-Abfragen auf das globale Data Warehouse ist deshalb nicht erforderlich. Nähere Information zum Query Rewriting befindet sich in Kapitel 2.3.

1.2. Aufbau der Arbeit

Zu Beginn der Arbeit wird in Kapitel 1 ein kurzer Überblick über den aktuellen Stand der Technik gegeben. Dabei werden die Begriffe Data Warehouse und Data Mart definiert und ihre wesentlichen Eigenschaften kurz erklärt. Da die Arbeit ein Data Warehouse in einer objekt-relationalen Datenbank verwendet, wird außerdem das Konzept der objekt-relationalen Datenbanken kurz vorgestellt. Zuletzt wird die Problematik der Integration von Daten inklusive eines oft verwendeten Ansatzes, dem Global-As-View-Ansatz, erläutert, dem auch diese Arbeit zugrunde liegt.

In Kapitel 0 wird das Konzept der hetero-homogenen Hierarchien und Cubes von [Neumayr et al. 2010] vorgestellt. Dabei wird erklärt, wie diese aufgebaut sind und welche Heterogenitäten damit abgebildet werden. Weiters wird hier der als Basis verwendete Prototyp des Data Warehouse Management Systems von [Schütz 2011] vorgestellt.

In Kapitel 1 wird der Ablauf des Integrationsprozesses beschrieben. Hier wird gezeigt, wie im ersten Schritt die Mappings zur Überbrückung der Schema-Heterogenitäten zwischen dem lokalen und dem globalen Schema definiert werden können. Danach wird beschrieben, wie die Dimensionen importiert werden können und wie im letzten Schritt der M-Cube selbst integriert werden kann. Der komplette Integrationsprozess wird dabei anhand eines konkreten Beispiels, das in Kapitel 4.1 vorgestellt wird, beschrieben. In den Kapiteln 4.7 und 4.8 wird außerdem auf einige Sonderfälle bei der Integration eines Data Marts eingegangen, und wie diese behandelt werden können. Am Ende dieses Abschnitts werden noch einige Einschränkungen angeführt, die der Integrationsprozess nicht behandeln kann.

Die technische Umsetzung der im Zuge der vorliegenden Arbeit erstellten Prototyp-Implementierung wird anschließend in Kapitel 0 beschrieben. Hier wird gezeigt wie die als Basis verwendete weiter entwickelte Version des Prototyps von [Schütz 2010] angepasst wurde, um die verschiedenen Mapping-Typen zu speichern. Außerdem wird hier der Algorithmus der Import-Methoden für den Dimensions- und den Cube-Import kurz vorgestellt. Abschließend wird auch gezeigt wie bei gewissen Spezialfällen vorgegangen wird und welche allgemeinen Hilfsfunktionen verwendet werden.

Am Ende werden in Kapitel 0 die Ergebnisse dieser Arbeit nochmals zusammengefasst und die noch offenen Punkte aufgezählt.

2. State of the art

In diesem Kapitel wird ein Überblick über den aktuellen Stand der Technik aller verwendeter Konzepte gegeben. Dazu werden die wesentlichen Begriffe definiert und alle für diese Arbeit relevanten Aspekte erläutert.

Zu Beginn werden in Kapitel 2.1 die Begriffe Data Warehouse und Data Mart definiert und deren wesentliche Eigenschaften aufgezeigt. Da die als Ausgangsbasis verwendete Erweiterung des Prototyps von [Schütz 2010] in einer objekt-relationalen Oracle Datenbank realisiert wird, wird anschließend das Konzept objekt-relationaler Datenbanken vorgestellt.

Zuletzt wird in Kapitel 2.3 die Problematik bei der Integration von Daten erläutert, wesentliche Aspekte dabei hervorgehoben und der allgemeine Ablauf der Daten-Integration näher beschrieben. Zudem wird ein gebräuchlicher Ansatz zur Integration von Daten, der Global-As-View-Ansatz, vorgestellt, der auch in dieser Diplomarbeit verwendet wird.

2.1. Data Warehouses

Für die verschiedenen operativen Teilbereiche eines Unternehmens existieren in der Regel eine Reihe von Anwendungssystemen, wie zum Beispiel für den Vertrieb oder das Finanz- und Rechnungswesen, die die Abwicklung der operativen Aufgaben unterstützen. Dem übergeordnet gibt es auch Systeme, die dem Management bei der Planung und Steuerung des Unternehmens helfen sollen [Goeken 2006]. Der Zweck des Data Warehousing besteht laut [Goeken 2006] „in der bedarfsgerechten Informationsversorgung von Entscheidungsträgern, also darin, entscheidungsrelevante Daten in aufbereiteter Form kompakt, konsistent und leicht zugänglich zur Verfügung zu stellen“ [Goeken 2006, S. 22]. Data Warehouses integrieren dabei Daten aus verschiedenen, operativen Datenquellen und stellen sie dem Benutzer für Analysezwecke beziehungsweise als Entscheidungshilfe zur Verfügung. Wesentliche Aspekte eines Data Warehouse sind demnach die Daten-Integration, die Datenanalyse und die Entscheidungsunterstützung [Navrade 2007].

Ein weiteres wichtiges Erfolgskriterium von Data Warehouses ist der Nutzen für den Anwender. Die Daten müssen schnell abgefragt werden können und für den Anwender adäquat aufbereitet werden um die Datenanalyse zu unterstützen [Chamoni & Gluchowski 2006].

Ein Data Warehouse ist laut W. H. Inmon „a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's Decision Support process" [Inmon 2005, S. 29]. Wie bereits zuvor erwähnt, integriert es die Daten aus verschiedenen Datenquellen und stellt sie dem Benutzer für Analysezwecke beziehungsweise als Entscheidungshilfe („decision support") zur Verfügung.

Die Definition von Inmon nennt vier grundlegende Eigenschaften von Data Warehouses [Inmon 2005, S. 29 - 32]:

- Themenorientierung („subject-oriented"):
  Im Gegensatz zu traditionellen, transaktionsorientierten Datenbanken, zielen Data Warehouses darauf ab, eine Entscheidungshilfe für das Management zur Verfügung zu stellen. Hierzu werden die Daten nach Themen des Unternehmens, wie beispielsweise Kunden, Absatzgebiete oder Produkte, strukturiert. Außerdem sollen berechnete Kennzahlen die Analyse der Daten erleichtern. Der Begriff Kennzahl wird am Ende dieses Kapitels ausführlich erklärt.

- Integration („integrated"):
  Data Warehouses kombinieren die Daten aus mehreren verschiedenen operativen Datenquellen in integrierter Form. Dabei gilt es im Allgemeinen, strukturelle und semantische Unterschiede zu vereinheitlichen und in einem einheitlich gestalteten System abzuspeichern.

- Historisierung („time-variant"):
  Bei den Daten wird eine zusätzliche Dimension „Zeit" eingeführt. Jeder Datensatz ist daher einem eindeutigen Zeitpunkt zugeordnet, zu dem er gültig ist. Des Weiteren werden die Daten dauerhaft über Jahre hinweg gespeichert. Es ist somit möglich Veränderungen und Entwicklungen zu erkennen.

- Beständigkeit („non-volatile"):
  Die Daten werden aus operativen Datenquellen gelesen und dauerhaft gespeichert. Die Daten werden in der Folge nicht mehr verändert. Es erfolgen meist nur lesende Zugriffe, schreibende Zugriffe sind nur in Ausnahmefällen nötig, wenn zum Beispiel Fehler beseitigt werden müssen. Dies erhöht die Nachvollziehbarkeit von Analysen.

Aus technischer Sicht stellt ein Data Warehouse eine eigenständige, von den operativen Daten losgelöste Datenbank dar, in der die Daten so gespeichert sind, damit sie der Benutzer optimal analysieren und auswerten kann [Goeken 2006].
Der Einsatz von Data Warehouses entlastet operative Datenbanken, da nicht mehr alle Abfragen direkt an das operative System gerichtet werden müssen. Zudem erfolgt durch das Data Warehouse eine Historisierung der Daten [Goeken 2006], [Inmon 2005]. Alte Daten können somit aus dem operativen Datenbestand entfernt werden, was einen Performance-Gewinn mit sich bringt [Goeken 2006].

Die Architektur eines Data Warehouses besteht laut [Goeken 2006] aus fünf verschiedenen Ebenen (vgl. Abbildung 1):

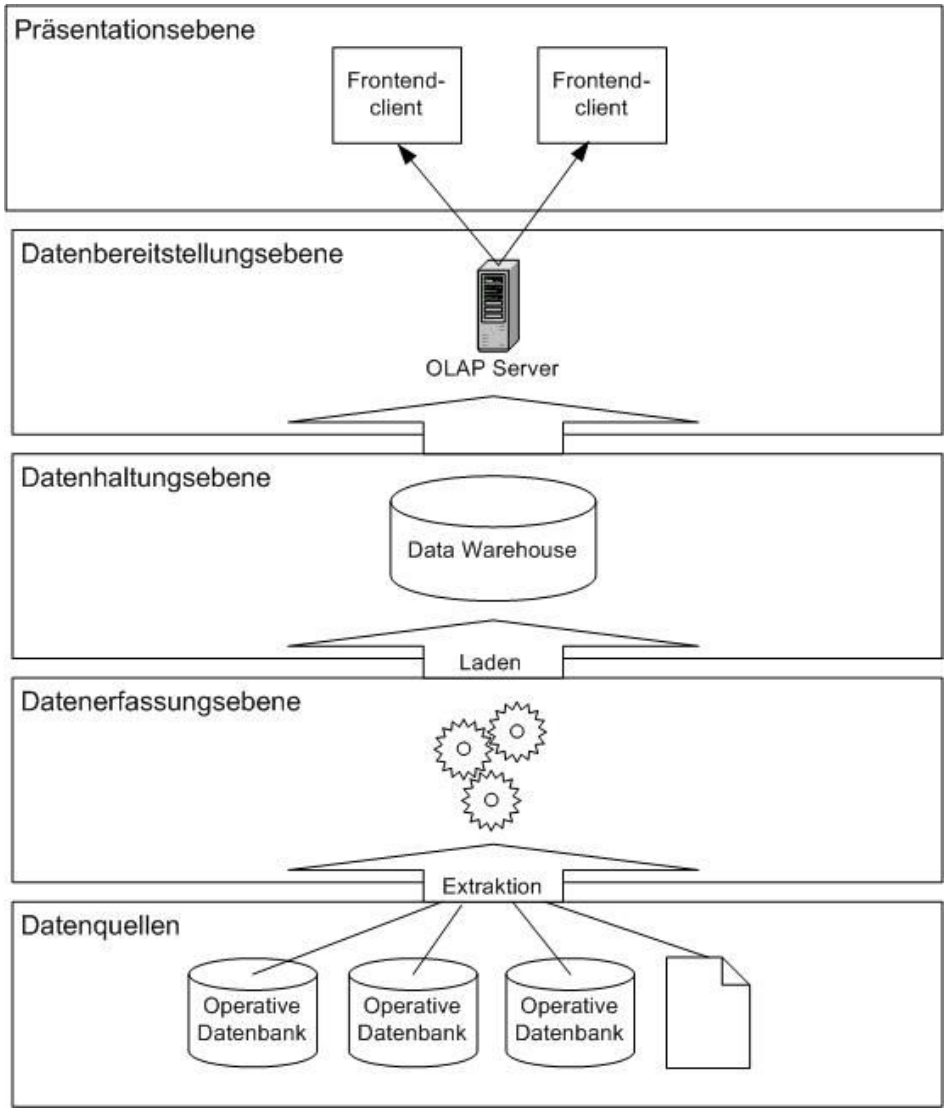

Abbildung 1: Data Warehouse Architektur [Goeken 2006]

Die Basis bildet die Ebene der Datenquellen. Hier befinden sich die operativen Datenbanken, aus denen die Daten extrahiert werden. Diese Datenquellen sind autonom, das heißt die Daten weisen semantische Heterogenitäten auf und sind strukturell unterschiedlich gespeichert. Zudem kann es sich um unterschiedliche Datenquellen, wie zum Beispiel Datenbanken oder Tabellenkalkulationsprogramme, handeln.

Die zweite Ebene ist die Datenerfassungsebene. Hier werden die Daten der untersten Ebene extrahiert, transformiert und in das Data Warehouse geladen. Dieser Prozess wird durch zahlreiche ETL-Werkzeuge (Extraktions-, Transformations- und Lade-Werkzeuge) unterstützt. Bei der Datenerfassung werden die Daten aus den einzelnen autonomen Quellen in das eigentliche Data Warehouse übertragen. Die Extraktionsphase liest die Daten aus den einzelnen Quellen. In der Transformationsphase werden die Daten homogenisiert und an das Datenmodell des Zielschemas angepasst. Hierfür müssen die Daten umgewandelt und in ein einheitliches Format gebracht werden. Auch fehlerhafte und inkonsistente Daten werden beim ETL Prozess bereinigt. Dies wird auch als „Data Cleaning“ bezeichnet. Den Abschluss bildet die Ladephase, in der die Daten ins Data Warehouse übertragen werden.

Der Kern der Architektur ist die dritte Ebene, die Datenhaltungsebene. Hier befindet sich eine Datenbank, die das eigentliche Data Warehouse repräsentiert. Auch Data Marts sind dieser Ebene zuzuordnen.

Die Datenbereitstellungsebene ist für die zweckmäßige Aufbereitung der Daten für die Endbenutzer verantwortlich [Goeken 2006]. Die Daten werden meist mit OLAP-Abfragen herausgelesen. OLAP steht für On-Line Analytical Processing. Es handelt sich dabei um „eine Software-Technologie, die qualifizierten Fach- und Führungskräften schnelle, interaktive und vielfältige Zugriffe auf relevante und konsistente Informationen ermöglichen soll“ [Chamoni & Gluchowski 2006, S. 14]. Wesentlich sind dabei dynamische, multidimensionale Abfragen auf konsolidierten, historischen Datenbeständen [Chamoni & Gluchowski 2006, S. 14]. Für die Definition von OLAP gibt es eine Reihe von Kriterien, die als funktionale Anforderungen aufzufassen sind. Ganz allgemein muss sich OLAP nach den FASMI-Kriterien von [Codd 1993] richten:

- Fast: Die Abfragen müssen sehr schnell Ergebnisse liefern. Selbst komplexe Berichte sollten in weniger als 20 Sekunden bearbeitet werden.

- Analysis: Es muss umfangreiche Analysefunktionen geben. Ein Beispiel hierfür wären die Drill-Operationen (Drill-down, Drill-across etc.).
- Shared: OLAP muss mehrbenutzerfähig sein.
- Multidimensional: Die Daten eines Data Warehouses werden in Form von Datenwürfeln, sogenannten Cubes, gespeichert. Die Multidimensionalität zielt auf eine Einordnung betrieblicher Kennzahlen entlang unterschiedlicher Dimensionen, wie Produkte, Kunden und Regionen ab, die eine geeignete Sichtweise auf die Kennzahlen zur Verfügung stellen sollen [Chamoni & Gluchowski 2006]. Eine genauere Beschreibung der Fakten, Kennzahlen, Dimensionen und Cubes folgt später.
- Information: Das System muss große Mengen von Daten aufnehmen können. Zudem müssen alle für die Analyse relevanten Daten uneingeschränkt verfügbar sein.

Die oberste Schicht eines Data Warehouses ist die Präsentationsschicht, deren Aufgabe es ist, die Daten für den Endbenutzer angemessen aufzubereiten. Die OLAP-Operationen der Datenbereitstellungsebene gewährleisten bloß die Analysierbarkeit, welche Werkzeuge und Programme zur Darstellung der Daten verwendet werden, wird jedoch erst auf Ebene der Präsentationsschicht festgelegt [Goeken 2006]. Beispielsweise können dafür Tabellenkalkulationsprogramme benützt werden.

Da oftmals nicht der gesamte Datenbestand zur Analyse und Entscheidungsfindung erforderlich ist, gibt es das Konzept des Data Marts. Ein Data Mart ist ein struktureller oder inhaltlicher Extrakt eines Data Warehouses. Er beinhaltet also nur einen kleinen Teil der Daten, der auf die Bedürfnisse einer bestimmten Abteilung oder eines bestimmten Funktionsbereiches zugeschnitten ist [Goeken 2006].
Durch die vergleichsweise kleinere Datenmenge erlaubt ein Data Mart relativ schnelle Antwortzeiten und bleibt übersichtlich. Außerdem kann er kostengünstig erstellt werden und ist in Bezug auf das globale Data Warehouse einfach zu warten [Schmidt-Volkmar 2008]. Weiters fokussieren Data Marts nur einen kleinen Teil der Funktionen und Daten.
Bei Data Marts wird in abhängige und unabhängige unterschieden. Ein Data Mart ist abhängig, wenn er nur einen Ausschnitt eines zentralen Data Warehouse darstellt, welches seine Daten aus verschiedenen Quellsystemen bezieht. Gibt es hingegen einige unabhängige Datenquellen beziehungsweise Data Marts in verschiedenen Bereichen, die erst später zu einem zentralen Data Warehouse zusammengefasst werden, so gelten diese als unabhängig. [Schmidt-Volkmar 2008]

Ein Data-Warehouse kann auf zwei unterschiedliche Wege gebildet werden: Top-Down oder Bottom-Up. [Breslin 2004] hat beide Paradigmen ausführlich verglichen.
Der erste Ansatz, der Top-Down Ansatz, basiert auf W. H. Inmon. Dabei wird aus den operativen Daten des Unternehmens ein unternehmensweit einheitliches Data Warehouse gebildet. In der Folge werden für die einzelnen Abteilungen auf Basis des Data Warehouse eigene Data Marts eingerichtet, um die Abteilungen bestmöglich bei der Entscheidungsfindung zu unterstützen [Breslin 2004]. Inmon unterteilt dazu die Architektur in vier Levels: den operativen, den auf das Data Warehouse bezogene, den abteilungsbezogenen und den individuellen Level. Der erste Level beinhaltet die operativen Datenbanken mit den Transaktionsdaten. Der zweite Level enthält das unternehmensweite Data Warehouse. Auf den abteilungsbezogenen Level werden die Daten je nach Bedarf in Data Marts zusammengefasst. Auf dem vierten Level befinden sich die individuellen Daten des Benutzers, welche bei der Analyse beziehungsweise beim Entscheidungsprozess entstehen [Breslin 2004]. Detaillierte Information zu diesem Ansatz findet man in [Inmon 2005].
Der Bottom-Up Ansatz basiert auf R. Kimball. Laut [Breslin 2004] verwendet dieser einen einzigartigen Datenmodellierungsansatz zur Modellierung der Dimension. Zudem nimmt Kimball statt einer unternehmensweiten Datenbank mehrere operative Datenbanken, genauer gesagt eine Datenbank pro Geschäftsbereich. Auf Basis dieser Datenbanken werden anschließend die Data Marts für die einzelnen Geschäftsbereiche beziehungsweise Abteilungen gebildet und erst aus diesen ein integriertes Data Warehouse [Breslin 2004]. Es handelt sich hier somit um unabhängige Data Marts, da diese erst später zu einem zentralen Data Warehouse zusammengefasst werden. Weitere Information zum Bottom-Up Ansatz findet man in [Kimball & Ross 2004].

Um die zuvor angesprochenen Geschäftsdaten abbilden und diese einfach bewerten und analysieren zu können, werden beim Datenmodell eines Data Warehouses Fakten, Kennzahlen und Attribute verwendet.
Golfarelli definiert ein Fakt als den Mittelpunkt des Interesses in einen Entscheidungsprozess [Golfarelli et al. 1998, S. 218]. Es handelt sich dabei um „numerische Messgrößen, die betriebliche Sachverhalte beschreiben“ [Sattler & Saake 2004, zitiert nach Marx Gómez et al. 2009]. Sie werden von [Marx Gómez et al. 2009] oft auch als Basiskennzahlen bezeichnet, da sich aus ihnen durch die Anwendung arithmetischer Operationen Kennzahlen ableiten.

Kennzahlen sind verdichtete numerische Kenngrößen, die aus den Fakten abgeleitet werden und diese aus verschiedenen Sichten beschreiben. Meist werden Kennzahlen aus Fakten durch Anwendung arithmetischer Operationen errechnet [Marx Gómez et al. 2009]. Beispiele für Kennzahlen wären etwa der Umsatz oder die verkaufte Menge. Laut [Rautenstrauch 2004, zitiert nach Marx Gómez et al. 2009] haben Kennzahlen „informativen Charakter und dienen dazu, durch systematisches Vergleichen Ursachen und Trends ableiten zu können". Fakten und Kennzahlen benötigen beschreibende Attribute die ihnen eine Bedeutung verleihen [Marx Gómez et al. 2009]. Attribute sind also Bezugsgrößen, nach denen Fakten und Kennzahlen gruppiert werden können.

Ein besonderes Augenmerk gilt dem Datenmodell selbst, da in einem Data Warehouses im Laufe der Zeit riesige Mengen an Daten anfallen können und die Daten deshalb mittels spezieller Technologien, wie multidimensionale Datenbanken und Datencubes, zusammengefasst werden [Koch 2001]. Das Datenmodell eines Data Warehouse ist deshalb multidimensional und wird in der Regel als ein Würfel (*Cube*) dargestellt. Dabei sind die Kanten beziehungsweise Achsen des Würfels die Dimensionen und die einzelnen Elemente/Datensätze des Würfels die Kennzahlen.

„Dimensionen sind diskrete Attribute, die die minimale Granularität, mit der die Fakten dargestellt werden, beschreiben" [Golfarelli et al. 1998, S. 218]. Typische Dimensionen wären Zeit, Ort und Produkt [Golfarelli et al. 1998, S. 218]. Dimensionen beschreiben eine spezielle Sicht auf Fakten und Kennzahlen. Sie geben ihnen somit eine Bedeutung. Andernfalls wären Fakten und Kennzahlen lediglich Zahlen. Beispielsweise legen sie fest, dass die Kennzahl Umsatz pro Produkt und Filiale im ersten Quartal 2010 dargestellt wird. Des Weiteren lassen sich Dimensionen im Allgemeinen hierarchisch gliedern. Zum Beispiel können Produkte zu Produktgruppen, Filialen zu Verkaufsbezirken oder gar Ländern und die einzelnen Quartale zum Verkaufsjahr 2010 gruppiert werden. Dimensionen die sich nicht hierarchisch gliedern lassen werden als nicht-hierarchischer Dimensionstyp oder degenerierte Dimension bezeichnet [Bauer & Günzel 2009]. Ein Beispiel dafür wäre eine Dimension mit IST-, PLAN- und SOLL-Werten [Marx Gómez et al. 2009]. Jede Dimension hat Attribute, welche die Merkmale der Kennzahlen sind. Dabei wird laut [Golfarelli et al. 1998] in dimensionale und nicht-dimensionale Attribute unterschieden. Dimensionale Attribute sind zum Beispiel das Land, die Stadt oder die Filiale in einer Dimension Ort. Die Einwohneranzahl und die Fläche eines Ortes wären hingegen nicht-dimensionale Attribute. Sie stellen nur eine inhaltliche Verfeinerung des dimensionalen Attributs Stadt dar und können nicht zur Aggregation verwendet werden.

Die Anzahl der Dimensionen eines Datenwürfels wird in der Regel als Dimensionalität bezeichnet. Sie kann durchaus größer als drei werden, jedoch kann ein Würfel mit mehr als drei Dimensionen nicht mehr grafisch dargestellt werden. Der Würfel selbst wird auch als Cube oder Datencube bezeichnet.

Das folgende Beispiel von [Marx Gómez et al. 2009] zeigt einen Cube mit den Dimensionen Studienfach, Studienabschnitt und Semester. Die einzelnen kleinen Würfel an den Schnittpunkten der Dimensionen stellen dabei die Fakten bzw. Kennzahlen dar. Eine Kennzahl könnte beispielsweise die Anzahl der Studenten pro Studienfach, Studienabschnitt und Semester sein.

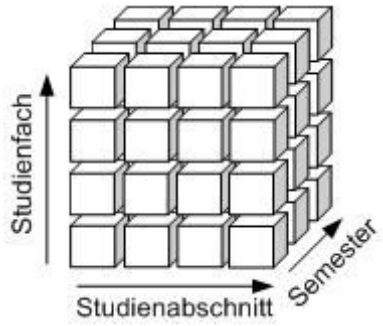

Abbildung 2: Multidimensionales Datenmodell [Marx Gómez et al. 2009]

2.2. Objekt-relationale Datenbanken

Eine Datenbank bzw. ein Datenbanksystem besteht aus einem Datenbankverwaltungssystem und der Datenbasis. Die Datenbasis sind die eigentlichen Daten, das Datenbankverwaltungssystem ist „die Gesamtheit der Programme zum Zugriff auf die Datenbasis, zur Kontrolle der Konsistenz und zur Modifikation der Daten“ [Kemper & Eickler 2006, S. 17]. Eine Datenbank kann auch als „eine strukturierte Sammlung von Daten, die einen speziellen Ausschnitt der realen Welt vereinfacht und schematisiert repräsentiert“ [Lange 2006, S. 289] definiert werden.

Es gibt verschiedene Datenmodelle, die einer Datenbank zugrunde liegen können. Zwei der gängigsten Datenmodelle sind das relationale und das objektorientierte Datenmodell. Ein Datenmodell beschreibt die logische Datenstruktur eines Datenbanksystems [Lange 2006]. Das gängigste Datenmodell ist das relationale Datenmodell, das auf dem Konzept von [Codd 1990] basiert. In diesem logischen Datenmodell werden die Daten als mathematische Relationen betrachtet, welche als Tabellen veranschaulicht werden. Eine Spalte entspricht

dabei einem Attribut, eine Zeile einem Tupel bzw. Datensatz. Jedes Attribut hat einen bestimmten elementaren Datentyp wie *Number* oder *Varchar*. Nähere Information zu relationalen Datenbanken findet man in [Codd 1990].
Bei objektorientierten Datenmodellen wird der Datenbestand in Form von Objekten verwaltet. Es werden also Objekte anstatt von Tabellen gespeichert. Dabei setzt sich jedes Objekt aus Attributen und Methoden zur Manipulation des Objekts zusammen. Objekte sind daher komplexer als Relationen. Die Daten in den Objekten sind in der Regel gekapselt, d.h. sie können von außen nur über zur Verfügung gestellte Methoden gelesen werden. Außerdem können Objekte hierarchisch gegliedert werden, sie können somit auch Subtypen haben. Jedes Objekt hat zudem eine eindeutige Objektidentität (OID), über die es referenziert werden kann [Lufter 1999]. In objektorientierten Datenmodellen gelten also generell „die Prinzipien der Objektorientierung wie Bildung komplexer Objekte mit eigener Objektidentität, Typen und Klassen, Vererbung, Kapselung, Polymorphismus und spätes Binden“ [Lange 2006, S. 313]. Der Hauptunterschied zum relationalen Datenmodell besteht darin, dass in relationalen Datenbanken sämtliche Programmoperationen von außen aufgerufen werden, während sie in objektorientierten Datenbanken direkt beim Objekt gespeichert sind [Stahlknecht & Hasenkamp 2006].

Das objekt-relationale Datenmodell ist eine Mischform aus dem relationalen und dem objektorientierten Datenmodell. Es handelt sich dabei um relationale Datenbanken, die im Hinblick auf Prinzipien der Objektorientierung erweitert wurden. Dabei wird es möglich, einzelnen Spalten bzw. Tabellen innerhalb der Datenbank eine Entität aus der realen Welt als Datentyp zu geben, also einen benutzerdefinierten Datentyp mit Attributen, Methoden und Konstruktoren zur Initialisierung der Objekte [Oracle 2010]. Wie bei den objektorientierten Systemen wird dabei jedes Objekt durch den Object Identifier (OID) eindeutig bestimmt. Es kann jedoch auch ein eigener Primary Key, wie zum Beispiel eine Sozialversicherungsnummer beim Objekt Person, als OID genommen werden.
Weiters ist es in objekt-relationalen Datenmodellen möglich die Prinzipien der Generalisierung/Vererbung und von verschachtelten Relationen zu nutzen [Lange 2006]. Methoden und Attribute werden dabei gemäß den Prinzipien der objektorientierten Programmierung vererbt.
In objekt-relationalen Oracle Datenbanken gibt es neben den relationalen Tabellen, die nun neben elementaren Datentypen auch Objekte als Attribute beinhalten können, auch sogenannte Objekttabellen (*„Object Tables“*). In Objekttabellen entspricht jeder Tupel einem

Objekt. Zudem bietet Oracle Erweiterungen von SQL und PL/SQL zur Manipulation von Objekten. Zum Beispiel gibt es nun mit dem VALUE-Operator die Möglichkeit, Objekte als Ganzes oder mit dem REF-Operator die Referenz auf das Objekt aus der Datenbank auszulesen.
Oracle Collections wie Varrays und Nested Tables können verwendet werden um m-n-Beziehungen beziehungsweise verschachtelte Relationen abzubilden. Um eine verschachtelte Tabelle, die als Collection-Typ gespeichert ist, als Relation darzustellen, wird der TABLE-Operator zur Verfügung gestellt [Oracle 2010].
Detaillierte Information zu den zur Verfügung stehenden objekt-relationalen Konzepten und Operatoren von Oracle findet man in [Oracle 2010].

2.3. Integration von Daten

Daten-Integration ist derzeit weltweit ein sehr wichtiges Themenfeld und die Probleme, die beim Überbrücken der Schema-Heterogenitäten und beim Transformieren der Daten entstehen, sind allgemein bekannt. Generell wird bei der Integration von Datenquellen in Schema-Integration und Daten-Integration unterschieden [Koch 2001]. Schema-Integration ist ein softwaretechnischer bzw. wissenstechnischer Ansatz, bei dem Informationssysteme umgestaltet werden, um ein einziges globales Schema zu erhalten [Koch 2001]. Daten-Integration beschäftigt sich mit der Kombination von Daten aus mehreren verschiedenen Quellen und der Behandlung von Heterogenitäten auf Datenebene. Daten-Integration kann somit folgendermaßen definiert werden: “Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data” [Lenzerini 2002, S. 233].
Bei der Integration von Daten geht es also darum die Daten von verschiedenen Quellen, wie etwa Data Marts, in ein globales Schema, zum Beispiel das globale Data Warehouse, zu integrieren und dem Benutzer eine einheitliche Sicht auf die Daten zur Verfügung zu stellen. Die Daten sind dabei weiterhin bei den Quellen gespeichert, das globale Schema bietet lediglich eine virtuelle, einheitliche Sicht auf die Daten [Lenzerini 2002]. Weiters ist es bei der Integration von Daten auch wichtig, die lokalen Schemata zu verstehen und wie diese die Daten strukturieren [Miller et al. 2001].
Die virtuelle, einheitliche Sicht, die dem Benutzer zur Verfügung gestellt wird, wird als *mediated schema* oder *global schema* bezeichnet [Levy 2000]. In dieser Arbeit wird diese virtuelle, einheitliche Sicht in der Folge das globale Schema genannt. Wenn nun der Benutzer

in weiterer Folge die Daten abfragen will, richtet er seine Abfrage an das globale Schema und nicht an die einzelnen Datenquellen. Es ist dann die Aufgabe des Daten-Integrationssystems, die Abfrage umzuformulieren, an die einzelnen Datenquellen zu schicken und die Ergebnisse wieder zusammenzufassen [Calì et al. 2004]. Der Benutzer kann sich beim Formulieren der Abfrage also darauf konzentrieren, *was* er wissen will, und muss sich nicht darum kümmern, *wie* er die gewünschten Daten von den einzelnen Quellen zusammen bekommt [Levy 2000].
Ziel für den Anwendungsdesigner ist es also, ein globales Schema zu entwerfen, das alle erforderlichen Daten in einer integrierten und einheitlichen Sicht beschreibt. Zu beachten ist jedoch, dass alle Relationen des globalen Schemas lediglich virtuell sind, d.h. die Daten werden nicht physisch integriert, sie verbleiben einzig und allein bei den einzelnen Quellen. Weiters sind Beschreibungen für die Beziehung zwischen dem globalen und dem lokalen Schema zu erstellen. Diese Beschreibungen spezifizieren den Inhalt der Daten, die einzelnen Attribute, die Einschränkungen der Daten (beispielsweise wenn eine Datenquelle lediglich Verkäufe der Jahre 2005 bis 2008 enthält) und die Vollständigkeit und die Zuverlässigkeit der Daten des lokalen Schemas. [Levy 2000]

Bei der Integration von Daten wird in die strukturelle und die semantische Integration unterschieden [Koch 2001]. Strukturelle Integration, oftmals auch unter der Bezeichnung „wrapping“ zu finden, beschäftigt sich mit strukturellen Heterogenitäten, wie unterschiedlichen Datenmodellen, Abfrage- und Datenzugriffssprachen und Protokollen. Dies ist vor allem bei Altsystemen nötig. [Koch 2001]
Die semantische Integration beschäftigt sich mit semantischen Unterschieden zwischen dem globalen und dem lokalen Schema. Semantische Unterschiede können auf Ebene von Entitäten (Relationen der Datenbank) oder auf Ebene der Daten selbst auftreten [Koch 2001].

Eine der wesentlichen Eigenschaften bei der Integration von Daten ist, dass die einzelnen Quellen autonome Systeme sind, die bereits vorher unabhängig vom globalen Schema existiert haben. Dies bringt für die Integration der Daten einige Schwierigkeiten mit sich [Levy 2000]:

- Die Daten können ähnlich, überlappend und gegebenfalls sogar widersprüchlich sein.
- Unterschiedliche Datenmodellierung:
  Da die Datenquellen für verschiedene Zwecke in mehreren unterschiedlichen Organisationen entworfen wurden, können sie semantisch unterschiedlich modelliert sein. Zum Beispiel kann in einem Schema für eine Person eine Telefonnummer und

eine Zweitnummer als Attribut gespeichert werden, in einem anderen können die Telefonnummern jedoch auch in eine eigene Tabelle ausgelagert werden. Des Weiteren können Attribute und Tabellen unterschiedlich benannt werden.

- Unterschiedliche Namenskonventionen für die Daten:
  Zwei autonome Datenquellen können verschiedene Wege nutzen, um auf äquivalente Objekte zuzugreifen. Zum Beispiel kann in einem Schema der Name von Personen als Ganzes, im Anderen hingegen separat als Vorname und Nachname gespeichert werden. Außerdem kann es auch Unterschiede bei den Datenformaten geben, beispielsweise für das Geburtsdatum von Personen.

Wie zuvor beschrieben, wird eine Abfrage an das globale Schema gestellt und muss in der Folge umgeformt werden, um die Daten von den einzelnen Quellen zu bekommen. Es sind jedoch auch weitere Schritte, wie eine Optimierung der Abfrage und der Übersetzung für ein einzelnes Quellsystem, erforderlich. Abbildung 3 zeigt die Architektur eines Daten-Integrationssystems nach [Levy 2000] mit allen Schritten, die bei der Abarbeitung einer Abfrage erforderlich sind.

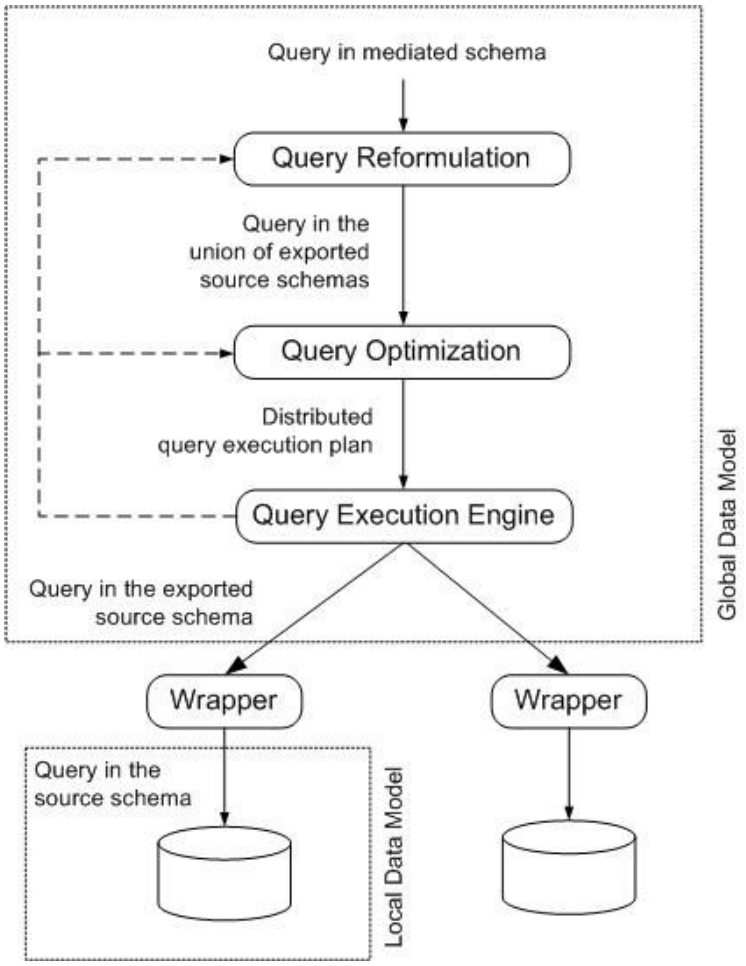

Abbildung 3: Architektur eines Daten-Integrationssystems [Levy 2000]

- Query Reformulation:
  Wie bereits zuvor beschrieben, ist eine Abfrage an das globale Schema gerichtet. Sie muss also im ersten Schritt in mehrere Abfragen an die einzelnen Datenquellen umformuliert werden. Diese Abfragen müssen direkt an das Quellschema gerichtet sein. Zudem müssen sie korrekt („sound"), vollständig und effizient sein [Levy 2000].
- Query Optimization:
  Im zweiten Schritt werden die Abfragen optimiert. Dabei wird ein Query Execution Plan erstellt, also ein Plan, in dem genau festgelegt ist, wie eine Abfrage ausgeführt werden soll und in welcher Reihenfolge die einzelnen Abfragen abgesetzt werden. Beim Festlegen, wie eine Abfrage ausgeführt werden soll, wird die Abfrage nochmals umformuliert. Es wird dabei genau festgelegt, in welcher Reihenfolge die einzelnen Operatoren (Join, Selektion, Projektion) ausgeführt werden sollen.
- Query Execution Engine:
  Die Query Execution Engine erhält den Query Execution Plan als Input, evaluiert die einzelnen Abfragen und leitet sie an die Quellen weiter.
- Wrapper:
  Der Wrapper befindet sich direkt bei der Datenquelle. Er ist dafür verantwortlich, dass die Abfrage in ein für die Quelle lesbares Format übersetzt wird. Außerdem muss er die Ergebnisdaten wiederum in ein für den Query Prozessor lesbares Format bringen.

Da eine Abfrage an das globale Schema und nicht direkt an einzelne Quellen geschickt wird, ist es ein Kernaspekt der Daten-Integration, die Beziehung zwischen dem globalen Schema und den Datenquellen zu modellieren, damit das Daten-Integrationssystem die Abfrage umformen und sie zur Abarbeitung an die einzelnen Quellen weiterleiten kann. Das Daten-Integrationssystem muss also genau wissen, wie die Daten des globalen Schemas aus den lokalen Schemata zusammengesetzt sind, das heißt wo welche Daten gespeichert sind.
Die Beziehung zwischen dem globalen Schema und den Datenquellen wird mittels Mappings beschrieben. [Miller et al. 2001] definieren ein Mapping als „ein Programm oder ein Satz von Queries, die zur Übersetzung von Daten zwischen unterschiedlichen Schemata verwendet werden können" [Miller et al. 2001, S. 79]. Solche Mappings müssen meist vom Benutzer definiert werden. Zum Beispiel wird das globale Schema eines Data Warehouses von den Administratoren (meist in monatelanger Arbeit) unabhängig von den einzelnen Datenquellen für einen bestimmten Zweck beziehungsweise einer Reihe wichtiger Abfragen auf das Data Warehouse entworfen. Um die Daten anschließend ins Data Warehouse laden zu können,

müssen noch manuell Mappings zu den Datenquellen erstellt werden. Es wird also zuerst entschieden, welche Daten in welcher Form erforderlich sind (Entwurf des globalen Schemas) und erst danach wie diese aus den Datenquellen geholt werden können. [Miller et al. 2001].

Global-As-View-Ansatz

Beim Global-As-View-Ansatz, kurz GAV-Ansatz, wird das globale Schema auf Basis der Datenquellen ausgedrückt [Lenzerini 2002]. Hier wird für jede Relation im globalen Schema eine Query spezifiziert, wie die Daten aus den einzelnen Quellen geholt werden können.
In der Folge wird der Global-As-View-Ansatz anhand eines Beispiels mit zwei Tabellen genauer erklärt. Dabei enthält die erste Tabelle verschiedene Produkte, ihren Preis und ihren Hersteller und die zweite die Produktnummer und die Bewertung der Produkte:
$Tabelle_1$(id, produkt, preis, hersteller)
$Tabelle_2$(id, bewertung)

Besteht nun zum Beispiel das globale Schema aus einer Relation mit allen Produkten inklusive dem Produkthersteller und einer Relation, die eine Bewertung zu den Produkten enthält, dann sieht das Schema der beiden Relationen folgendermaßen aus:

```
Create View Produkthersteller AS
    Select db1.id, db1.produkt, db1.hersteller From db1;

Create View Produktbewertung AS
    Select db1.id, db1.produkt, db2.bewertung
    From db1, db2
    Where db1.id = db2.id;
```

Query Umformulierung ist in GAV relativ einfach, da die Relationen des globalen Schemas auf Basis der Relationen der Datenquellen definiert sind. Um eine Abfrage umzuformulieren, muss diese Definition einfach aufgefaltet werden.

Wenn nun beispielsweise alle Produkte des Herstellers A inklusive ihrer Bewertung abgefragt werden, würde die Abfrage wie folgt umformuliert werden:

```
Select produkt, bewertung
From   Produktbewertung
Where  produkt in (Select produkt
                   From Produkthersteller
                   Where hersteller = 'Hersteller A');
```

```
Select db1.produkt, db2.bewertung
From   db1, db2
Where  db1.id = db2.id
And    db1.hersteller = 'Hersteller A';
```

Der Hauptvorteil des GAV-Ansatzes ist, dass die genaue Definition, woher das System seine Daten bekommt, eine sehr einfache und zudem effiziente Umformulierung einer Query ermöglicht.
Der Ansatz ist besonders geeignet für Systeme mit einer gleich bleibenden Menge von Datenquellen, jedoch hat er den Nachteil, dass das Hinzufügen von neuen Datenquellen sehr komplex ist [Lenzerini 2002]. Für jede neue Quelle muss genau definiert werden, welche Daten des globalen Schemas daraus genommen werden. Hinzu kommt, dass mit allen bestehenden Quellen abgestimmt werden muss, welche Daten nun aus der neuen Quelle genommen werden. Der Global-As-View-Ansatz eignet sich also nicht besonders für Daten-Integrationssysteme mit besonders vielen Datenquellen [Levy 2000].
Ein weiterer Nachteil des Ansatzes ist, dass nicht jede Abfrage in eine äquivalente Query für die Quellen aufgelöst werden kann. Oftmals muss man sich mit einer „maximally-contained“ Umformulierung, also einer best möglichen Annäherung an die eigentliche Lösung, begnügen [Levy 2000]. Dies lässt sich dadurch erklären, dass nicht immer vollständige Information in den Quellen enthalten sein muss. Dies ist beispielsweise der Fall, wenn die Quelle im oben angeführten Beispiel bloß Produktbewertungen seit 1980 enthält, jedoch alle Autobewertungen seit 1960 abgefragt werden. Als Lösung würde man dann also nur einen Teil der tatsächlichen Antwort bekommen.
Laut [Lenzerini 2002] lässt sich die Genauigkeit, mit der die Sicht einer Quelle mit den eigentlichen Quelldaten übereinstimmt, in drei Kategorien unterteilen:

- Sound Views (= korrekte Sichten):
  Wenn eine Sicht *korrekt* ist, dann ist jedes Tupel, das in der eigentlichen Datenbank enthalten ist, auch in der mit der Quelle verbundenen Sicht enthalten.
- Complete Views (= komplette Sichten):
  Ist eine Sicht *komplett*, so kann nicht darauf geschlossen werden, dass ein Tupel der eigentlichen Datenbank auch in der damit verbundenen Sicht enthalten ist. Jene Tupel, die nicht in der Datenbank enthalten sind, sind jedoch auch nicht in der Sicht.

- Exact Views (= exakte Sichten):
  Wenn eine Quelle *exakt* ist, enthält sie genau jene Menge an Tupeln, die auch in der damit verbundenen Sicht sind.

Beim Global-As-View-Ansatz wird in den meisten Ansätzen implizit angenommen, dass die Sichten exakt sind. Dies ist allerdings nur dann der Fall, wenn im globalen Schema keine Einschränkungen definiert wurden. [Lenzerini 2002]

3. Data Warehouses mit hetero-homogenen Hierarchien und Cubes

In diesem Abschnitt wird das Konzept von [Neumayr et al. 2010] zur Modellierung von hetero-homogenen Hierarchien und Cubes vorgestellt. Dazu wird zu Beginn aufgezeigt, wozu hetero-homogene Hierarchien benötigt werden. Anschließend wird in Kapitel 3.2 gezeigt, wie [Neumayr et al. 2010] das Konzept der Multilevel Objects (M-Objects) adaptieren. Anschließend werden noch Multilevel Relationships (M-Relationships) und das Konzept des Multilevel Cubes (M-Cubes) vorgestellt. Die gesamte Beschreibung der hetero-homogenen Hierarchien und Cubes in diesem Kapitel basiert auf [Neumayr et al. 2010].
Zuletzt wird in Kapitel 3.4 der in der Arbeit als Ausgangsbasis verwendete Prototyp des Data Warehouse Management Systems von [Schütz 2011] beschrieben, welcher das Konzept von [Neumayr et al. 2010] in einer Oracle 11g Datenbank umsetzt.

3.1. Motivation

Bei der Integration von Data Marts in ein globales Data Warehouse treten oft Heterogenitäten zwischen dem globalen und dem lokalen Schema auf. Um dieses Problem lösen zu können, haben [Neumayr et al. 2010] das Konzept der hetero-homogenen Hierarchien eingeführt. Bei hetero-homogenen Hierarchien handelt es sich um Dimensionshierarchien die ein minimales, gemeinsames Schema teilen, wo jedoch die Sub-Hierarchien heterogen sein können. Eine Sub-Hierarchie ist eine Hierarchie, deren Wurzel ein Kindknoten des Wurzelknotens der ursprünglichen Hierarchie ist.

Innerhalb von Sub-Hierarchien und Sub-Cubes können die folgenden Heterogenitäten auftreten [Neumayr et al. 2010]:

- Zusätzliche nicht-dimensionale Attribute:
  Eine Sub-Hierarchie kann zusätzliche nicht-dimensionale Attribute einführen. Beispielsweise kann ein Auto in einer Dimension *Produkt* ein zusätzliches Attribut *Maximalgeschwindigkeit* einführen. Im allgemeinen Schema ist dies jedoch noch nicht enthalten, da eine Maximalgeschwindigkeit für andere Produkte, wie Bücher oder Musik, keine Relevanz hat.

- Zusätzliche Levels:
  Sub-Hierarchien können auch zusätzliche Levels einführen. Ein Beispiel dafür wäre ein zusätzlicher Level *Marke* bei Produkten der Kategorie *Auto*.
- Zusätzliche Kennzahlen:
  Sub-Cubes können auch zusätzliche Kennzahlen einführen. Beispielsweise kann ein Sub-Cube für Verkäufe in den USA zusätzlich zum Umsatz auch noch eine Kennzahl verkaufte Menge enthalten.
- Unterschiedliche Einheiten:
  Die Einheit der Kennzahlen von Sub-Cubes kann unterschiedlich sein. So wird der Umsatz von Verkäufen in Österreich in Euro angegeben, in den USA jedoch in Dollar.
- Unterschiedliche Granularität der Kennzahlen:
  Die einzelnen Kennzahlen können unterschiedliche Granularitäten aufweisen. Beispielsweise kann der Umsatz monatsweise pro Produkt und Filiale ausgewiesen werden und die verkaufte Menge nur jährlich, pro Produkt und Verkaufsbezirk.
  Weiters kann eine Kennzahl in verschiedenen Sub-Cubes eine unterschiedliche Granularität aufweisen, wenn etwa der Umsatz in Österreich pro Monat, Produkt und Stadt gemessen wird, in der Schweiz jedoch pro Monat, Produkt und Filiale. Zu beachten ist jedoch, dass die Granularität in einem Sub-Cube nur auf eine feinere Granularität geändert werden darf, nicht auf eine gröbere. Dadurch ist gewährleistet, dass es eine Mindestgranularität für einen Sub-Cube gibt.

Dabei ist anzumerken, dass sich die ersten beiden Punkte, die zusätzlichen nicht-dimensionalen Attribute und die zusätzlichen Levels, auf die Dimensionen des Data Warehouses, und die letzten drei auf Heterogenitäten im Cube beziehen.

3.2. Hetero-homogene Dimensionen

Um Heterogenitäten innerhalb von Sub-Hierarchien darstellen zu können, verwenden [Neumayr et al. 2010] hetero-homogene Hierarchien. Diese werden mit M-Objects, genauer gesagt Konkretisierungshierarchien von M-Objects, dargestellt. So hat jedes M-Object eine Level-Hierarchie, von abstrakt zu konkret, und beschreibt sich selbst und die gemeinsamen Eigenschaften der Sub-Hierarchie. Ein M-Object weist allen Attributen seines Top-Levels konkrete Werte zu. Der Top-Level beschreibt das M-Object selbst, alle weiteren Levels die

gemeinsamen Eigenschaften aller darunter liegenden M-Objects. Jedes M-Object ist somit zugleich Instanz des jeweiligen Dimensionslevels und Schema für die Sub-Hierarchie. Des Weiteren besteht jedes M-Object aus Levels und Attributen. Jedes Attribut ist einem bestimmten Level zugeordnet und hat einen konkreten Datentyp, und einen optionalen Wert. Beispielsweise hat das M-Object *Product* (vgl. Abbildung 4) die drei Levels Top, Kategorie und Modell, wobei Top der Top-Level des Objekts ist. Weiters besitzt das M-Object die Attribute Verantwortlicher und Steuersatz auf dem Level Kategorie und Kosten auf dem Level Modell. Den einzelnen Attributen ist ein Datentyp zugewiesen, so ist zum Beispiel das Attribut *Kosten* eine Gleitkommazahl.

Ein M-Object kann ein weiteres M-Object konkretisieren, wobei das konkretisierende Objekt Werte für seine Top-Level Attribute festlegen kann und neue Levels und Attribute einführen kann. Abbildung 4 zeigt das M-Object *Auto* als Konkretisierung des M-Objects *Produkt*. Es definiert Werte für den Verantwortlichen und den Steuersatz und führt zusätzlich ein Level für die Automarke und ein Attribut für die maximale Geschwindigkeit ein. Außerdem wird dem neuen M-Object das M-Object Produkt als Elternobjekt zugeordnet. Ein Kindobjekt befindet sich immer am Second-Top-Level des Elternobjekts (hier *category*). Zudem erbt es alle Levels unter dem Top-Level des Elternobjekts mit derselben hierarchischen Ordnung und alle Attribute, die sich auf den geerbten Levels befinden. Ein Attribut kann nicht mehrmals auf unterschiedlichen Levels definiert werden. Zudem kann es auf einem Level nicht verschiedene Datentypen haben. Hat ein Kindobjekt mehrere Elternobjekte, so befindet es sich am gemeinsamen Second-Top-Level aller Elternobjekte.

Wie bereits zuvor beschrieben, wird eine hetero-homogene Dimensionshierarchie mit M-Objects repräsentiert. Dabei gelten für eine homogene Dimension folgende Eigenschaften [Neumayr et al. 2010]:

- Jede Dimension besteht aus einer Hierarchie von M-Objects.
- Jeder Dimensionslevel entspricht einem Level des Wurzel M-Objects.
- Jeder Level wird durch dessen Attribute beschrieben. Diese Attribute entsprechen den nicht-dimensionalen Attributen von [Golfarelli et al. 1998].
- Eine Level-Ausprägung wird durch ein M-Object repräsentiert, welches als Top-Level den jeweiligen Dimensionslevel hat.
- Jede Level-Ausprägung wird durch die Top-Level Attribute der M-Objects, die sich auf dem jeweiligen Level befinden, beschrieben.

Abbildung 4 zeigt eine Produkthierarchie als Beispiel für eine hetero-homogene Dimension. Jedes Kindobjekt (Child-M-Object) in einer Dimensionshierarchie kann zusätzliche Levels und nicht-dimensionale Attribute einführen (vgl. M-Object *Car*). Führt ein M-Object zusätzliche Levels oder Attribute ein, so ist es zugleich Instanz des jeweiligen Dimensionslevels, und Schema für die Sub-Hierarchie.

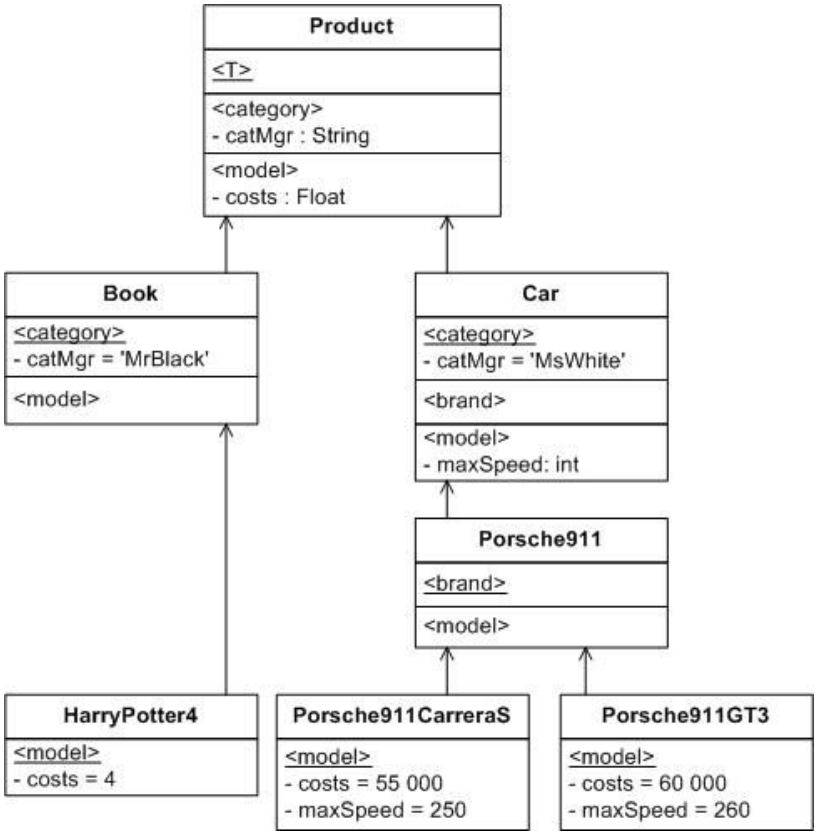

Abbildung 4: Hetero-homogene Produktdimension [Neumayr et al. 2010]

Die Dimensionen, in denen die Kennzahlen des Cubes organisiert sind, bestehen aus einer Menge von M-Objects. Jedes M-Object, außer dem Wurzelobjekt, hat ein oder mehrere Parent-M-Objects. Bei den Kindobjekten eines M-Objects wird in direkte und transitive Konkretisierungen unterteilt [Neumayr et al. 2010]. So ist etwa das M-Object *Auto* eine direkte Konkretisierung des Produkts, die Marke *Porsche911* hingegen nur eine transitive. Da M-Objects mehrere Parent-M-Objects haben können, könnten Probleme durch Mehrfachvererbung auftreten. Solche Probleme verhindert die *Unique Induction* Regel. Diese besagt, dass Level und Attribute nur einmal von einem Objekt eingeführt werden können [Neumayr et al. 2010]. Wenn es zum Beispiel eine zusätzliche Produktkategorie *Musik* gäbe, könnte diese also nicht ein Level *brand* einführen, da dieser Level bereits vom M-Object

Auto eingeführt wurde. Ein Objekt *Musik* müsste also ein Level *music_label* oder dergleichen einführen.

3.3. Hetero-homogene Cubes

Hetero-homogene Cubes werden mit M-Relationships modelliert. M-Relationships beschreiben Beziehungen von M-Objects auf verschiedenen Abstraktionsebenen. Laut [Neumayr et al. 2010] haben M-Relationships folgende Eigenschaften:

- M-Relationships auf verschiedenen Abstraktionsebenen werden analog zu M-Objects in Konkretisierungshierarchien angeordnet. Diese hierarchische Einteilung wird aus der Hierarchie der M-Objects abgeleitet.
- Eine M-Relationship repräsentiert mehrere Abstraktionsebenen. Sie besteht aus einer konkreten Beziehung und mehreren Beziehungsklassen. Eine Beziehungsklasse enthält alle Nachfolge-M-Relationships, die Objekte der entsprechenden nachfolgenden Levels verbinden.
- Eine M-Relationship impliziert extensionale Einschränkungen für ihre Konkretisierungen auf verschiedenen Levels.
- M-Relationships können hetero-homogene Cubes bzw. Sub-Cubes abbilden.
- M-Relationships können für Abfragen verwendet und navigiert werden.

Jede M-Relationship besteht aus einer Reihe von M-Objects, die zusammen die Koordinate der M-Relationship bilden. Dabei ist genau ein M-Object aus jeder Dimension Bestandteil der Koordinate. Da Cubes beliebig viele Dimensionen haben können, sind M-Relationships mehrstellig. Des Weiteren werden sie von Attributen, den Kennzahlen, beschrieben. Jede Kennzahl hat zugehörige Aggregationsfunktionen (SUM, MAX, MIN etc.), eine Einheit (Euro, Dollar oder dergleichen) und einen Connection-Level auf dem der Wert der Kennzahl definiert wird. Ein Connection-Level legt die Granularität der Kennzahlen fest. Er setzt sich aus genau einem Level aus jeder Dimension zusammen.

Abbildung 5 zeigt ein Beispiel eines homogenen Data Warehouses, das mit M-Objects und M-Relationships modelliert wurde. Das Data Warehouse hat drei Dimensionen (Produkt, Zeit und Ort) und eine Kennzahl Umsatz. Weiters definiert es eine drei-wertige M-Relationship *sales* zwischen den M-Objects, Produkt, Zeit und Ort, für die die Kennzahl Umsatz auf den

Connection Level <*model,month,city*> mit der Einheit Euro und der Aggregationsfunktion SUM definiert wurde. Die Aggregationsfunktion SUM gibt an, dass die Kennzahl aufsummiert werden kann.

M-Relationships bilden wie M-Objects eine Konkretisierungshierarchie. Der Nachfolger einer M-Relationship in der Konkretisierungshierarchie wird als eine Konkretisierung bezeichnet, genau dann wenn ein oder mehrere M-Objects der Koordinate des Nachfolgers durch konkretisierte M-Objects ersetzt werden. Eine Konkretisierung beziehungsweise ein Nachfolger kann zusätzliche Kennzahlen einführen, Kennzahlen auf ein konkreteres Level setzen und Werte für die Kennzahlen auf dem Top-Connection-Level zur Verfügung stellen. Der Top-Connection-Level einer M-Relationship setzt sich aus den Top-Levels der mit ihr verbundenen M-Objects zusammen.

Beispielsweise ist die M-Relationship *sales* zwischen den Objekten HarryPotter4, February 2009 und Salzburg mit dem Top-Connection-Level <model,month,city> ein Nachfolger der M-Relationship *Product*, *Time* und *Location*, da in diesem Fall alle Objekte der einzelnen Koordinatenbestandteile konkreter sind als jene der übergeordneten M-Relationship.

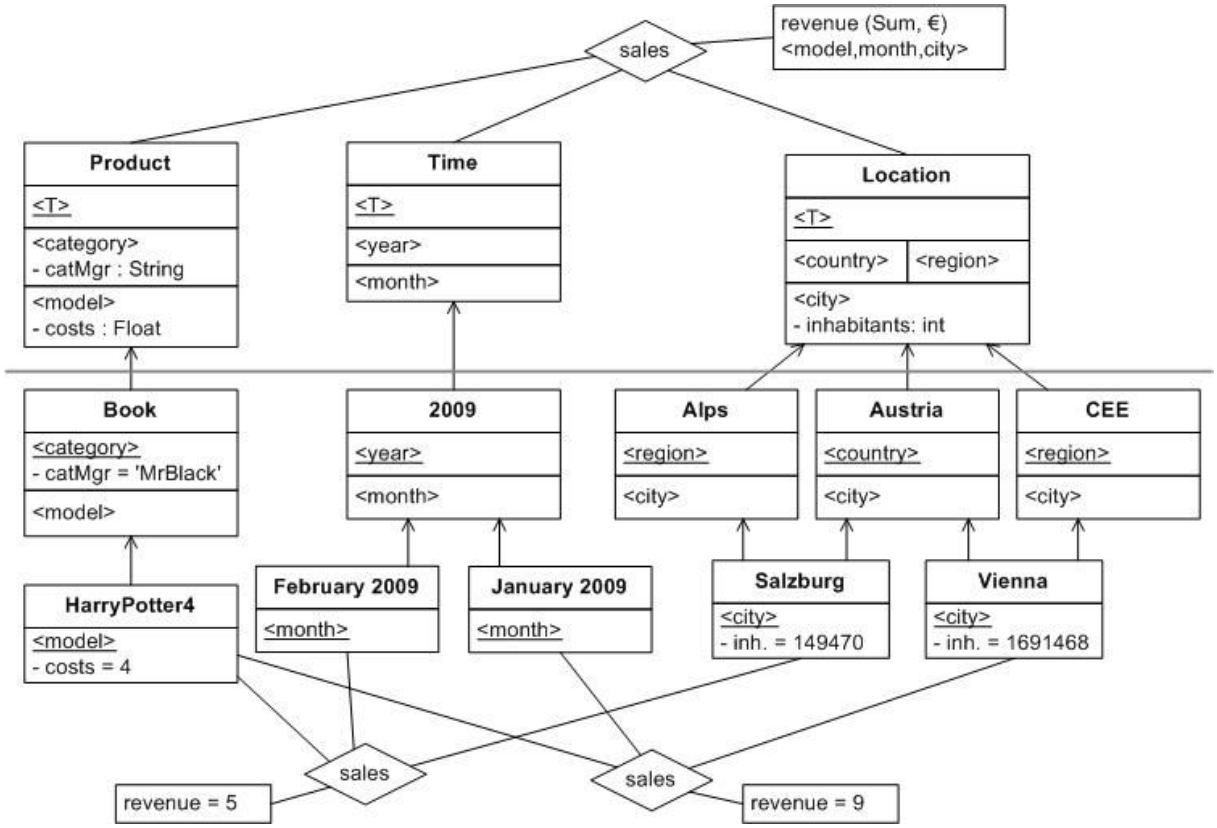

Abbildung 5: Data Warehouse mit M-Relationships [Neumayr et al. 2010]

[Neumayr et al. 2010] verwenden statt eines Cubes einen Multilevel Cube, kurz M-Cube. Ein M-Cube ist eine Menge von M-Relationships, auf denen Konsistenzregeln und Query-

Operationen definiert sind. Ein M-Cube mit n Dimensionen besteht aus einer Konkretisierungshierarchie n-stelliger M-Relationships. Jede M-Relationship repräsentiert eine Zelle des Cubes auf einer bestimmten Abstraktionsebene und stellt auch einen eigenen Sub-Cube dar. Die Wurzel M-Relationship des M-Cubes besteht aus den Wurzel Objekten der n Dimensionen und hat ein oder mehrere Kennzahlen, die sich auf einem Connection Level auf den unteren, detaillierteren Levels der n Dimensionen befindet. Die einzelnen Zellen beziehungsweise Kennzahlen des Cubes werden durch Konkretisierungen der Wurzel-M-Relationship repräsentiert, welche n M-Objects auf dem Connection Level, für den die Kennzahl definiert wurde, miteinander verbindet.

Abbildung 6 zeigt anhand eines konkreten Cube-Ausschnitts, wie Heterogenitäten in einem M-Cube abgebildet werden können:

- Wie bereits beschrieben, können Sub-Cubes zusätzliche Kennzahlen definieren, wie in diesem Beispiel der Sub-Cube *Car*, 2009 und Switzerland die Kennzahl *verkaufte Menge* hinzufügt.
- Sub-Cubes können unterschiedliche Einheiten aufweisen. So wird die Kennzahl *cheapestOffer* generell zwar in Euro angegeben, im Teil-Cube der Schweiz jedoch in Schweizer Franken.
- Verschiedene Kennzahlen können eine unterschiedliche Granularität haben. Zum Beispiel ist das billigste Angebot auf dem Connection-Level <category,year,country> definiert, der Umsatz jedoch auf <model,month,city>.
- Eine Kennzahl kann in Sub-Cubes eine andere Granularität haben. Beispielsweise wird im Beispiel aus Abbildung 6 der Umsatz generell pro Stadt ausgewiesen, im Sub-Cube der Schweiz jedoch auf Filialebene.

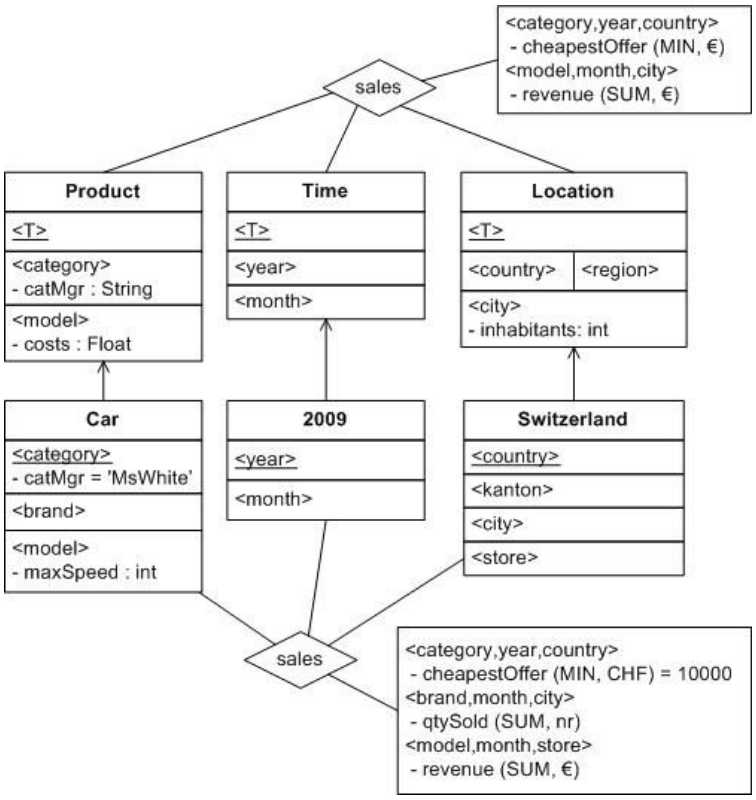

Abbildung 6: Konkretisierung von M-Relationships [Neumayr et al. 2010]

Ein M-Cube ist ein Cube von Cubes, bei dem jede Zelle wiederum ein eigener Cube sein kann. Damit ein M-Cube konsistent ist, legen [Neumayr et al. 2010] folgende Kriterien fest:

- Ein M-Cube muss genau eine Wurzel M-Relationship mit der Wurzel Koordinate haben. Die Wurzel Koordinate verbindet die Wurzel Objekte der Dimensionen des M-Cubes.
- Für jede Zelle des Cubes gibt es höchstens eine zugehörige M-Relationship.
- Jede Kennzahl mit einer bestimmten Granularität darf nur auf einer M-Relationship eingeführt werden. Es ist aber möglich, die Kennzahl mit einer genaueren Granularität auf einer Konkretisierung der M-Relationship nochmals zu definieren.
- Wenn zwei überlappende M-Relationships eine Kennzahl enthalten und eine der beiden einen Wert für die Kennzahl definiert, darf die andere der Kennzahl keinen Wert zuweisen. Zwei mehrstellige M-Relationships eines M-Cubes mit n Dimensionen sind überlappend, wenn mindestens ein M-Object der ersten Koordinate eine Konkretisierung des M-Objects der zweiten Koordinate ist, die zweite Koordinate jedoch auch mindestens ein M-Object besitzt, das eine Konkretisierung der ersten Koordinate ist. Ein Beispiel für zwei überlappende M-Relationships wären (Car, March2009, Switzerland) und (Porsche911, 2009, Switzerland).

- Für jede nicht-leere Zelle des Cubes und für jedes Paar von M-Relationships, für die eine Kennzahl auf dem Connection-Level der M-Relationship oder auf einem genaueren Connection-Level definiert wurde, müssen die Einheit und der Basis Connection Level der Kennzahl gleich sein.

3.4. Prototyp eines hetero-homogenen Data Warehouses

Ausgangsbasis dieser Arbeit war die erweiterte Version des Prototyps eines Management-Systems für hetero-homogene Data Warehouses in einer objekt-relationalen Datenbank von [Schütz 2010]. Die Neuerungen der erweiterten Version sind unter [Schütz 2011] beschrieben. Der Prototyp selbst ist unter [HH-DW] erhältlich.
Der als Ausgangsbasis verwendete Prototyp wurde in einer Oracle 11g Datenbank realisiert und nutzt die objekt-relationalen Features von Oracle. Die Konzepte der M-Objects, hetero-homogenen Hierarchien, M-Relationships und M-Cubes von [Neumayr et al. 2010] werden dabei mit Objekttypen abgebildet. Im Allgemeinen besteht jeder Typ aus einem abstrakten Super-Typ und einer speziellen Cube-spezifische Konkretisierung. Diese Konkretisierungen werden dynamisch unter Verwendung der dynamischen SQL Fähigkeiten von PL/SQL erstellt. Dabei wird auf das dbms_sql Paket von Oracle und auf dynamisches SQL mittels EXECUTE IMMEDIATE Statements zurückgegriffen. Die einzelnen Cube-spezifischen Typen werden dynamisch erstellt, da im Vorhinein nicht feststeht, welche Attribute es geben wird und welche Rückgabewerte bzw. wie viele Parameter Methoden haben werden. Beispielsweise hat ein M-Cube mit drei Dimensionen drei Attribute mit den Namen der jeweiligen Dimensionen, ein M-Cube mit vier Dimensionen jedoch vier. Die Konstruktoren der M-Relationships würden somit drei beziehungsweise vier Parameter mit den Namen und/oder Referenzen der Dimensionen bekommen.

Abbildung 7 zeigt die Realisierung von M-Objects und hetero-homogenen Dimensionen von [Schütz 2010]. Das Konzept der M-Objects wird dabei als Typ mobject_ty repräsentiert. Der Typ mobject_ty ist ein abstrakter, vererbbarer Typ, wo der Großteil der Funktionalität von M-Objects realisiert ist. Die Cube-spezifische Konkretisierung des Typs mobject_ty ist der mobject_*_ty Typ. Sie definiert einige zusätzliche Methoden wie etwa zum Persistieren und zum Löschen von M-Objects. Ein „*“ im Typnamen weist daraufhin, dass es sich um einen konkretisierten Typ handelt. Beispielsweise ist mobject_*_ty ein konkretisiertes Objekt. Der

„*“ wird in der tatsächlichen Implementierung ersetzt. Ein M-Object der Dimension „1“ würde also den Typen mobject_d000000001_ty haben.
Jedes M-Object besitzt einen Namen, einen Top-Level, eine Reihe von Elternobjekten, eine Reihe von Vorgängern, eine Level-Hierarchie, eine Reihe von Attributtabellen, Metadaten und eine Referenz auf die Dimension, in der es sich befindet. Eine Referenz wird in Abbildung 7 als <<REF>> dargestellt. Die Elternobjekte sind in einer verschachtelten Tabelle abgespeichert, welche pro Parent lediglich die Referenz auf das Objekt speichert. Die direkten und indirekten Vorgänger Objekte befinden sich zusammen mit deren Top-Level in einer geschachtelten Tabelle *ancestors*. Die Attribute eines M-Objects werden in eigenen Attributtabellen abgespeichert. Grundsätzlich hat jedes Attribut eine eigene Attributtabelle. Führt ein M-Object auf einem Level jedoch mehrere Attribute ein, werden diese in einer Attributtabelle zusammengefasst. Die Attributtabelle eines Attributs wird bei dem M-Object, welches es einführt, gespeichert. Jedes M-Object besitzt also eine verschachtelte Tabelle mit den Namen seiner Attributtabellen und dem Level, auf dem sich die Attribute darin befinden. Die eigentlichen Werte der Attribute werden in der Attributtabelle persistiert. Die Level-Hierarchie eines M-Objects wird als verschachtelte Tabelle als level_hierarchy_tty Typ gespeichert, wobei jedes Tupel ein level_hierarchy_ty ist. Der level_hierarchy_ty besteht wiederum aus einem Level und dem darüber liegenden Parent Level. Man beachte, dass der level_hierarchy_tty im ursprünglichen Prototyp als p_tty bezeichnet wurde.

In der erweiterten Version des Prototyps [Schütz 2011] gehört jedes M-Object zu einer Dimension. Die Dimensionen sind vom Typ dimension_*_ty. Dieser ist eine Konkretisierung des dimension_ty Typs. Jede Dimension hat einen Namen und eine M-Object-Tabelle, in der die einzelnen M-Objects gespeichert werden. Zu beachten ist noch, dass die M-Objects in der M-Object-Tabelle nicht als mobject_ty sondern als Konkretisierung davon, als mobject_*_ty, gespeichert werden. Die Dimensionen selbst werden alle in einer Dimensionstabelle gespeichert. Diese enthält unter anderem den Namen der Dimension, ihre Level-Hierarchie und den Namen der M-Object-Tabelle.

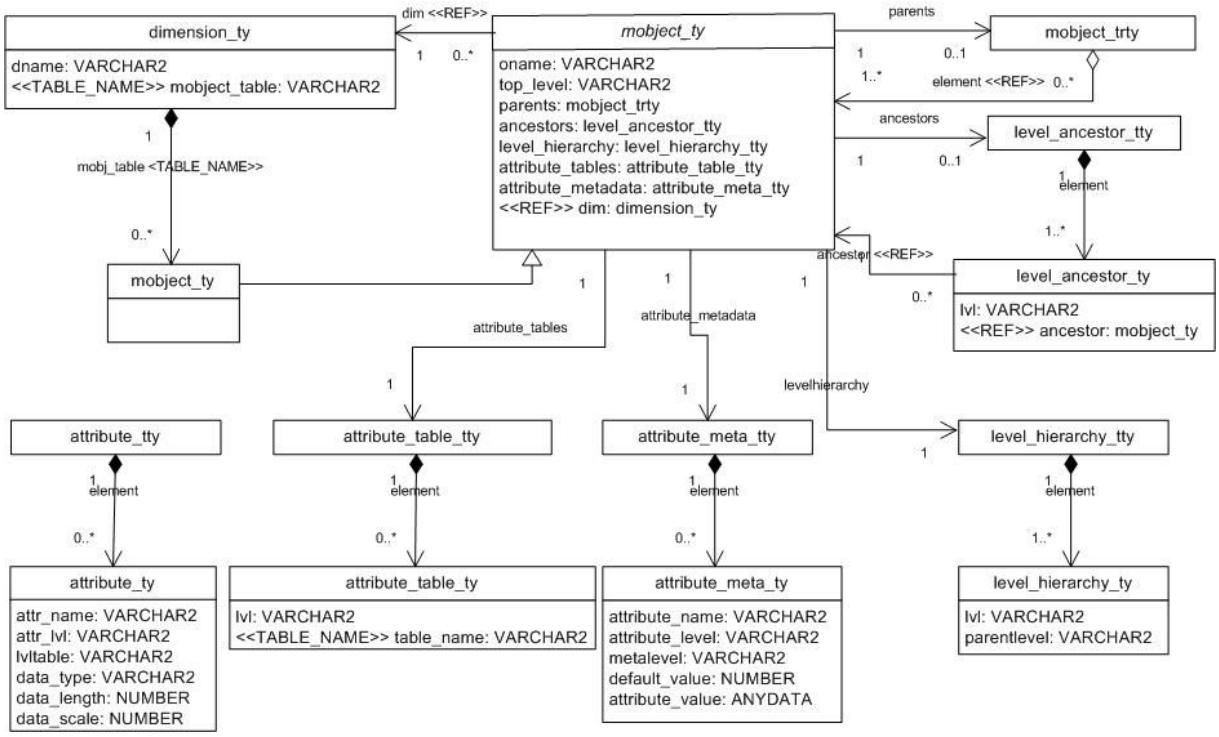

Abbildung 7: M-Object und Dimensions Architektur [Schütz 2010], Änderungen nach [Schütz 2011]

Physisch werden, wie bereits zuvor beschrieben, alle Dimensionen in der Tabelle *dimensions* gespeichert. Hier ist jedes Tupel vom Typ dimension_*_ty. Zudem besitzt jede Dimension eine eigene M-Object-Tabelle, welche die einzelnen M-Objects enthält. Diese Tabelle ist nach der ID der jeweiligen Dimension benannt. Also zum Beispiel *d000000001* für die Produktdimension. Die einzelnen Attribute und die Werte dazu werden in eigenen Attributtabellen hinterlegt. Jeder Level jedes einzelnen M-Objects, für den ein Attribut definiert wurde, wird in einer eigenen Attributtabelle gespeichert. Der Name der Attributtabellen setzt sich aus der ID der Dimension, der ID des M-Objects und dem Level des M-Objects zusammen, also beispielsweise *d000000001_o000000001_category* für die Attribute *Verantwortlicher* und *Steuersatz* des M-Objects Produkt in der Produktdimension.

Zum Anlegen von Dimensionen gibt es eine Methode *create_dimension*. Diese nimmt den Namen der Dimension als Parameter und erzeugt damit die Dimension. M-Objects werden mit der Methode *create_mobject* angelegt. Diese Methode benötigt den Namen des Objekts den Top-Level und verschachtelte Tabellen mit den Parent-M-Objects und der Level-Hierarchie als Parameter. Attribute werden einem M-Object mit der Methode *add_attribute* hinzugefügt. *Add_attribute* benötigt den Namen, den zugehörigen Level und den Datentyp des Attributs als Parameter. Außerdem gibt es eine Methode *has_attribute*, um abzufragen, ob ein M-Object ein bestimmtes Attribut hat.

In [Schütz 2011] stehen zudem Bulk-Methoden zum Hinzufügen von Objekten und zum Setzen von Attributen zur Verfügung. Die Methode *bulk_create_mobject* kann verwendet werden, um M-Objects, die dieselben Elternobjekte und dieselbe Level-Hierarchie haben, gemeinsam in die Dimension einzufügen. Dadurch müssen die Vorfahren nur einmal berechnet und die Konsistenz nur einmal gecheckt werden. Zudem verwendet die Methode eine FORALL-Schleife, was einen zusätzlichen Performance-Gewinn mit sich bringt. Die Methode *bulk_set_attribute* kann verwendet werden, um ein Attribut für mehrere M-Objects zu setzen, wenn das M-Object in derselben Tabelle liegt. Der Vorteil der neu hinzugefügten Bulk-Methoden liegt in der höheren Performance. Details zu den Bulk-Methoden befinden sich in [Schütz 2011].

M-Relationships und M-Cubes werden ebenfalls als abstrakte Super-Typen realisiert, welche dann für den jeweiligen M-Cube konkretisiert werden. Der abstrakte Super-Typ mcube_ty fungiert mehr als Interface, um alle Cubes in einer Tabelle speichern zu können. Er besteht unter anderem aus dem Cube-Namen, den Namen der zugehörigen M-Relationship-Tabelle, einer ID und den Namen der konkretisierenden Typen. Die M-Relationship-Tabelle ist eine Objekt Tabelle mit dem konkretisierten M-Relationship Typ mrel_*_ty. Die Konkretisierung des mcube_ty wird als mcube_*_ty bezeichnet. Sie enthält die Wurzel Koordinate und n-Attribute für die n- Dimensionen des Cubes mit den Referenzen auf die einzelnen Dimensionen.

M-Relationships werden mit den abstrakten Super-Typ mrel_ty bzw. der Konkretisierung mrel_*_ty abgebildet. Der Super-Typ fungiert hier wiederum nur, um Vererbung zu ermöglichen. Er enthält lediglich ein Attribut mit der Referenz auf den Cube, eines mit der ID des Cubes und eine Funktion zum Löschen der M-Relationship. Der konkrete Typ mrel_*_ty besitzt analog zu den Attributen bei den M-Objects eine verschachtelte Tabelle mit den Namen der Fakttabellen und ein Attribut mit den Metadaten des Fakts beziehungsweise der Kennzahl. Für jede Kennzahl, die von einer M-Relationship eingeführt wurde, wird eine neue Fakttabelle angelegt, außer es besteht bereits eine entsprechende Tabelle. Kennzahlen auf dem gleichen Connection Level werden wieder innerhalb einer Tabelle gespeichert. Connection Level definieren die Granularität von Kennzahlen, also beispielsweise <month,city,model> für eine Kennzahl Umsatz.
Weiters hat jede M-Relationship eine Koordinate, die die Position innerhalb des Cubes eindeutig festlegt. Koordinaten und Connection Level sind ebenso Cube-spezifisch, da sie so

viele Dimensionen haben wie der jeweilige Cube. Zum Beispiel besitzt eine Koordinate in einem M-Cube mit drei Dimensionen drei Attribute, die die Namen von M-Objects der jeweiligen Dimensionen enthalten. Allerdings existiert für Koordinaten und Connection Level im Gegensatz zu M-Cubes und M-Relationships kein Basistyp. Die ursprüngliche Architektur der M-Cubes und M-Relationships des Prototyps von [Schütz 2010] wird in Abbildung 8 nochmals genau dargestellt.

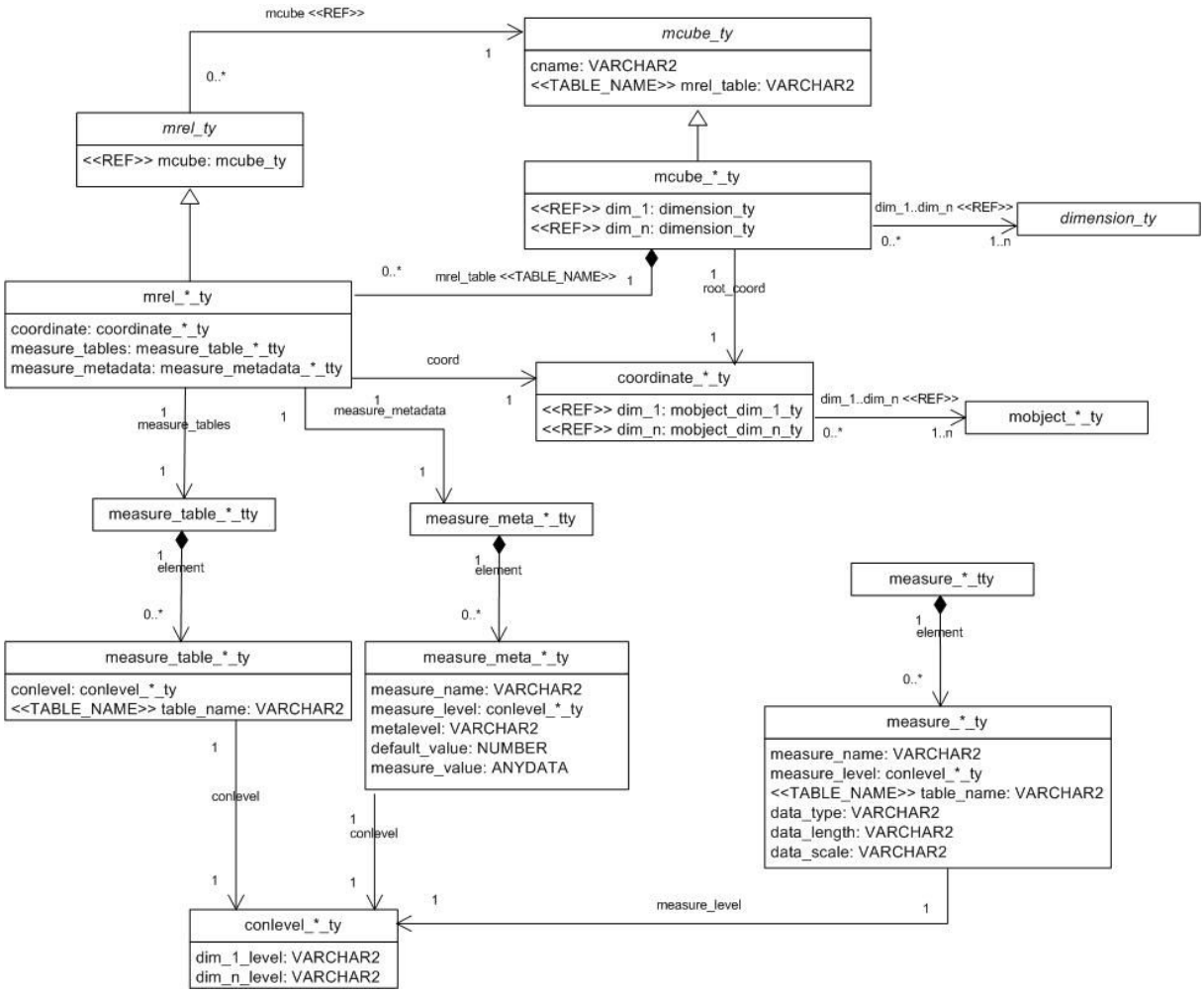

Abbildung 8: M-Cube und M-Relationship Architektur [Schütz 2010], Änderungen nach [Schütz 2011]

Physisch werden die M-Cubes in der Tabelle *mcubes* gespeichert. In dieser ist jeder Datensatz vom Typ mcube_*_ty. Jeder Cube besitzt eine eigene M-Relationship-Tabelle. Diese ist nach der ID des Cubes benannt, also zum Beispiel *c000000001* für einen Sales-Cube. Die M-Relationship-Tabelle enthält alle M-Relationships des Cubes. Jedes Tupel der Tabelle ist vom Typ mrel_*_ty. Die einzelnen Kennzahlen werden analog zu den Attributen in einer eigenen Fakttabelle hinterlegt, welche die Werte der Kennzahlen enthält. Auch hier gibt es für jeden Connection-Level jedes einzelnen M-Objects eine eigene Fakttabelle. Der Name der Fakttabelle setzt sich aus der ID des Cubes, der ID der M-Relationship und den ersten beiden

Buchstaben jedes Levels des Connection-Levels zusammen. Beispielsweise wären Kennzahlen der Wurzel M-Relationship *(<top,top,top>)* des Sales-Cubes in der Tabelle *c000000001_r000000001_tototo* abgespeichert.

Zum Anlegen von M-Relationships stellt der Prototyp von [Schütz 2011] die Methode *create_mrel* zur Verfügung. Diese wird für den M-Cube, in dem die M-Relationship angelegt werden soll, aufgerufen. Die Methode *create_mrel* bekommt entweder die Koordinate der M-Relationship oder die Namen oder die Referenzen der einzelnen M-Objects der Koordinate als Parameter und gibt die neu generierte M-Relationship vom Typ mrel_*_ty zurück. Um einer M-Relationship eine Kennzahl hinzuzufügen, gibt es die Methode *add_measure*. Diese Methode wird für die M-Relationship, der die Kennzahl hinzugefügt werden soll, aufgerufen. *Add_measure* nimmt den Namen der Kennzahl, den Connection Level für den sie hinzugefügt werden soll und den Datentypen als Parameter und fügt der M-Relationship die Kennzahl hinzu. Zum Setzen der Werte der Kennzahlen gibt es verschiedene *set_measure* Methoden. Im einfachsten Fall benötigt die Methode *add_measure* lediglich den Namen und den zu setzenden Wert als Parameter. Mit der *has_measure* Methode gibt es zusätzlich die Möglichkeit zu prüfen ob eine M-Relationship eine Kennzahl hat oder nicht.
Analog zu den M-Objects gibt es auch hier in der erweiterten Version des Prototyps von [Schütz 2010] Bulk-Methoden zum Erstellen von M-Relationships (*bulk_create_mrel*) und zum Setzen der Werte der Kennzahlen (*bulk_set_measure*). Auch hier liegt der Vorteil der Bulk-Methoden darin, dass sie einen Performance-Gewinn bewirken können. Details zu den Bulk-Methoden befinden sich in [Schütz 2011].

4. Integrationsprozess

In diesem Kapitel wird der Integrationsprozess zum Importieren von Data Marts anhand eines konkreten Beispiels beschrieben. Die Integration eines Data Marts ist ein semi-automatischer Prozess. Der Benutzer muss zu Beginn Mappings für die semantischen und strukturellen Heterogenitäten erstellen. Der eigentliche Importvorgang ist überwiegend automatisch, allerdings muss der Benutzer auch hier im Fehlerfall selbst eingreifen.

Die Integration der Daten erfolgt grundsätzlich nach dem Global-As-View-Ansatz, jedoch werden dabei die Daten im Gegensatz zum von [Levy 2000] vorgeschlagenen Ansatz auch physisch integriert. Dies hat den Vorteil, dass für die Beantwortung von Abfragen über das globale Data Warehouse kein Query Rewriting erforderlich ist. Eine Abfrage der Daten kann direkt auf Basis des globalen Schemas an das globale Data Warehouse gerichtet werden. Zudem muss der Query Execution Plan beim Absetzen einer Abfrage keine verteilten Datenbanken unterstützen.

Die physische Integration der Daten bringt jedoch auch Nachteile mit sich. Zum Einen beinhalten Data Warehouses und manchmal auch Data Marts sehr große Datenmengen; das globale Data Warehouse benötigt also sehr viel Speicherplatz, und die mitunter extrem große Datenmenge kann sich negativ auf die Leistung des Systems auswirken. Zum Anderen entstehen bei einer physischen Daten-Integration Probleme hinsichtlich der dynamischen Aspekte [Koch 2001], das heißt die Daten müssen in regelmäßigen Abständen aktualisiert und gewartet werden, da sich die Originaldaten mit der Zeit verändern können.

Um den Integrationsprozess besser erläutern zu können, wird zu Beginn in Kapitel 4.1 ein konkretes Beispiel vorgestellt, anhand dessen in der Folge der komplette Prozess aufgezeigt wird. In Kapitel 4.2 wird die konzeptuelle Vorgehensweise bei der Integration eines Data Marts inklusive der drei Einzelschritte angeführt. Die einzelnen Schritte, die Definition der Mappings durch den Benutzer, das Importieren von Dimensionen und das Importieren des Cubes, werden anschließend in den Kapiteln 4.3 bis 4.5 genauer erklärt. Im Kapitel 4.3 wird zudem eine Übersicht über die verschiedenen, zur Verfügung stehenden Mapping-Typen gegeben und anschließend aufgezeigt, wie und wo die entsprechenden Mappings erstellt werden können. In Kapitel 4.6 wird die Integration eines Data Marts anhand des Beispiels aus der Problemstellung komplett von der Definition der erforderlichen Mappings bis hin zur Integration der Dimensionen und des M-Cubes gezeigt. Anschließend werden in Kapitel 4.7 und 4.8 eine Reihe von Sonderfällen behandelt, die beim Import von Dimensionen beziehungsweise eines Cubes auftreten können. Dabei wird jeweils das Problem aufgezeigt

und die Problembehandlung einschließlich der zu erstellenden Mappings angeführt. Im letzten Abschnitt werden einige Einschränkungen des Import Algorithmus und Sonderfälle, die mit dem aktuellen System nicht behandelt werden können, angeführt.

4.1. Problemstellung - Beispielszenario

In der Folge wird der Integrationsprozess anhand eines konkreten Beispiels erläutert. Die Grundlage dafür bildet ein globales Data Warehouse, in dem die Verkäufe verschiedener Produkte in unterschiedlichen Städten gespeichert sind. Das Data Warehouse hat einen dreidimensionalen Cube mit den Dimensionen: Produkt, Zeit und Ort.
Ziel ist es nun, im Zuge des Integrationsprozesses einen lokalen Data Mart zu importieren, welcher die Daten aller Verkäufe von Musikprodukten in der Schweiz enthält. Der Data Mart hat ebenfalls drei Dimensionen, jedoch hat er zusätzlich zu den Dimensionen Zeit und Ort eine Dimension *Musik* anstatt der Produktdimension.
Abbildung 9 zeigt links die Produktdimension des globalen Data Warehouses und rechts zum Vergleich die Dimension *Musik* des lokalen Data Marts. Daraus geht hervor, dass im Zuge des Integrationsprozesses eine Reihe von Heterogenitäten zu beachten sind. Wie bereits in Kapitel 3.1 angeführt, kann eine Sub-Hierarchie bzw. ein Sub-Cube eine Reihe von Heterogenitäten aufweisen. Dies kann allerdings auch zwischen dem lokalen Schema des Data Marts und dem globalen Schema des Data Warehouses der Fall sein. Beispielsweise können die Level, Attribute und Dimensionen unterschiedlich benannt sein, der lokale Data Mart kann zusätzliche Level und Attribute einführen und die Kennzahlen können eine unterschiedliche Granularität haben. Es sind daher zur Überbrückung der Schema-Heterogenitäten eine Reihe von Mappings zu erstellen. Weiters muss der lokale Cube nicht zwingend dieselbe Anzahl an Dimensionen aufweisen (nähere Informationen zur Integration von Cubes mit mehr bzw. weniger Dimensionen befinden sich in Kapitel 4.8.1 bzw. 4.8.2). Hier wird jedoch zur Illustration vorerst ein lokaler Cube mit derselben Anzahl von Dimensionen verwendet.
Abbildung 9 stellt die Produktdimension des globalen Data Warehouses und des Data Marts gegenüber. Dabei geht hervor, dass die Bezeichnung der Levels und Attribute in den beiden Schemata unterschiedlich ist. Im Data Mart sind sie Deutsch benannt, im globalen Schema Englisch. Dem Level *category* entspricht der Level *kategorie* im Data Mart, und das Attribut *taxRate* wird lokal *steuersatz* bezeichnet. Des Weiteren führt der lokale Data Mart einen zusätzlichen Level *label* und ein zusätzliches Attribut *komponist* ein.

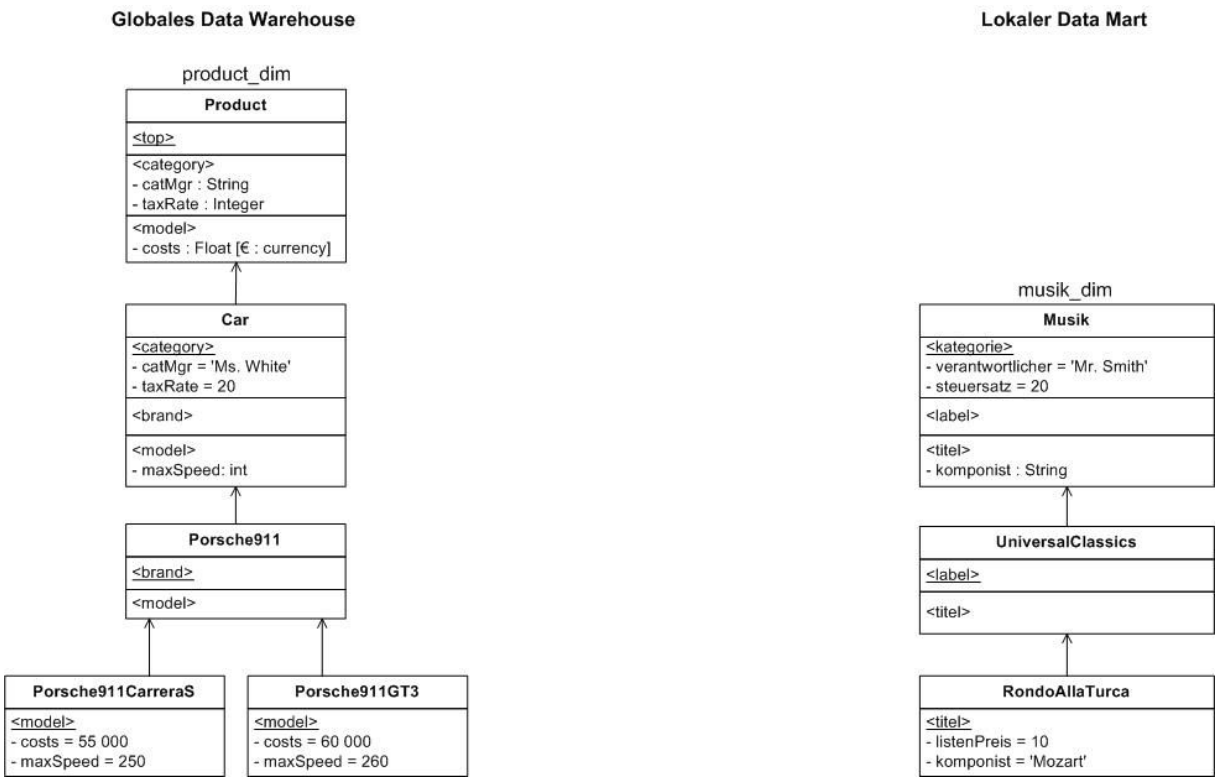

Abbildung 9: Problemstellung - Produktdimension

Um das Fallbeispiel einfach zu halten, wird angenommen, dass die Dimensionen Ort und Zeit des globalen Schemas und des lokalen Data Marts identisch sind, das heißt, dass beide dieselbe Orts- beziehungsweise Zeitdimension verwenden. Abbildung 10 und Abbildung 11 zeigen die verwendeten Dimensionen *Ort* und *Zeit*.

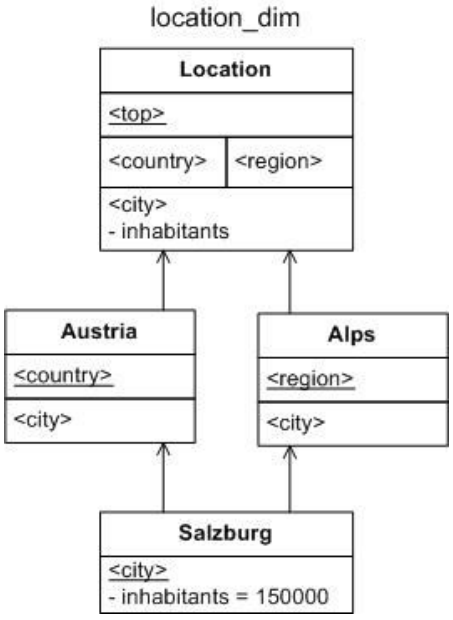

Abbildung 10: Problemstellung - Dimension Ort

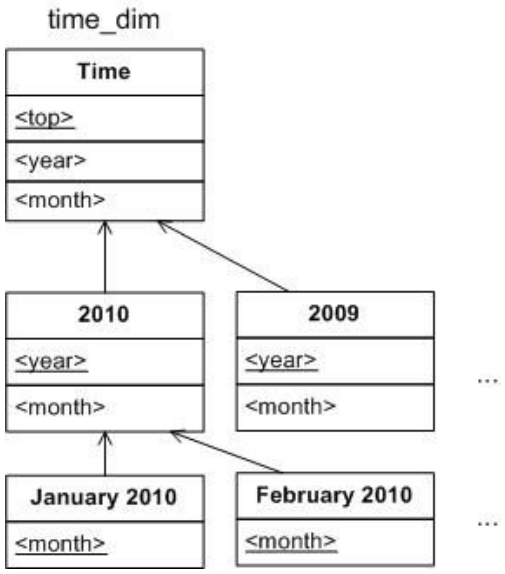

Abbildung 11: Problemstellung - Dimension Zeit

Heterogenitäten können jedoch auch auf Ebene der Cubes auftreten. Ein Sub-Cube beziehungsweise ein lokaler Cube kann zum Beispiel die Fakten und Kennzahlen unterschiedlich bezeichnen. Zudem kann ein Sub-Cube bzw. ein lokaler Cube zusätzliche Kennzahlen einführen, Kennzahlen mit unterschiedlicher Granularität haben oder Kennzahlen mit unterschiedlichen Einheiten besitzen.

Abbildung 12 zeigt einen Vergleich der beiden in diesem Beispiel verwendeten M-Cubes. Auf der linken Seite ist jener des globalen Data Warehouses abgebildet und auf der rechten Seite jener des lokalen Data Marts. Auch hier ist auf Anhieb ersichtlich, dass es Heterogenitäten zwischen den beiden Cubes gibt, die behandelt werden müssen. Beispielsweise gibt es Unterschiede bei den Kennzahlen. Global gibt es nur eine Kennzahl *revenue*, lokal jedoch eine zusätzliche Kennzahl *verkaufte Menge*. Außerdem heißt die Kennzahl *revenue* im lokalen Data Mart *Umsatz* und hat eine andere Granulariätsstufe, das heißt in diesem Fall, dass der Umsatz lokal auf Filialebene gespeichert wird, im globalen Schema jedoch für eine Stadt.

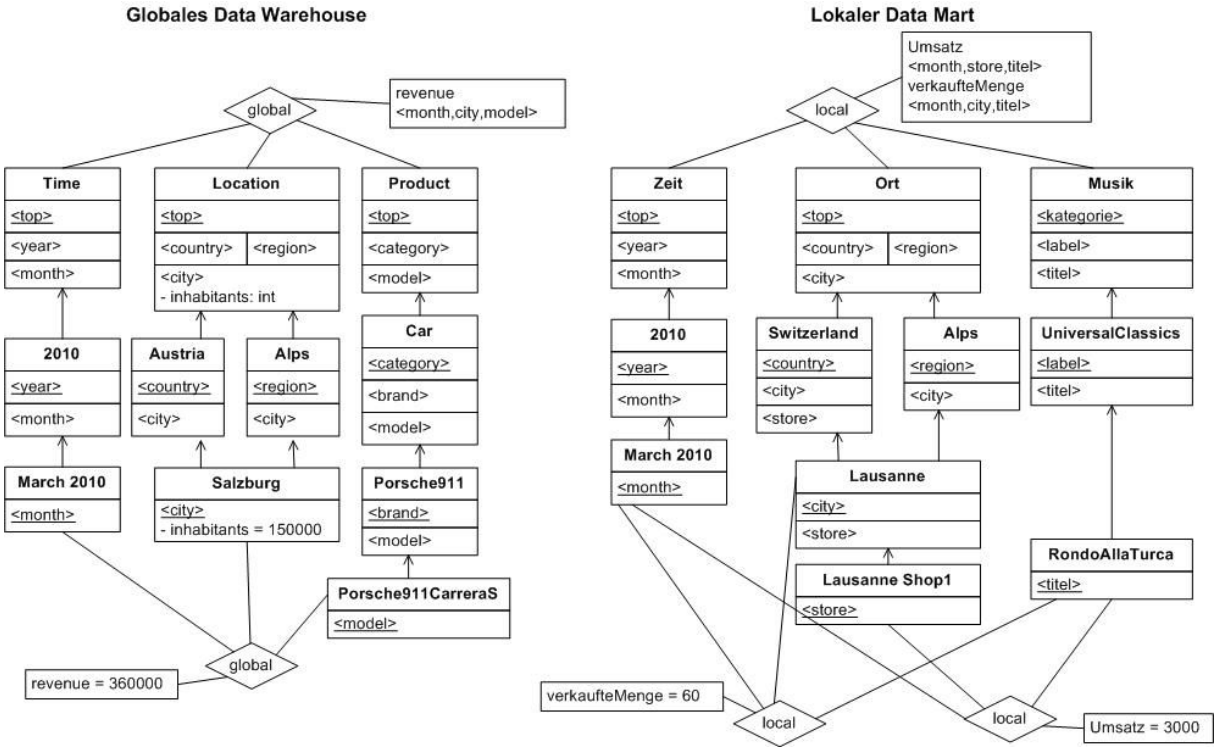

Abbildung 12: Problemstellung - M-Cubes

Neben den im zuvor beschriebenen Beispiel aufgetreten Heterogenitäten kann es auch noch eine Reihe anderer Unterschiede zwischen dem globalen Schema und den lokalen Schemata geben. Diese werden in den Kapiteln 4.7 und 4.8 näher betrachtet:

- Der lokale Data Mart kann zusätzliche Levels haben. Dies wäre beispielsweise der Fall, wenn eine Stadt in der lokalen Ortsdimension nicht dem Land sondern einem Bundesland untergeordnet ist und die einzelnen Bundesländer zu einem Land zusammengefasst werden.
- Im lokalen Data Mart können Levels, die im globalen Schema vorhanden sind, fehlen. Dies wäre der Fall, wenn lokal die Städte zum Land zusammengefasst werden, global jedoch zuerst zu Bundesländern und erst eine Ebene darüber zum Land.
- Der lokale Cube kann mehr Dimensionen haben als der globale.
- Der lokale Cube kann weniger Dimensionen haben als der globale.
- Die Kennzahlen des lokalen Data Marts können eine unterschiedliche Granularität haben als jene des globalen Data Warehouses.
- Die dimensionalen Attribute von Fakten und Kennzahlen können lokal teilweise auf Instanzebene modelliert werden. Es können daher Heterogenitäten in Form von Pivoting und Unpivoting auftreten (vgl. Kapitel 4.3.1.8, 4.8.6 und 4.8.7).

4.2. Vorgehensweise

Der Import eines Data Marts gliedert sich grundsätzlich in drei Schritte, welche in den darauffolgenden Unterkapiteln näher betrachtet werden:

- Definition der Mappings
- Import der Dimensionen
- Import des M-Cubes

Die einzelnen Schritte des Integrationsprozesses müssen stets sequentiell abgearbeitet werden. Jeder Schritt hat als Voraussetzung, dass der Vorgänger abgeschlossen ist.
Allerdings ist der Import selbst ein iterativer Prozess. Es können also einige Schritte wiederholt werden. So ist es zum Beispiel möglich, nach dem Import der Dimensionen, den Integrationsprozess neu zu starten, also neue Mappings zu definieren, den Import der Dimensionen zu wiederholen und erst anschließend den Cube selbst zu importieren. Wird der Import von Dimensionen oder des M-Cubes wiederholt, werden jedoch nur neue beziehungsweise geänderte Daten importiert. Für Daten, die im globalen Data Warehouse bereits bestehen, wird beim abermaligen Import geprüft, ob sich die Daten geändert haben und diese gegebenfalls aktualisiert.

Definition von Mappings

Wie bereits in der Problemstellung hervorgehoben, bestehen Heterogenitäten zwischen dem globalen Data Warehouse und dem lokalen Data Mart, die behandelt werden müssen. Um Heterogenitäten berücksichtigen zu können, müssen Mappings definiert werden. Ein Mapping ist eine Beschreibung der Heterogenitäten. Grundsätzlich gilt, dass aus Namensgleichheit nicht gleich Äquivalenz folgt. Folglich müssen für alle Levels, Attribute, Dimensionen etc., des lokalen Schemas Mappings erstellt werden, um die entsprechenden Levels, Attribute und Dimensionen im globalen Schema zu definieren. Um den Benutzer zu unterstützen, wird jedoch bei Namensgleichheit zwischen dem globalen Schema und den lokalen Schemata für die einzelnen Attribute, Levels usw. automatisch ein Mapping erstellt, sofern nichts anderes festgelegt wurde. Konzeptuell gilt also die Annahme, dass Namensgleichheit nicht Äquivalenz impliziert, praktisch wird dem Benutzer allerdings die Definition der Mappings erleichtert. Wenn also kein gegenteiliges Mapping oder Alias definiert wurde, wird für jedes Level, Attribut etc., das im globalen und im lokalen Schema dieselbe Bezeichnung hat, automatisch ein Mapping angelegt. Ein Alias ist eine alternative Bezeichnung. Aliase werden

für Level, Attribute, M-Objects und Kennzahlen benötigt, für die kein Mapping erstellt werden kann, da diese im globalen Schema nicht existieren, die jedoch unter einem anderen Namen in das globale Schema integriert werden sollen.
Die Unterstützung des Benutzers durch das automatische Anlegen von Mappings gilt für alle Levels, Attribute etc., die im globalen und im lokalen Schema dieselbe Bezeichnung haben, mit Ausnahme der M-Objects. Der Grund hierfür ist, dass es mehrere M-Objects mit demselben Namen geben kann, diese aber trotzdem unterschiedliche Objekte repräsentieren. Beispielsweise kann es in einer Ortsdimension mehrere Orte *Linz* geben, die nicht alle denselben Ort meinen, sondern lediglich gleich heißen.

Wichtig ist, dass die Mappings immer definiert werden müssen, bevor der Import gestartet werden kann. Grundsätzlich gibt es eigene Mappings für den Import von Dimensionen und eigene für den M-Cube-Import. Jene Mappings, die ausschließlich für den Import des Cubes benötigt werden, können auch erst im Anschluss des Dimensionsimports definiert werden. Allerdings empfiehlt es sich, gleich alle Mappings zu erstellen und erst anschließend den Importvorgang zu starten.
Welche Arten von Mappings es genau gibt und wie beziehungsweise wann diese angelegt werden können, wird im Kapitel 4.3 erklärt.

Import der Dimensionen

Wurden alle benötigten Mappings definiert, kann der Import der Dimensionen gestartet werden. Dafür ist lediglich ein Aufruf der zur Verfügung gestellten Import Methode erforderlich. Im Anschluss an den Dimensionsimport kann der Cube selbst importiert werden. Es ist jedoch auch möglich, neue Mappings zu definieren und den Import der Dimensionen zu wiederholen.

Import des M-Cubes

Sobald der Import der Dimensionen abgeschlossen ist und alle benötigten Mappings angelegt wurden, kann der Cube selbst importiert werden. Hierfür ist aus Benutzersicht lediglich ein Aufruf der entsprechenden Import Methode nötig, auf welche später noch genauer eingegangen wird.

4.3. Definition von Mappings

In diesem Abschnitt werden die verfügbaren Mappings aufgelistet und erklärt wofür sie benötigt werden. Anschließend wird anhand des Fallbeispiels aus Kapitel 4.1 illustriert wie man die Mappings anlegen kann. Zuletzt wird auf einige Sonderfälle eingegangen und gezeigt wie diese behandelt werden können.

4.3.1. Mapping Übersicht

Die folgende Aufzählung gibt einen Überblick über alle verfügbaren Mappings. Die einzelnen Mapping-Typen werden anschließend anhand des Beispiels aus Kapitel 4.1 näher beschrieben:

- Level-Mappings
- Attribut-Mappings
- M-Object-Mappings
- Measure-Mappings
- M-Relationship-Mappings
- Dimension-Mappings
- Hidden-Dimension-Mappings
- Pivot Mappings
- Unpivot-Mappings

Die Level-, Attribut- und M-Object-Mappings gehören zur Gruppe der Mappings für den Dimensionsimport. Diese müssen also angelegt werden, bevor die Dimensionen importiert werden. Alle übrigen Mapping-Typen sind lediglich für den Import des M-Cubes relevant und können deshalb auch nach dem Import der Dimensionen spezifiziert werden, jedoch wiederum vor dem Import des M-Cubes.

4.3.1.1. Level-Mappings

Level-Mappings werden benötigt, wenn ein Level im globalen Data Warehouse anders benannt wurde als im lokalen. Im Beispiel aus Kapitel 4.1 sind für die Levels *category, brand* und *model* Level-Mappings erforderlich, da diese im lokalen Data Mart den Levels *kategorie, label* und *titel* entsprechen. Es werden also Mappings benötigt, um festzulegen, dass der Level

category im globalen Schema, dem Level *kategorie* im lokalen Data Mart, *brand label* und *model titel* im lokalen Schema entspricht.

4.3.1.2. Attribut-Mappings

Attribut-Mappings sind erforderlich, wenn Attribute im lokalen Data Mart eine andere Bezeichnung haben als im globalen Schema. Es muss also festgelegt werden, dass *catMgr* im globalen Schema dem Verantwortlichen im lokalen Schema entspricht. Weiters sind für *taxRate-steuersatz* und *costs-listenPreis* Attribut-Mappings nötig.

4.3.1.3. M-Object-Mappings

Mit Hilfe von M-Object-Mappings kann angegeben werden, dass ein Objekt des lokalen Data Marts einem Objekt im globalen Data Warehouse entspricht. Wie bereits zuvor beschrieben, wird aus Namensgleichheit bei M-Objects nicht auf Äquivalenz geschlossen. Es ist daher für jedes M-Object, dass in beiden Schemata besteht ein M-Object-Mapping erforderlich. Andernfalls wird angenommen, dass es sich um ein anderes M-Object mit demselben Namen handelt.
Wird beispielsweise vom lokalen Data Mart eine zusätzliche Ortsdimension importiert, die ebenfalls die Stadt Salzburg enthält, so muss durch ein M-Object-Mapping angegeben werden, dass dies dieselbe Stadt ist. Andernfalls wird angenommen, dass dies eine andere Stadt Salzburg ist. Die lokale Stadt Salzburg würde umbenannt und unter neuem Namen in das globale Schema integriert werden.

4.3.1.4. Measure-Mappings

Measure-Mappings werden benötigt, wenn eine Kennzahl im lokalen M-Cube einen anderen Namen als im globalen M-Cube hat. Im konkreten Beispiel aus Kapitel 4.1 ist dies bei der Kennzahl *revenue* der Fall. Hier wäre ein Measure-Mapping erforderlich, um anzugeben, dass *revenue* der lokalen Kennzahl *Umsatz* entspricht. Die Tatsache, dass sich *Umsatz* auf einer niedrigeren Granularitätsstufe befindet als die globale Kennzahl *revenue*, ist für das Measure-Mapping selbst unerheblich.

4.3.1.5. M-Relationship-Mappings

Mit Hilfe dieses Mapping-Typs kann angegeben werden, dass eine M-Relationship des lokalen M-Cubes einer M-Relationship im globalen M-Cube entspricht. M-Relationship-Mappings sind für jede M-Relationship, die sowohl im globalen als auch im lokalen Cube besteht, zwingend erforderlich.

4.3.1.6. Dimension-Mappings

Wenn der globale Cube und der lokale Cube nicht dieselben Dimensionen verwenden, ist ein Dimension-Mapping für jede Dimension erforderlich, die statt einer globalen Dimension verwendet wird. Beispielsweise besitzt der Cube des globalen Data Warehouses eine Dimension *Produkt*, der lokale Cube jedoch eine Dimension *Musik*. Hier ist also Mapping zwischen den beiden Dimensionen erforderlich.

4.3.1.7. Hidden-Dimension-Mappings

Wie bereits zuvor beschrieben, ist es möglich, Cubes mit weniger Dimensionen als jener des globalen Schemas zu importieren. Hierfür wäre kein spezielles Mapping erforderlich. Jedoch gibt es auch die Möglichkeit, für fehlende Dimensionen ein Hidden-Dimension-Mapping zu erstellen, wenn der lokale Data Mart eine Dimension weniger hat, der Benutzer jedoch weiß, welche Werte für die fehlende Dimension verwendet werden müssen.
Hätte zum Beispiel der lokale Cube lediglich die Dimensionen Musik und Ort, da er ohnehin nur Musikverkäufe des Jahres 2010 enthält, so könnte man ein Hidden-Dimension-Mapping erstellen und angeben, dass alle Verkäufe im Jahr 2010 stattgefunden haben. Würde hierfür kein Hidden Dimension erstellt werden, so würden alle Verkäufe beim Importieren dem Wurzelobjekt der Zeitdimension, *Time*, zugeordnet werden. Es wäre also keine Information über die zeitliche Komponente vorhanden.

4.3.1.8. Pivot Mappings / Unpivot Mappings

Pivot Mappings sind erforderlich, um Schema-Instanz-Konflikte zu lösen. Schema-Instanz-Konflikte entstehen durch die unterschiedliche Modellierung des multi-dimensionalen Data Warehouses und der autonomen Data Marts [Berger 2009].

Data Warehouses erhöhen die Analysierbarkeit der Daten durch die Einführung von Fakten und Kennzahlen. Die Bedeutung der Fakten und Kennzahlen entsteht auf Schema Ebene durch ihre dimensionalen Elemente. Beispielsweise hat die Kennzahl Umsatz = 30000 mit den Dimensionswerten März 2010, Salzburg und Musik für die Dimensionen Zeit, Ort und Produkt die Bedeutung, dass mit Produkten der Kategorie Musik in Salzburg im März 2010 30.000 € Umsatz erzielt wurde. Der Wert der Kennzahl erhält also durch die Kombination der dimensionalen Werte eine Bedeutung [Berger 2009].

Es kann jedoch auch der Fall auftreten, dass ein Teil des Kontextes der Kennzahlen auf Instanzebene modelliert wird. Dies kann erfolgen, indem die Kennzahlen mit speziellen Namen versehen werden und ihnen somit eine spezielle Bedeutung gegeben wird [Berger 2009]. Wenn nun zum Beispiel im obigen Beispiel die Dimension Produkt entfällt und man dafür eine Kennzahl MusikUmsätze einführt, so könnte man durch MusikUmsätze = 30000 mit den Dimensionswerten Monat = März 2010 und Ort = Salzburg ebenfalls ausdrücken, dass mit Produkten der Kategorie Musik in Salzburg im März 2010 ein Umsatz von 30.000 € erzielt wurde. Die Bedeutung der Kennzahlen bliebe also gleich, sie wurden lediglich auf unterschiedliche Weise modelliert.

Grundsätzlich gibt es zwei Formen von Pivoting, welche hier in weiterer Folge als Pivoting und Unpivoting bezeichnet werden, aber beide Pivot Operationen darstellen. Der einzige Unterschied zwischen ihnen ist die Perspektive, mit der sie betrachtet werden, also die Richtung des Pivot Operators. Pivoting und Unpivoting behandeln beim Import von Data Marts Heterogenitäten aufgrund von Schema-Instanz-Konflikten:

- Pivoting:

  Pivoting vermindert die Dimensionalität des M-Cubes – das heißt die Anzahl der Dimensionen. Dies wäre zum Beispiel der Fall, wenn der lokale Cube eine Kennzahl Umsatz und eine zusätzliche Dimension Produkt mit einem Level Produktkategorie hat, das globale Schema allerdings nur aus den Dimensionen Zeit und Ort besteht, zusätzlich jedoch zwei Kennzahlen MusikUmsätze und ProduktUmsätze hätte.

- Unpivoting:

  Unpivoting ist die umgekehrte Richtung des Pivot Operators, das heißt hier wird die Dimensionalität erhöht. Also beispielsweise, wenn der globale Cube eine Kennzahl Umsatz und eine zusätzliche Dimension Produkt mit einem Level Produktkategorie hat, das lokale Schema jedoch nur aus den Dimensionen Zeit und Ort besteht, zusätzlich aber zwei Kennzahlen MusikUmsätze und ProduktUmsätze hätte.

4.3.2. Definition der Mappings

Nachdem im vorherigen Kapitel gezeigt wurde, welche Mapping-Typen es gibt, wird in diesem Kapitel erklärt, wie die einzelnen Mappings erstellt werden können und an welcher Stelle sie zu erstellen sind. Grundsätzlich wird in drei verschiedene Arten von Mappings unterschieden:

- In Mappings, die bei M-Objects gespeichert werden
- In M-Cube Mappings
- In M-Relationship Mappings

Zur ersten Gruppe gehören die Level-, die Attribut- und die M-Object-Mappings. Diese Mappings werden bei M-Objects im globalen Schema gespeichert, genauer gesagt beim Wurzelobjekt, unter dem die lokale Dimension integriert werden soll. Grundsätzlich sollten die Mappings für Kindobjekte vererbt werden. In der derzeitigen Implementierung werden jedoch nur Mappings, die bei der globalen Wurzel definiert wurden, berücksichtigt. Die Mappings werden noch nicht vererbt.
Tabelle 1 listet die Methoden zum Hinzufügen der Mappings, die beim M-Object gespeichert werden:

Methode	Parameter	Beschreibung
add_level_mapping	global_level Varchar2, local_obj Ref mobject_ty, local_level Varchar2	Fügt ein Level-Mapping hinzu.
add_attribute_mapping	global_level Varchar2, global_attr Varchar2, local_obj Ref mobject_ty, local_lvl Varchar2, local_attr Varchar2	Fügt ein Attribut-Mapping hinzu.
add_same_as_mapping	local_obj Ref mobject_ty, local_attr Varchar2, priority Number	Fügt ein M-Object-Mapping hinzu und setzt die Prioritäten für die Attributwerte.
add_same_as_mapping	local_obj Ref mobject_ty, attr_priorities priority_tty	Fügt ein M-Object-Mapping hinzu und setzt die Prioritäten für die Attributwerte.

Tabelle 1: M-Object Mappings

Die Methode *add_level_mapping* wird zum Hinzufügen von Level-Mappings verwendet. Sie benötigt den globalen Level-Namen, eine Referenz des lokalen Wurzelobjekts und den lokalen Level-Namen. Das folgende Beispiel legt ein Level-Mapping für den Level *kategorie*

an. *Local_root_obj* ist dabei eine Referenz auf das M-Object Musik in der lokalen Dimension *Musik*:

```
global_root.add_level_mapping('category', local_root_obj, 'kategorie');
```

Add_attribute_mapping nimmt die globalen Level und Attribut Namen, eine Referenz auf das lokale Wurzelobjekt und die lokalen Level und Attribut Namen als Parameter und erstellt daraus ein Attribut-Mapping. Ein Attribut-Mapping für das Attribut *verantwortlicher* müsste folgendermaßen erstellt werden:

```
global_root.add_attribute_mapping('category', 'catMgr', local_root_obj,
'kategorie', 'verantwortlicher');
```

Zum Erstellen von M-Object-Mappings gibt es zwei Methoden. Die erste Methode bekommt eine Referenz des lokalen M-Objects, ein Attribut und eine Priorität als Parameter. Wenn beispielsweise eine Dimension Ort in das globale Data Warehouse importiert werden soll, welche ebenfalls eine Stadt Salzburg enthält, muss ein M-Object-Mapping erstellt werden, um der Import Methode mitzuteilen, dass es sich dabei um dieselbe Stadt handelt. Wenn nun die lokale Stadt ebenso wie die globale ein Attribut Einwohner hat, so kann eine Priorität mit gegeben werden, ob der globale Wert weiterhin behalten (priority = 0) oder ob der lokale Wert übernommen werden soll (priority = 1). Für das M-Object-Mapping ist es unerheblich, ob das Attribut im globalen und im lokalen Schema gleich benannt ist oder nicht. Zu beachten ist jedoch, dass dem M-Object-Mapping der globale Attributname mitgegeben werden muss. Der folgende Codeausschnitt zeigt, wie ein M-Object-Mapping für die Stadt Salzburg in der globalen Dimension Ort angelegt werden kann, bei dem der Wert des Attributs *inhabitants* der lokalen Dimension verwendet werden soll. *Local_obj_ref* ist dabei eine Referenz auf das lokale M-Object Salzburg:

```
obj_salzburg.add_same_as_mapping(local_obj_ref, 'inhabitants', 1);
```

Die zweite Methode zum Hinzufügen eines M-Object-Mappings kann verwendet werden um die Prioritäten von mehreren Attributen zu setzen oder wenn keine Prioritäten gesetzt werden sollen. Diese Methode benötigt lediglich die Referenz des lokalen Objekts und eine Tabelle aus Attributen und ihrer Prioritäten, welche auch leer sein kann:

```
global_root.add_same_as_mapping(local_obj_ref, priority_tty());
```

Der folgende Codeausschnitt, zeigt wie ein M-Object-Mapping für das M-Object Salzburg erstellt werden kann, wenn das globale und lokale Schema die Attribute *inhabitants* und *area* haben:

```
global_root.add_same_as_mapping(local_obj_ref, priority_tty(
    priority_ty(inhabitants, 1),
    priority_ty(area, 0)));
```

Die zweite Gruppe von Mapping-Typen wird beim globalen M-Cube gespeichert. Dazu gehören die Dimension-Mappings und die Hidden-Dimension-Mappings. Tabelle 2 listet die Methoden zum Hinzufügen der Mappings, die beim M-Cube gespeichert werden:

Methode	Parameter	Beschreibung
add_dimension_mapping	global_dim_ref Ref dimension_ty, local_dim_ref Ref dimension_ty, local_cube_ref Ref mcube_ty	Fügt ein Dimension-Mapping hinzu.
add_hidden_dim_mapping	local_cube_ref Ref mcube_ty, dimension_name VARCHAR2, object_ref Ref mobject_ty	Fügt ein Hidden-Dimension-Mapping hinzu.

Tabelle 2: M-Cube Mappings

Die Methode *add_dimension_mapping* erstellt Dimension-Mappings. Sie benötigt die Referenz der globalen Dimension, die Referenz der lokalen Dimension und die Referenz des lokalen Cubes. Um zum Beispiel ein Dimension-Mapping für die Produkt und die Musik Dimension zu erstellen (siehe Problemstellung in Kapitel 4.1), müsste man folgendermaßen vorgehen:

```
global_cube.add_dimension_mapping(product_dim_ref, musik_dim_ref,
local_cube_ref);
```

Add_hidden_dim_mapping nimmt die Referenz auf den lokalen M-Cube, den Namen der fehlenden Dimension und eine Referenz auf das Objekt mit dem Wert für die fehlende Dimension als Parameter und erstellt ein Hidden-Dimension-Mapping. Das folgende Statement erstellt ein Hidden-Dimension-Mapping für einen Cube ohne Zeit-Dimension, bei dem alle Verkäufe 2010 stattgefunden haben. *Obj_year2010_ref* ist eine Referenz auf das M-Object *Year2010* in der globalen Dimension Zeit:

```
global_cube.add_dimension_mapping(lcube_ref, 'Zeit_dimension',
obj_year2010_ref);
```

Die dritte Gruppe von Mapping-Typen wird bei den M-Relationships gespeichert. Zur dritten Gruppe gehören die M-Relationship-, die Measure-, die Pivot- und die Unpivot-Mappings. Auch hier werden derzeit alle Mappings außer den M-Relationship-Mappings bei der Wurzel im globalen Cube gespeichert. Außerdem werden hier die Mappings im derzeitigen System ebenfalls noch nicht weitervererbt. Die M-Relationship-Mappings hingegen werden nicht bei der Root-M-Relationship, sondern direkt bei der jeweiligen M-Relationship im globalen Cube hinzugefügt.

Tabelle 3 listet alle verfügbaren Methoden um Mappings zu M-Relationships hinzuzufügen:

Methode	Parameter	Beschreibung
add_same_as_mapping	global_mrel Ref mrel_ty, measure Varchar2, priority Number	Fügt ein M-Relationship-Mapping hinzu und setzt die Priorität für den Wert einer Kennzahl.
add_same_as_mapping	global_mrel Ref mrel_ty, measure_priorities mrel_priority_tty	Fügt ein M-Relationship-Mapping hinzu und setzt Prioritäten für die Werte von Kennzahlen.
add_measure_mapping	global_measure Varchar2, local_measure Varchar2, lcube_ref Ref mcube_ty	Fügt ein Measure-Mapping hinzu.
add_pivot_mapping	global_measure Varchar2, local_measure Varchar2, object_ref Ref mobject_ty, cube_ref Ref mcube_ty	Methode zum Erstellen eines Pivot-Mappings.
add_pivot_mapping	global_measure Varchar2, local_measure Varchar2, object_refs mobject_trty, cube_ref Ref mcube_ty	Methode zum Erstellen eines Pivot-Mappings.
add_unpivot_mapping	global_measure Varchar2, local_measure Varchar2, object_ref Ref mobject_ty, cube_ref Ref mcube_ty	Methode zum Erstellen eines Unpivot-Mappings.
add_unpivot_mapping	global_measure Varchar2, local_measure Varchar2, object_refs mobject_trty, cube_ref Ref mcube_ty	Methode zum Erstellen eines Unpivot-Mappings.

Tabelle 3: M-Relationship Mappings

Ähnlich der *add_same_as_mapping* Methoden für die M-Objects gibt es auch bei den M-Relationships zwei verschiedene Methoden. Analog zum Setzen der Prioritäten mehrerer

Attributwerte gibt es hier für die Werte der Kennzahlen eine Methode, wo zusätzlich zur Referenz der lokalen M-Relationship für eine konkrete Kennzahl angegeben werden kann, ob der globale oder der lokale Wert genommen werden soll. Der zweiten Methode kann eine verschachtelte Tabelle mit Kennzahl Prioritäten mitgegeben werden. Die Priorität ist wiederum der Wert „0“, wenn der globale Wert beibehalten, und der Wert „1“, wenn der lokale Wert verwendet werden soll. Ein M-Relationship-Mapping wird immer der betroffenen M-Relationship im globalen Cube hinzugefügt.

```
global_mrel.add_same_as_mapping(local_ mrel_ref, 'revenue', 1);
```

Alternativ, wenn man keine oder gleich mehrere Kennzahl-Prioritäten setzen will, kann man auch hier wieder die Prioritätentabelle direkt angeben:

```
global_mrel.add_same_as_mapping(local_mrel_ref, mrel_priority_tty());
```

Wenn zum Beispiel im globalen Data Warehouse und im lokalen Data Mart Werte für den Umsatz bzw. die verkaufte Menge in der Stadt Innsbruck, im März 2010 für das Produkt *RondoAllaTurca* existieren, so könnte man folgendermaßen angeben, dass der Wert der verkauften Menge beibehalten, der Wert des Umsatzes jedoch durch den lokalen Wert überschrieben werden soll:

```
global_mrel.add_same_as_mapping(local_mrel_ref,
            mrel_priority_tty(mrel_priority_ty('qtySold', 0),
                              mrel_priority_ty('revenue', 1)));
```

Die Prozedur *add_measure_mapping* ermöglicht es, Measure-Mappings zu erstellen. Dies wird zum Beispiel beim *Umsatz* benötigt, um anzugeben, dass dies der Kennzahl *revenue* im globalen Schema entspricht. Die Methode benötigt den globalen und den lokalen Namen der Kennzahl sowie die Referenz des lokalen Cubes, für den das Mapping gilt, als Parameter und wird folgendermaßen aufgerufen:

```
root_mrel.add_same_as_mapping('revenue', 'Umsatz', local_cube_ref);
```

Add_pivot_mapping nimmt die Namen der globalen und lokalen Kennzahlen, die Referenz auf das Objekt, welches den dimensionalen Wert enthält und die Referenz des lokalen Cubes als Parameter und erstellt ein Pivot Mapping. Wenn man beispielsweise im globalen Cube nur die Dimensionen Zeit und Ort hat und dafür zwei Kennzahlen *musicRevenues* und *carRevenues*, könnte man die Pivot Mappings für die zwei Kennzahlen wie im unten angeführten Code Beispiel erstellen. *Local_music_obj_ref* ist dabei eine Referenz auf das M-

Object Musik in der lokalen Dimension Musik und *local_car_obj_ref* eine Referenz auf das M-Object Auto:

```
root_mrel.add_pivot_mapping('musicRevenue', 'Umsatz',
      local_music_obj_ref, local_cube_ref);
root_mrel.add_pivot_mapping('carRevenue', 'Umsatz', local_car_obj_ref,
      local_cube_ref);
```

Anstatt einer einzelnen Objektreferenz könnte man der Methode auch eine verschachtelte Tabelle mit Objektreferenzen mitgeben. Dies wird insbesondere benötigt, wenn es lokal mehrere zusätzliche Dimensionen gibt, also wenn es zum Beispiel eine Kennzahl musicRevenue2010, eine Kennzahl musicRevenue2009 usw. gäbe. In diesem Fall müssten der Methode die Referenzen auf die Objekte Musik und Year2010 beziehungsweise Musik und Year2009 mitgegeben werden. Das folgende Beispiel zeigt, wie die Methode zum Hinzufügen von Pivot Mappings bei bloß einer zusätzlichen Dimension Produkt verwendet werden kann:

```
root_mrel.add_pivot_mapping('musicRevenue', 'Umsatz',
      mobject_trty(local_music_obj_ref), local_cube_ref);
root_mrel.add_pivot_mapping('carRevenue', 'Umsatz',
      mobject_trty(local_car_obj_ref), local_cube_ref);
```

Die Methode *add_unpivot_mapping* wird benötigt, um Unpivot-Mappings zu erstellen. Sie nimmt ebenfalls wie die *add_pivot_mapping* Methode die Namen der globalen und lokalen Kennzahlen, die Referenz auf das Objekt, welches den dimensionalen Wert enthält und die Referenz des lokalen Cubes als Parameter. Sie wird benötigt, wenn der lokale Cube nur die Dimensionen Zeit und Ort hätte und dafür anstatt der Kennzahl Umsatz zwei Kennzahlen MusikUmsätze und AutoUmsätze:

```
root_mrel.add_unpivot_mapping('revenue', 'MusikUmsätze',
      global_music_obj_ref, local_cube_ref);
root_mrel.add_unpivot_mapping('revenue', 'AutoUmsätze',
      global_car_obj_ref, local_cube_ref);
```

Auch hier gibt es als Alternative eine zweite Methode, welche verwendet werden kann um gleich eine ganze Tabelle an Objektreferenzen mitzugeben:

```
root_mrel.add_unpivot_mapping('revenue', 'MusikUmsätze',
      mobject_trty(global_music_obj_ref), local_cube_ref);
root_mrel.add_unpivot_mapping('revenue', 'AutoUmsätze',
      mobject_trty(global_car_obj_ref), local_cube_ref);
```

4.4. Import von Dimensionen

Wenn alle Level-, Attribut- und M-Object-Mappings erstellt wurden, können die Dimensionen importiert werden. Dafür gibt es eine spezielle *import_branch* Methode, welche sich im *dimension* Package befindet:

Methode	Parameter	Beschreibung
import_branch	global_root Ref mobject_ty, local_root Ref mobject_ty, object_aliases aliases_tty, level_aliases aliases_tty, attribute_aliases aliases_tty	Methode zum Importieren einer Dimension. Die Attribute object_aliases, level_aliases und attribute_aliases sind optional und können auch NULL sein.

Die Methode *import_branch* benötigt ein Objekt der globalen und ein Objekt der lokalen Dimension als Parameter. Das globale Objekt ist der Punkt in der globalen Dimension, unter dem die lokale Dimension eingefügt werden soll, das Lokale ist die Wurzel des Teilbaums der lokalen Dimension, der importiert werden soll. Wenn die gesamte lokale Dimension importiert werden soll, muss einfach die Wurzel der lokalen Dimension als Parameter mitgegeben werden. Optional können auch noch Aliase für die Namen bestimmter Objekte, Levels oder Attribute mitgegeben werden.
Der folgende Codeausschnitt zeigt, wie die Dimension Musik in die Produktdimension integriert werden kann. Die Methode bekommt eine Referenz auf das globale M-Object *Product*, eine Referenz auf das lokale Wurzelobjekt *Musik* und Aliase für die Objektnamen, die Levels und die Attribute als Parameter:

```
dimension.import_branch(product_obj_ref, musik_obj_ref,
    aliases_tty(aliases_ty('Music', 'Musik')),
    aliases_tty(aliases_ty('music_label', 'label')),
    aliases_tty(aliases_ty('composer', 'Komponist')));
```

Will man keine Aliase angeben, gibt man der Methode einfach NULL Werte mit:

```
dimension.import_branch(
    product_obj_ref, musik_obj_ref, NULL, NULL, NULL);
```

4.5. Import eines M-Cube

Nachdem alle Mappings definiert und die Dimensionen importiert wurden, kann die Methode *import_mcube* verwendet werden, um den Cube des lokalen Data Marts zu importieren:

Methode	Parameter	Beschreibung
import_branch	global_root_mrel Ref mrel_ty, local_root_ref Ref mrel_ty, measure_aliases aliases_tty	Methode zum Importieren eines M-Cubes. Da Attribut measure_aliases ist optional und kann somit auch NULL sein.

Die *import_mcube* Methode befindet sich im *mcube* Package und benötigt drei Parameter:

- global_root_mrel:
  Wurzel M-Relationship des M-Cubes des globalen Data Warehouses, unter der der neue Teilcube eingefügt werden soll.
- local_root_mrel:
  Wurzel M-Relationship des lokalen Cubes. Wird benötigt, um anzugeben, welcher Teil-Cube importiert werden soll. Wenn die Wurzel des lokalen Cubes angegeben wird, wird der gesamte lokale Cube importiert.
- measure_aliases:
  Wenn Kennzahlen, welche im globalen Schema nicht existieren, unter einem anderen Namen importiert werden sollen, müssen die Namen der Kennzahlen der Import Methode in Form von Aliase als Parameter mitgegeben werden. Wenn keine Aliase benötigt werden, kann der Methode auch NULL als Parameter übergeben werden.

Das folgende Beispiel zeigt, wie der lokale Sales-Cube in den globalen integriert werden kann. Global_root_ref ist eine Referenz auf die M-Relationship {Time, Location, Product} im globalen Cube, local_root_ref ist eine Referenz auf die M-Relationship {Time, Location, Musik} im lokalen Cube. Zusätzlich wird ein Alias mitgegeben, um die Kennzahl *verkaufteMenge* im globalen Data Warehouse als *qtySold* zu behandeln:

```
mcube.import_mcube(global_root_ref, local_root_ref,
      aliases_tty(aliases_ty('qtySold', verkaufteMenge');
```

4.6. Mapping Beispiel

Im folgenden Abschnitt wird nochmals illustriert, welche Mappings für das Beispiel aus Kapitel 4.1 erforderlich sind. Dazu werden zuerst alle Mappings für den Import der Dimension *Musik* definiert und die Dimension anschließend importiert. Zuletzt werden die für den Cube Import erforderlichen Mappings angelegt und der lokale Cube importiert.

Abbildung 13 zeigt die erforderlichen Level- und Attribut-Mappings für die Produkt Dimension. Alle Mappings, die für den Import einer Dimension erforderlich sind, werden beim Einfügepunkt in der globalen Dimension, hier beim M-Object *Product*, dauerhaft gespeichert. Die erforderlichen Mappings sind in der Abbildung in rot gekennzeichnet. Die rot strichlierte Linie zum M-Object Produkt weist darauf hin, dass die Mappings diesem Objekt hinzugefügt werden müssen. *Category* ≡ *musik_dim.kategorie* sagt aus, dass der Level *kategorie* der lokalen Dimension *Musik* dem Level *category* im globalen Schema entspricht. *Category.catMgr* ≡ *musik_dim.kategorie.verantwortlicher* stellt ein Attribut-Mapping für das lokale Attribut *verantwortlicher* auf dem Level *kategorie* dar, welches global dem Attribut *catMgr* entspricht. Generell gilt, dass alle Neuerungen im Schema in den folgenden Abbildungen immer rot hervorgehoben werden.

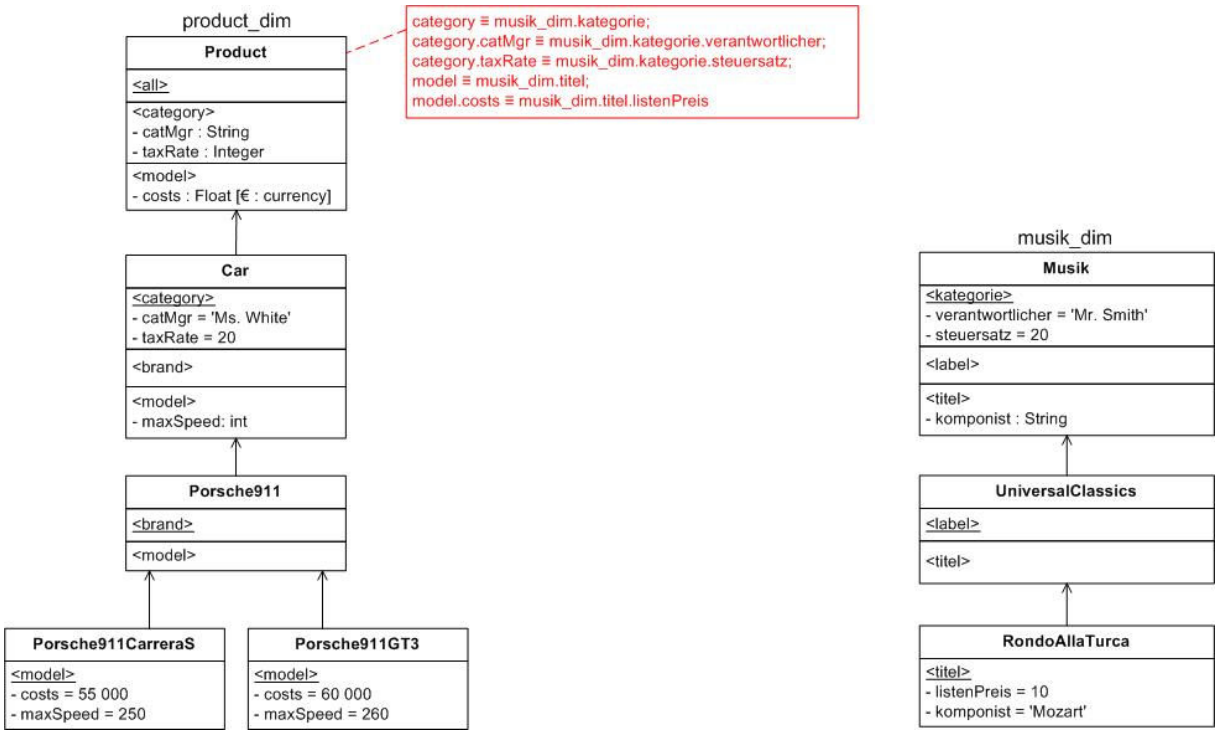

Abbildung 13: Mappings Problemstellung – Produkt Dimension

Der folgende Code Ausschnitt zeigt wie die Mappings erstellt werden können:

```
Declare
    obj mobject_ty;
    obj_ref REF mobject_ty;
Begin
    -- get the root M-Object of the global Dimension
    Select Value(o) Into obj From product_dim o Where o.oname =
        'Product';


    -- get the local root M-Object
    Select REF(o) Into obj_ref From musik_dim o Where o.oname = 'Musik';

    -- add the level mappings to the global root object
    obj.add_level_mapping('category', obj_ref, 'kat');
    obj.add_level_mapping('model', obj_ref, 'titel');

    -- add the attribute mappings to the global root object
    obj.add_attribute_mapping('category', 'catMgr', obj_ref, 'kat',
        'verantwortlicher');
    obj.add_attribute_mapping('category', 'taxRate', obj_ref, 'kat',
        'steuersatz');
    obj.add_attribute_mapping('model', 'costs', obj_ref, 'kat',
        'listenpreis');
End;
```

Da lediglich für jene Attribute Mappings angelegt werden, die im globalen Schema beim Einfügepunkt auch vorhanden sind, kann für den Level *label* und das Attribut *Komponist* kein Mapping erstellt werden. Wenn diese unter einer anderen Bezeichnung im globalen Schema gespeichert werden sollen, müssen die lokalen und die globalen Namen der Import Methode in Form eines Alias als Attribut mitgegeben werden. Dasselbe gilt auch für Objekte, die unter einem anderen Namen im globalen Schema eingefügt werden sollen. Abbildung 14 veranschaulicht die Attribute (in rot gekennzeichnet) die der Import Methode zusätzlich in Form von Level-, Attribut- und M-Object-Aliase mitgegeben werden.

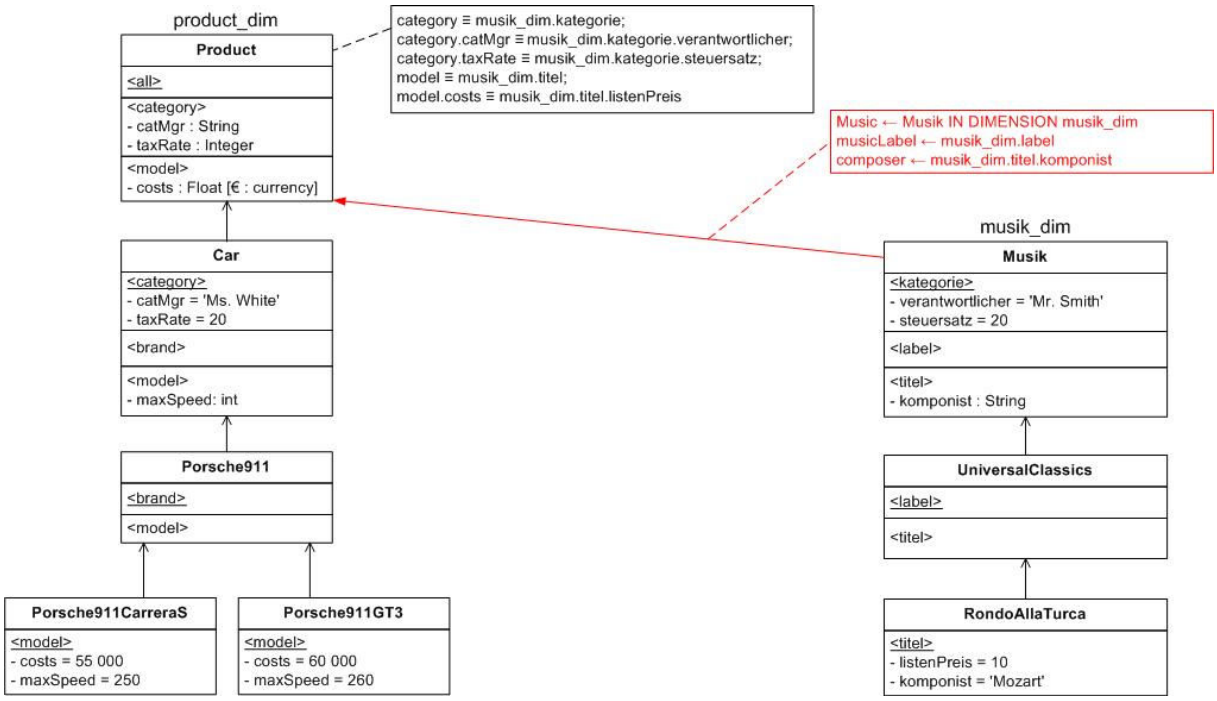

Abbildung 14: Mappings Problemstellung – Zusätzliche Attribute

Der folgende Code zeigt wie die Dimension importiert werden kann:

```
Declare
    local_obj_ref REF mobject_ty;
    global_obj_ref REF mobject_ty;

Begin
    -- get the parent M-Object of the global dimension
    Select Ref(o) Into global_obj_ref From product_dim o Where o.oname =
        'Product';

    -- get the local root M-Object
    Select Ref(o) Into local_obj_ref From musik_dim o Where o.oname =
        'Musik';

    --Test of the standard import method
    dimension.import_branch(global_obj_ref, local_obj_ref,
                  aliases_tty(aliases_ty('Music','Musik')),
                  aliases_tty(aliases_ty('music_label','label')),
                  aliases_tty(aliases_ty('composer','Komponist')));
End;
```

Abbildung 15 zeigt die fertig importierte Dimension *Musik*. Alle neu integrierten M-Objects sind dabei in rot hervorgehoben. Alle Attribute der Import Methode, außer den Namensänderungen der Objekte, werden beim Import ebenfalls in Mappings übersetzt und gespeichert. Die rot strichlierte Line weist wiederum daraufhin, dass die generierten Mappings beim M-Object *Music* gespeichert werden:

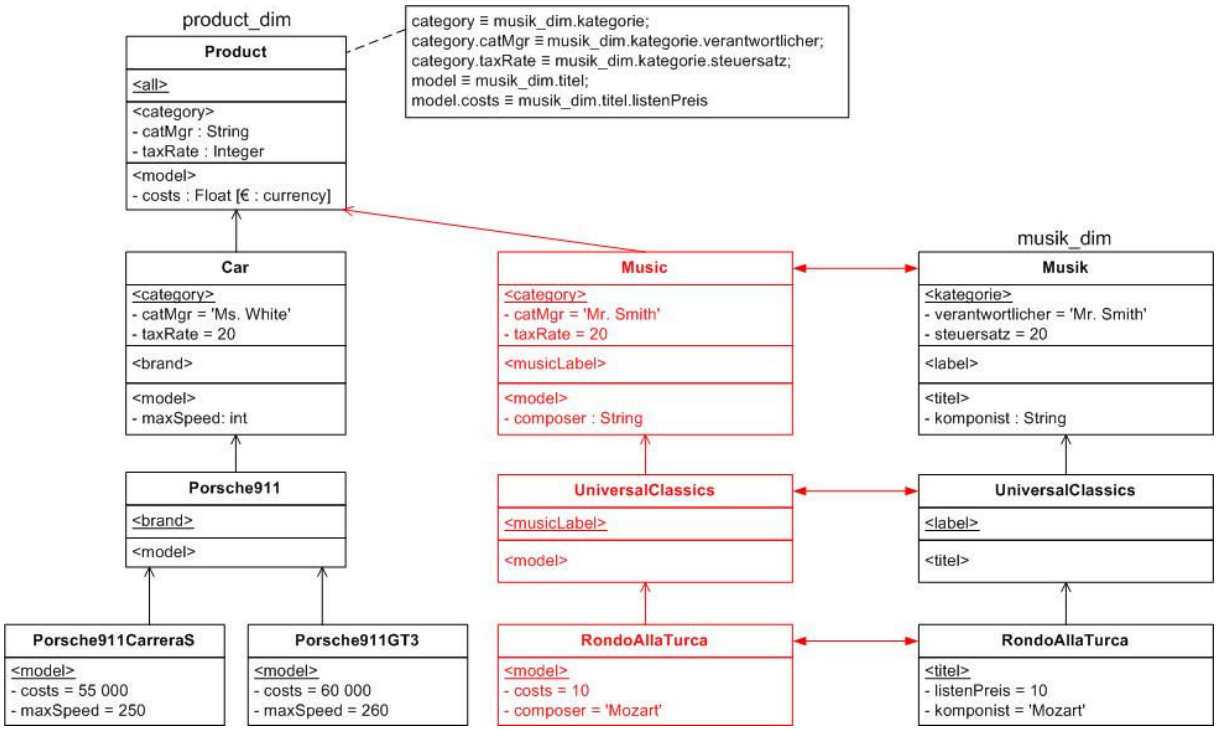

Abbildung 15: Problemstellung - Importierte Dimension Musik

Bevor der M-Cube selbst importiert werden kann, müssen noch die Mappings für den Cube Import angelegt werden. Hierfür muss ein Dimension-Mapping erstellt werden, um zu spezifizieren, dass im lokalen Cube nicht die Produkt Dimension, sondern die Dimension *Musik* verwendet wird. Außerdem ist ein Measure-Mapping für die Kennzahl *revenue* erforderlich, da die Import Methode sonst eine zusätzliche Kennzahl Umsatz erkennt und die Kennzahl zum globalen Schema hinzufügen würde. Wichtig dabei ist, dass alle Dimension-Mappings direkt dem Cube hinzugefügt werden, Measure-Mappings jedoch jener M-Relationship unter der der neue Teil-Cube eingefügt wird:

```
Declare
    global_dim_ref REF dimension_ty;
    local_dim_ref REF dimension_ty;
    global_mcube mcube_c000000001_ty;
    local_mcube_ref REF mcube_ty;
    global_root mrel_c000000001_ty;

Begin
    -- Add mappings
    SELECT TREAT(Value(mc) AS mrel_c000000001_ty) INTO global_root
    FROM   global_cube mc WHERE mc.coordinate.equals(
        coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

    SELECT TREAT(Value(mc) AS mcube_c000000001_ty) INTO global_mcube
    FROM   mcubes mc WHERE mc.cname = 'sales_cube';

    SELECT REF(d) INTO global_dim_ref FROM dimensions d WHERE dname =
        'product_dim';
    SELECT REF(d) INTO local_dim_ref FROM dimensions d WHERE dname =
        'musik_dim';
```

```
    SELECT Ref(mc) INTO local_mcube_ref FROM mcubes mc WHERE cname =
        'local_cube';

    --add the dimension mapping (product_dim – musik_dim)
    global_mcube.add_dimension_mapping(global_dim_ref, local_dim_ref,
        local_mcube_ref);

    -- add the measure mapping for the measure revenue
    global_root.add_measure_mapping('revenue', 'Umsatz',
        local_mcube_ref);
End;
```

Wenn alle Mappings angelegt wurden, kann der M-Cube selbst mit Hilfe der *import_mcube* Methode importiert werden. Die *import_mcube* Methode liegt im Package *mcube*. Sie benötigt die Parent M-Relationship im globalen Cube und die Wurzel des lokalen Teil-Cubes, welcher importiert werden soll, als Parameter. Des Weiteren wird hier ein Alias die verkaufte Menge mitgegeben, um die Kennzahl im globalen Schema unter der Bezeichnung *qtySold* einzufügen. Ein Measure-Alias ist für Fakten und Kennzahlen erforderlich, die im globalen Schema nicht existieren und für die deshalb kein Mapping erstellt werden kann.

```
Declare
    local_root_ref REF mrel_ty;
    global_root_ref REF mrel_ty;

Begin
    -- Import local M-Cube
    SELECT REF(mc) INTO global_root_ref FROM global_cube mc
    WHERE  mc.coordinate.equals(
        coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

    SELECT REF(mc) INTO local_root_ref FROM local_cube mc
    WHERE  mc.coordinate.equals(
        coordinate_c000000002_ty('Time', 'Location', 'Musik')) = 1;

    mcube.import_mcube(global_root_ref, local_root_ref,
        aliases_tty(aliases_ty('qtySold','verkaufteMenge')));
End;
```

4.7. Dimensionsimport - Sonderfälle

In den vorherigen Kapiteln wurden die einzelnen Mappings vorgestellt und erklärt, wie man die Dimensionen und den Cube eines lokalen Data Marts importieren kann. Jedoch wurde bisher immer nur der Standardfall betrachtet. In diesen Abschnitt wird nun auf ein paar Spezialfälle beim Import von Dimensionen eingegangen, die von der Implementierung ebenfalls behandelt werden können.

4.7.1. Zusätzliche Levels

Wenn der lokale Data Mart in einer Dimension mehr Levels hat als die globale, so erfordert dies eine spezielle Behandlung zur Behebung der Heterogenitäten. In der Folge wird als Beispiel die Integration einer lokalen Ortsdimension *Österreich* in eine globale Dimension Ort betrachtet. Abgesehen davon, dass die Bezeichnungen für Levels und Attribute der globalen Dimension in Englisch und jene der lokalen Dimension in Deutsch sind, besteht im verwendeten Beispiel der Unterschied, dass es lokal einen zusätzlichen Level *bundesland* gibt. Abbildung 16 stellt die beiden Ortsdimensionen gegenüber. Dabei ist die globale Dimension Ort links abgebildet, die lokale rechts. Die beidseitigen Pfeile zwischen den M-Objects *Austria* und *Österreich* bzw. *Salzburg* und *Sbg* weisen darauf hin, dass die Objekte äquivalent sind.

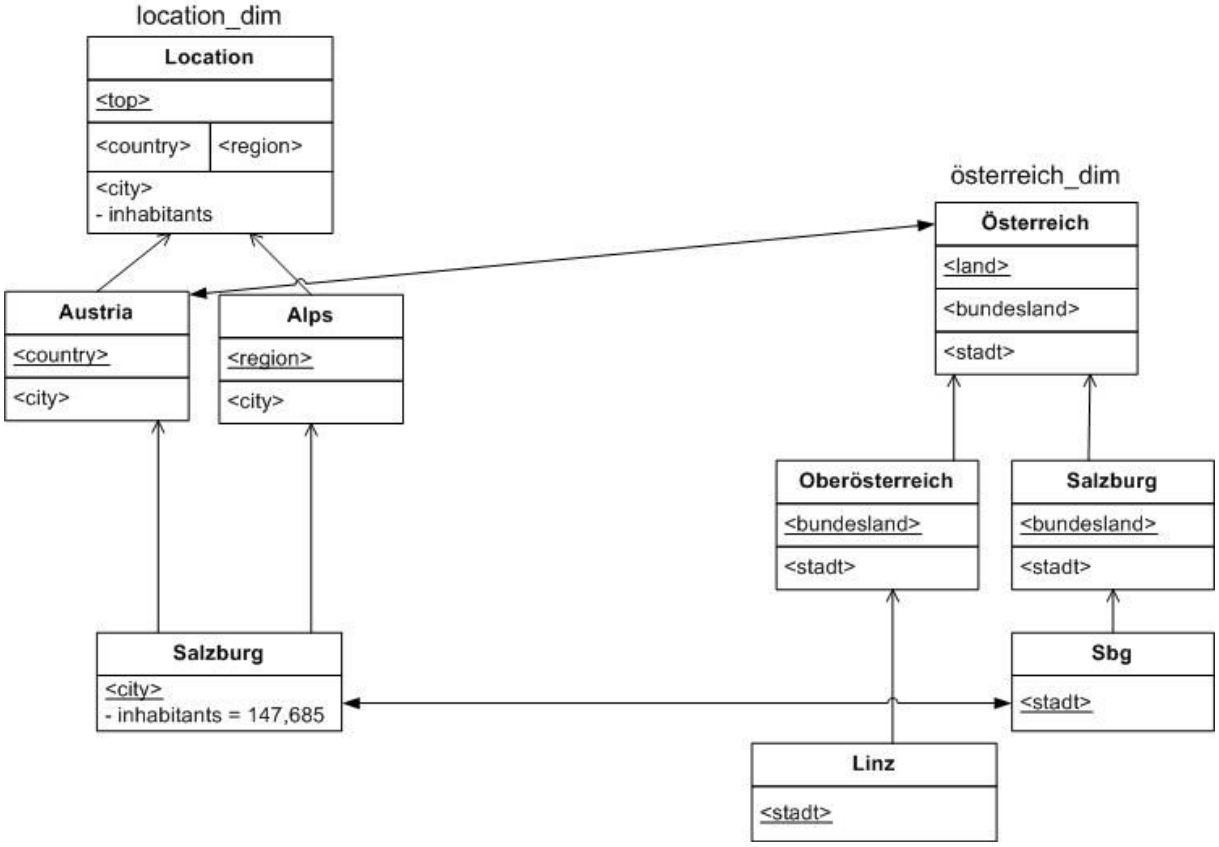

Abbildung 16: Ortsdimension - Ausgangspunkt

Für die unterschiedlichen Bezeichnungen der Levels *country* (*land*) und *city* (*stadt*) sind nun Level-Mappings zu erstellen, wie in Kapitel 4.3.1.1 beschrieben. Zusätzlich sind M-Object-Mappings erforderlich, um anzugeben, dass das Land *Austria* dem lokalen Land *Österreich* und *Salzburg* dem M-Object *Sbg* entspricht. Für alle zusätzlichen Levels des lokalen Schemas sind grundsätzliche keine Mappings erforderlich, die Import Methode erkennt diese

automatisch und passt das globale Schema an. Will man zusätzliche Levels, jedoch unter einem anderen Namen, ins globale Schema integrieren, so muss dieses unter Zuhilfenahme von Aliases erfolgen. Zum Beispiel wird hier ein Level Alias für das Level *bundesland* (*state*) vergeben. Sollen einzelne Objekte ebenfalls unter einem anderen Namen importiert werden, so müssen M-Object Aliase vergeben werden (zum Beispiel für das M-Object Oberösterreich):

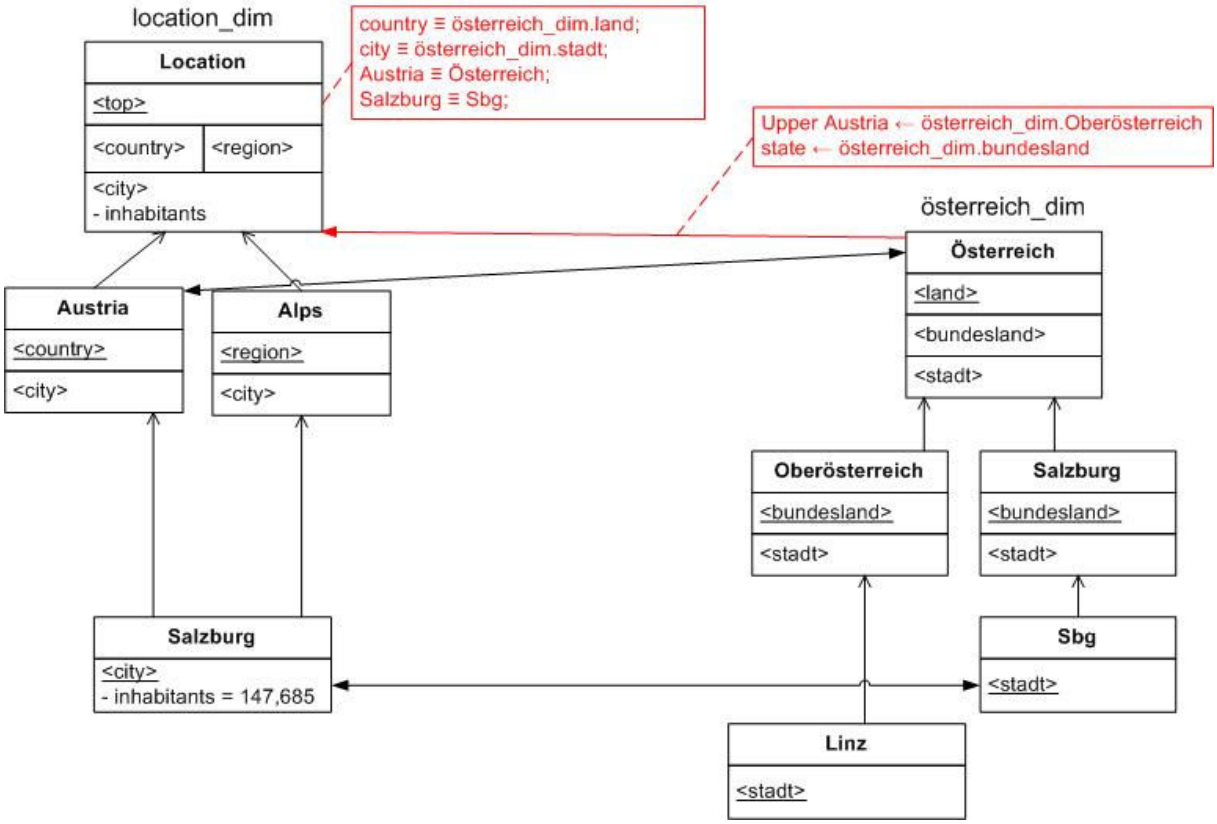

Abbildung 17: Ortsdimension - Mappings

Der unten abgebildete Codeausschnitt zeigt wie die lokale Dimension Ort importiert werden kann:

```
Declare
    obj_ref REF mobject_ty;
    obj mobject_ty;

Begin
    Select Value(o) Into obj From location_dim o Where o.oname =
         'Location';
    Select REF(o) Into obj_ref From austria_dim o Where o.oname =
         'Österreich';

    -- Add the level mappings
    obj.add_level_mapping('country', obj_ref, 'land');
    obj.add_level_mapping('city', obj_ref, 'stadt');

    -- add a Same-As Mapping of the local M-Object Österreich to the global
    -- M-Object Austria
```

```
    Select Value(o) Into obj From location_dim o Where o.oname = 'Austria';
    Select REF(o) Into obj_ref From Austria_dim o Where o.oname =
          'Österreich';
    obj.add_same_as_mapping(obj_ref, priority_tty());

    -- add the Same-As Mapping for Salzburg
    Select Value(o) Into obj From location_dim o Where o.oname =
          'Salzburg';
    Select REF(o) Into obj_ref From Austria_dim o Where o.oname =
          'Salzburg';
    obj.add_same_as_mapping(obj_ref, priority_tty());

    -- get the global and the local root M-Object
    Select Ref(o) Into global_obj_ref From product_dim o Where o.oname =
          'Product';
    Select Ref(o) Into local_obj_ref From musik_dim o Where o.oname =
          'Musik';

    -- import the local dimension
    dimension.import_branch(global_obj_ref, local_obj_ref,
          aliases_tty(aliases_ty('Upper Austria','Oberösterreich')),
          aliases_tty(aliases_ty('state','bundesland')), NULL);
End;
```

Beim Import einer Dimension mit zusätzlichen Levels wird das globale Schema dann automatisch angepasst und in der neuen Teildimension der neue Level eingefügt:

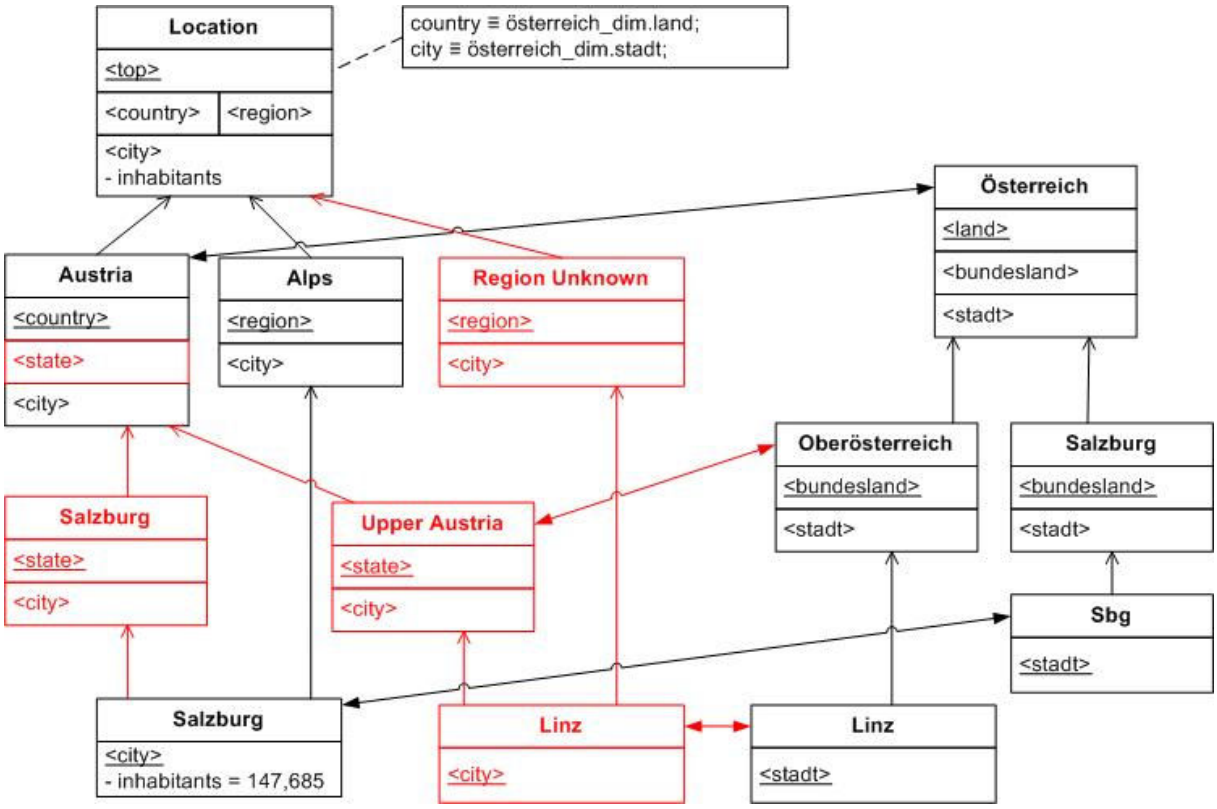

Abbildung 18: Importierte Ortsdimension

Weiters ist zu beachten, dass immer alle Elternobjekte gesetzt werden müssen. Wird ein Parent-M-Object nicht gesetzt, wie es zum Beispiel beim vorherigen Beispiel beim Objekt

Linz wäre, das lokal keine Region als Parent hat, wird automatisch ein ‘Unknown‘ Objekt in die Dimension eingefügt, welches dann als Parent fungiert (vgl. Abbildung 18).
Es ist für die Mappings unerheblich, ob der lokale Data Mart ein zusätzliches Level hat oder gleich mehrere. Hätte die lokale Ortsdimension zum Beispiel zwischen dem Bundesland und der Stadt auch noch einen zusätzlichen Level Bezirk, wären ebenfalls keine zusätzlichen Mappings erforderlich. Lediglich der Import Methode könnte ein zusätzliches Level-Alias mitgegeben werden, um den Level Bezirk gegebenfalls unter einem anderen Namen zu speichern. Die Anpassung der Level-Hierarchie im globalen Schema würde wiederum automatisch erfolgen.

4.7.2. Fehlende Levels

Wie beim Fall zusätzlicher Levels im lokalen Data Mart sind auch bei fehlenden Levels keine speziellen Mappings erforderlich. Es sind lediglich die Standard Mappings für die Levels *country* (*land*) und *city* (*stadt*) zu setzen (vgl. Abbildung 19), da diese im globalen Schema und im lokalen Schema unterschiedlich bezeichnet wurden. Für den lokal fehlenden Level *state* ist kein Mapping erforderlich. Die Level-Hierarchie der zu importierenden Objekte wird automatisch angepasst und für Elternobjekte, die nicht bekannt sind, werden automatisch ‘Unknown‘ Objekte als Parent gesetzt. Dies ist zum Beispiel bei der Stadt *St.Johann* der Fall, weil nicht bekannt ist, zu welchem Bundesland beziehungsweise zu welcher Region sie gehört (siehe Abbildung 20).

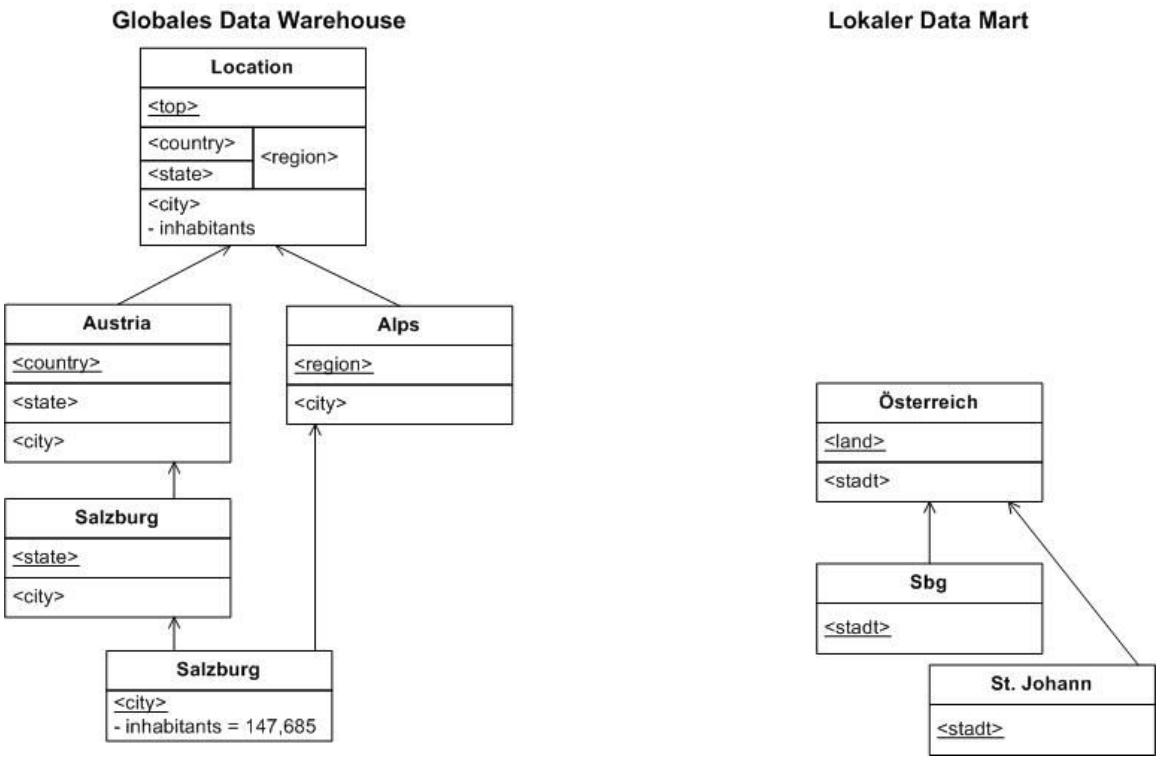

Abbildung 19: Ortsdimension mit fehlenden Levels

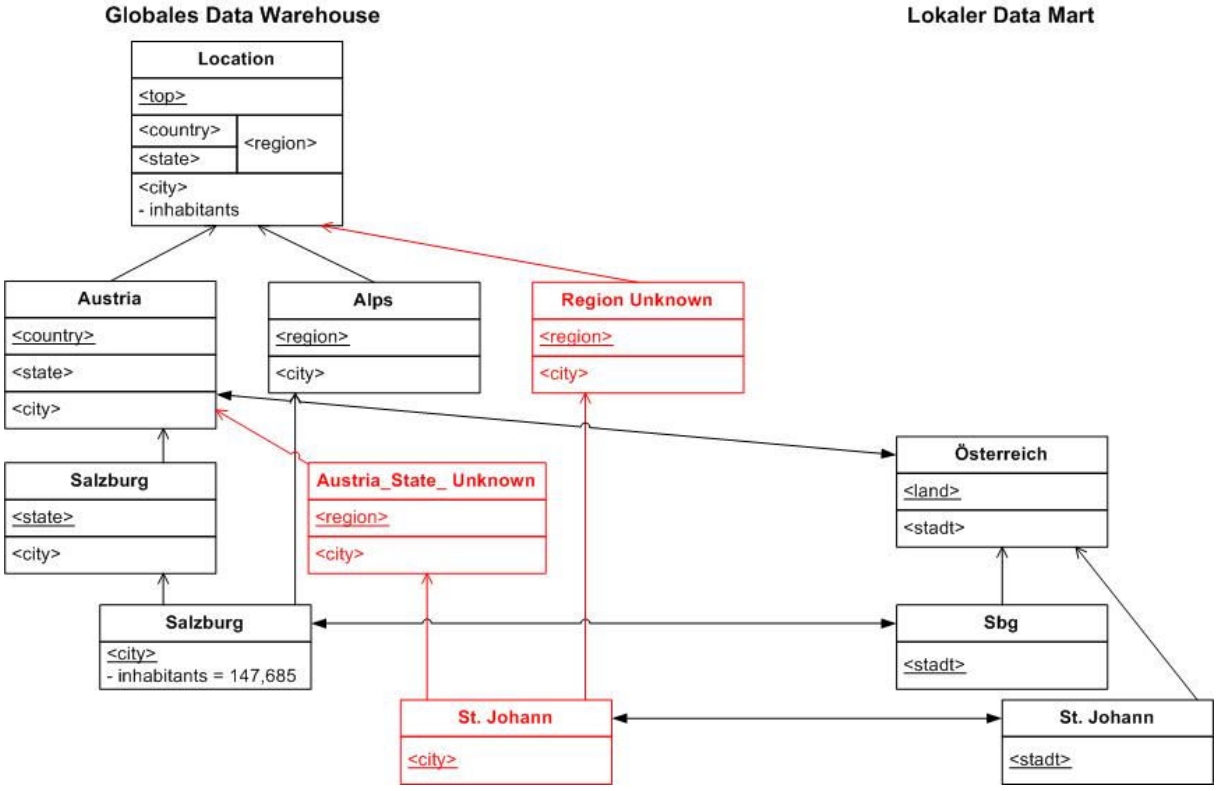

Abbildung 20: Importierte Ortsdimension mit fehlenden Levels

Auch hier ist es egal, wie viele Levels im lokalen Schema fehlen. Gäbe es zum Beispiel im globalen Schema zwischen dem Bundesland Salzburg und der Stadt Salzburg auch noch einen

Bezirk, so hätte dies keinen Effekt auf die zu setzenden Mappings. Die Import Methode würde in diesem Fall automatisch für St. Johann ein Objekt *Austria_county_unknown* erstellen und dieses als Elternobjekt setzen. *Austria_county_unknown* würde wiederum *Austria_state_unknown* untergeordnet werden, und dieses Objekt wiederum dem Land Österreich.

4.7.3. Zusätzliche Attribute

Hat der lokale Data Mart zusätzliche Attribute in einer Dimension, sind dafür ebenfalls keine Mappings zwingend erforderlich. Will man jedoch ein zusätzliches Attribut unter einem anderen Namen im globalen Schema einfügen, ist dafür ein Alias für das Attribut erforderlich, welches der *import_branch* Methode mitgegeben wird. Für zusätzliche Attribute kann kein Mapping definiert werden, da das Attribut, auf welches sich das Mapping bezieht, nicht in der globalen Dimension existiert. Hätte nun der Level Stadt im Beispiel aus Kapitel 4.7.2 ein Attribut Fläche, so würde folgender Aufruf der *import_branch* Methode genügen, um die lokale Dimension zu importieren:

```
dimension.import_dimension(location_obj_ref, austria_obj_ref, NULL, NULL,
      NULL);
```

Soll das Attribut *Fläche* global als *area* eingefügt werden, muss ein Alias für das Attribut verwendet werden:

```
dimension.import_dimension(location_obj_ref, austria_obj_ref, NULL, NULL,
      aliases_tty(aliases_ty('area', 'Fläche')));
```

4.7.4. Äquivalente Objekte

In diesem Kapitel wird aufgezeigt, wie äquivalente Objekte behandelt werden müssen. Äquivalente Objekte sind M-Objects, die es sowohl im globalen Data Warehouse als auch im lokalen Data Mart gibt. Sie müssen speziell behandelt werden, da die Attribute von äquivalenten Objekten widersprüchliche Werte haben können. Um auf widersprüchliche Werte reagieren zu können, wird es dem Benutzer ermöglicht, jedem lokalen Attributwert eine Priorität zuzuweisen.

Die Problematik von äquivalenten Objekten soll anhand des Imports einer Dimension Ort mit einem zusätzlichen Level Bundesland detailliert erklärt werden. Dabei gibt es in beiden Ortsdimensionen die Städte Salzburg und Innsbruck. Jede Stadt hat ein Attribut Einwohner,

die Werte dafür weichen jedoch voneinander ab. Weiters haben die einzelnen Städte im globalen Schema ein zusätzliches Elternobjekt Region.

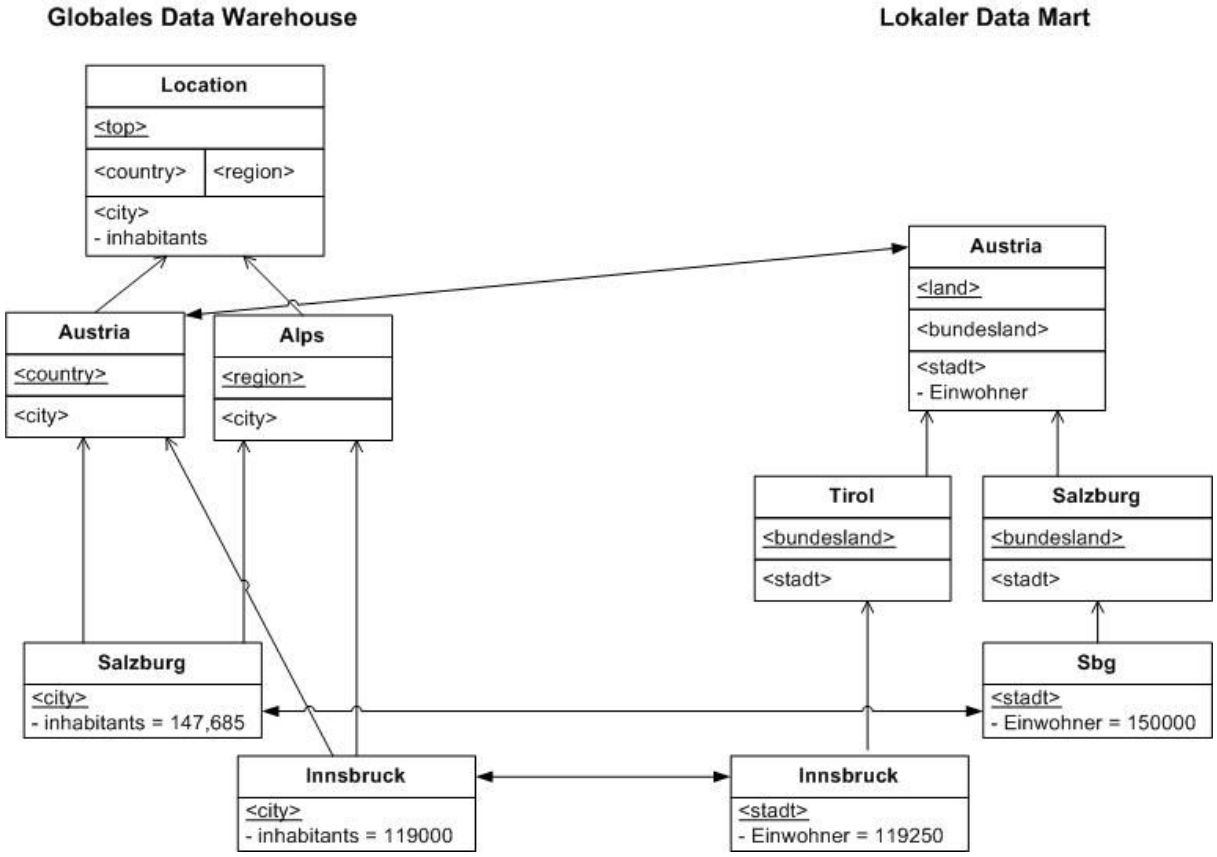

Abbildung 21: M-Object Mapping Beispiel - Ausgangspunkt

Im ersten Schritt müssen zunächst wie in allen bisherigen Beispielen die Mappings definiert werden. Hierfür werden zu Beginn die Level- und Attribut-Mappings für die Levels Land (*country – land*) bzw. Stadt (*city - stadt*) und für das Attribut Einwohner (*inhabitants – Einwohner*) angelegt. Anschließend müssen für die Objekte *Austria*, *Sbg* und *Innsbruck* M-Object-Mappings erstellt werden, da die Import Methode ansonsten annimmt, dass es sich um neue Objekte handelt. Wird kein M-Object-Mapping definiert, dann erzeugt die Import Methode einen neuen, automatisch vergebenen Namen für die Objekte und fügt sie unter dem neuen Namen ins globale Schema ein. Vergisst man also zum Beispiel das M-Object-Mapping für die Stadt Innsbruck, so wird angenommen, dass es sich um eine weitere Stadt mit den Namen Innsbruck handelt. Daraufhin würde ein neuer Name erzeugt, z.B. Innsbruck#2, und das Objekt unter diesem neuen Namen ins globale Schema eingefügt werden.

Des Weiteren können bei den M-Object-Mappings für alle Attribute Prioritäten vergeben werden. Da die Einwohnerzahlen im Data Mart von jenen im Data Warehouse abweichen,

kann der Benutzer angeben, welche Werte genommen werden sollen. Wenn explizit keine Priorität angegeben wird, wird standardmäßig der Attributwert des globalen Schemas beibehalten. Wenn nun aber der lokale Wert für das Attribut genommen werden soll, muss beim M-Object-Mapping die Priorität „1“ für das Attribut gesetzt werden. Priorität „0“ würde hingegen bedeuten, dass der globale Wert beibehalten werden soll:

```
global_obj.add_same_as_mapping(local_obj_ref,
      priority_tty(priority_ty(attribute1, 1),
                   priority_ty(attribute2, 0)));
```

Der folgende Code Auszug zeigt die Mappings, die für das oben abgebildete Beispiel erforderlich sind. Dabei wird angegeben, dass die Einwohnerzahl der Stadt Innsbruck mit dem Wert der lokalen Dimension überschrieben und jene der Stadt Salzburg beibehalten werden soll:

```
Select Value(o) Into obj From location_dim o Where oname = 'Location';
Select REF(o) Into obj_ref From austria_dim o Where oname = 'Austria';

-- add the level mappings
obj.add_level_mapping('country', obj_ref, 'land');
obj.add_level_mapping('city', obj_ref, 'stadt');

-- add the attribute mapping
obj.add_attribute_mapping('city', 'inhabitants', obj_ref, 'stadt',
      'einwohner');

-- add the Same-As Mappings for object Austria
Select Value(o) Into obj From location_dim o Where oname = 'Austria';
obj.add_same_as_mapping(obj_ref, priority_tty());

-- add the Same-As Mappings for object Salzburg
Select Value(o) Into obj From location_dim o Where oname = 'Salzburg';
Select Ref(o) Into obj_ref From austria_dim o Where oname = 'Sbg';
obj.add_same_as_mapping(obj_ref, priority_tty());

-- add the Same-As Mappings for object Innsbruck
Select Value(o) Into obj From location_dim o Where oname = 'Innsbruck';
Select Ref(o) Into obj_ref From austria_dim o Where oname = 'Innsbruck';
obj.add_same_as_mapping(obj_ref,
      priority_tty(priority_ty('inhabitants',1)));
```

Nachdem die Mappings gesetzt wurden, kann die Dimension importiert werden. Hierfür sind wiederum die Wurzel des Teilbaumes des lokalen Schemas und das Objekt im globalen Schema, unter dem die neue Dimension eingefügt werden soll, erforderlich. Zusätzlich wird hier der Import Methode ein Alias für den Level mitgegeben damit das Bundesland im globalen Data Warehouse als *state* eingefügt wird:

```
Select Ref(o) Into global_obj_ref From location_dim o Where o.oname =
      'Location';
Select Ref(o) Into local_obj_ref From austria_dim o Where o.oname =
      'Austria';
```

```
-- import the local dimension
dimension.import_branch(global_obj_ref, local_obj_ref, NULL,
      aliases_tty(aliases_ty('state','bundesland')), NULL);
```

Abbildung 22 zeigt die fertig importierte Dimension. Die Level-Hierarchie des globalen Data Warehouses wurde für das Objekt Austria und alle Nachfolger angepasst, die Elternobjekte für alle Städte aktualisiert und für das M-Object Innsbruck der lokale Attributwert übernommen:

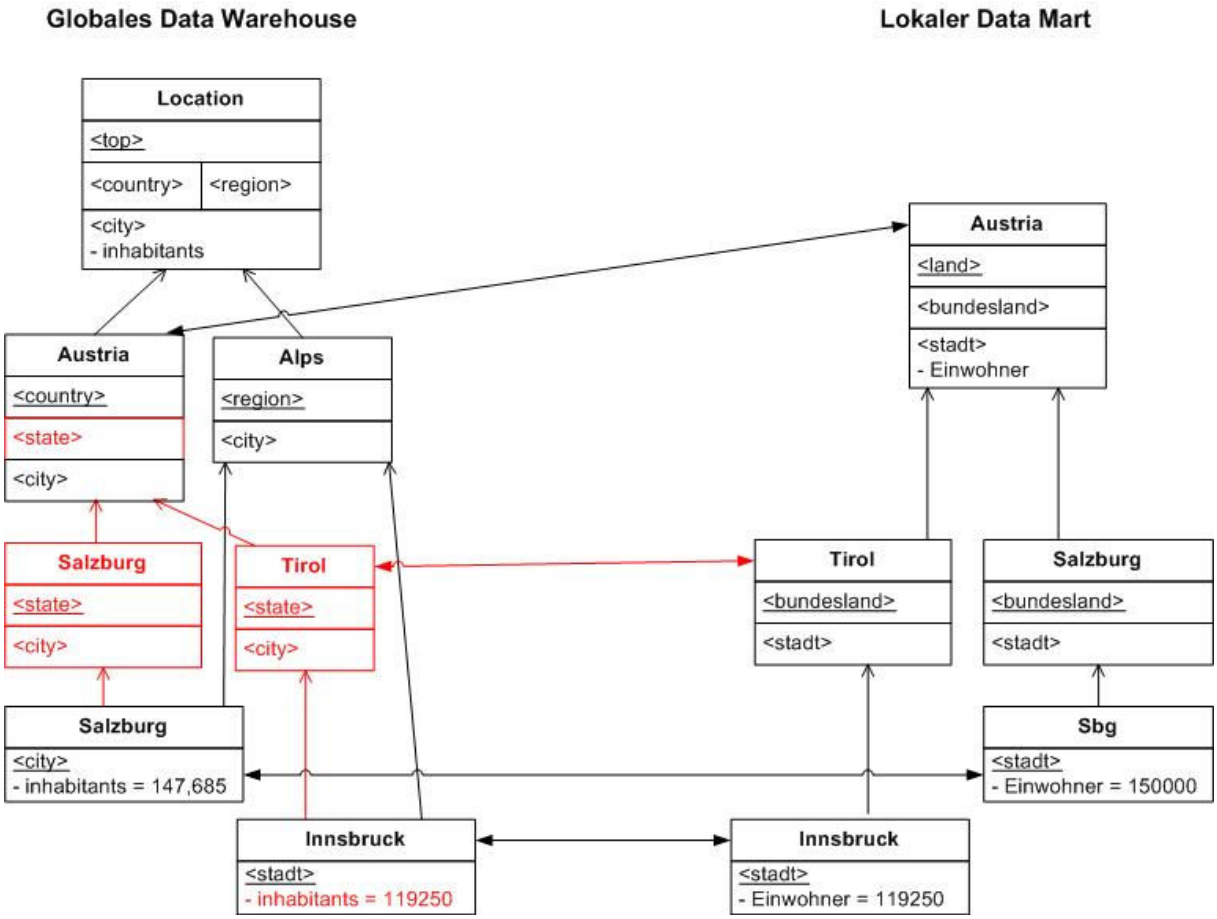

Abbildung 22: M-Object-Mapping Beispiel - Importierte Dimension

4.8. M-Cube-Import - Sonderfälle

In diesem Kapitel werden die Sonderfälle beim Import eines Cubes näher betrachtet und aufgezeigt, wie sich diese auf die Definition der Mappings beziehungsweise auf den Importvorgang selbst auswirken.

4.8.1. Zusätzliche Dimensionen

In diesem Abschnitt werden die Auswirkungen auf die Mappings und den Importvorgang näher beschrieben, wenn der lokale Data Mart zusätzliche Dimensionen hat. Der Import wird anhand des Basis Beispiels aus Kapitel 4.1 erklärt, jedoch hat der Data Mart dieses Mal eine zusätzliche Dimension *Promotion*, welche die Werbemaßnahme (keine Werbung, TV-Werbung, Messe etc.) bei den Verkaufsdaten zusätzlich abspeichert.

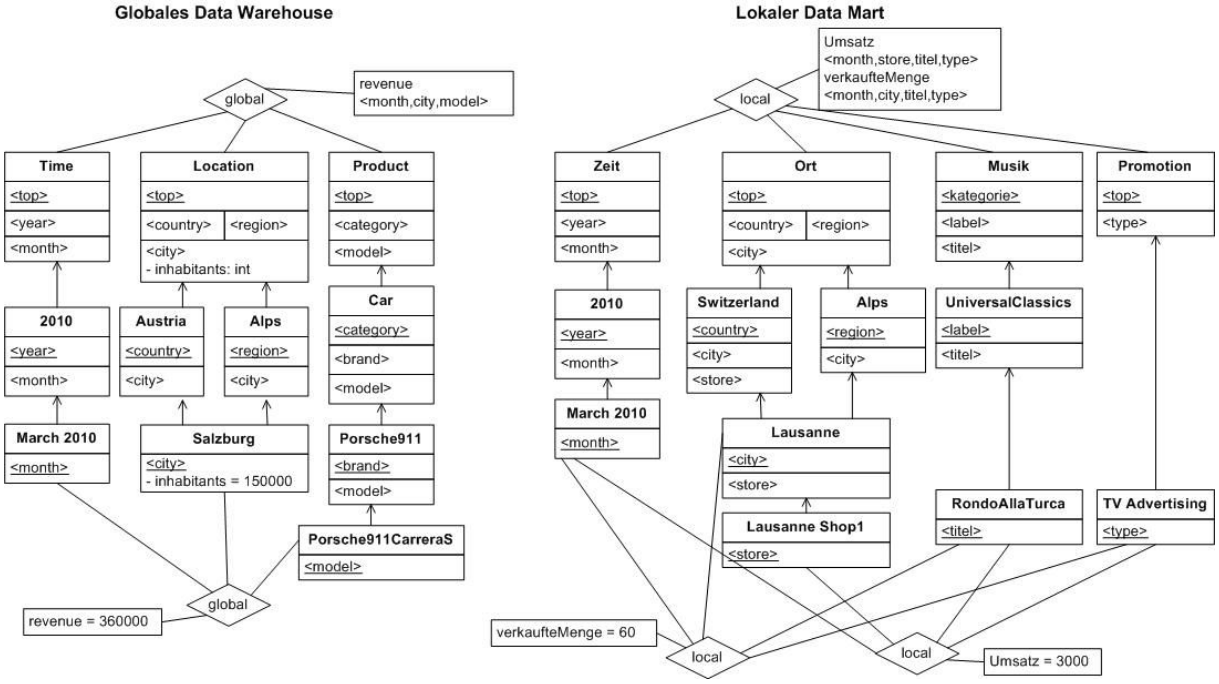

Abbildung 23: M-Cube mit mehr Dimensionen

Wenn ein Data Mart zusätzliche Dimensionen hat, sind dafür keine zusätzlichen Mappings erforderlich. Es müssen lediglich die standardmäßigen Mappings aus Kapitel 4.3.1 angelegt werden.

Der folgende Codeausschnitt zeigt wie die Level-Mappings für die Levels *category* (*kategorie*) und *model* (*titel*), die Attribut-Mappings für die Attribute *catMgr* (*Verantwortlicher*), *taxRate* (*Steuersatz*) und *costs* (*Listenpreis*), das Measure-Mapping für die Kennzahl *Umsatz* und das Dimension-Mapping für die Dimension *Musik* erzeugt werden müssen. Die Tabelle *d000000001* enthält dabei die Produktdimension, die Tabelle *d000000004* die Musikdimension und die Tabelle *c000000001* den globalen Cube:

```
Select Value(o) Into obj From d000000001 o Where o.oname = 'Product';
Select REF(o) Into obj_ref From d000000004 o Where o.oname = 'Musik';

obj.add_level_mapping('category', obj_ref, 'kat');
obj.add_level_mapping('model', obj_ref, 'titel');
```

```
obj.add_attribute_mapping('category', 'catMgr', obj_ref, 'kat',
      'verantwortlicher');
obj.add_attribute_mapping('category', 'taxRate', obj_ref, 'kat',
      'steuersatz');
obj.add_attribute_mapping('model', 'costs', obj_ref, 'kat',
      'listenpreis');

SELECT TREAT(Value(mc) AS mrel_c000000001_ty) INTO global_root
FROM   c000000001 mc
WHERE mc.coordinate.equals(
      coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

SELECT TREAT(Value(mc) AS mcube_c000000001_ty) INTO global_mcube
FROM   mcubes mc WHERE mc.cname = 'sales_cube';

SELECT REF(d) INTO global_dim_ref
FROM   dimensions d
WHERE  d.dname = 'product_dim';

SELECT REF(d) INTO local_dim_ref
FROM   dimensions d
WHERE  d.dname = 'musik_dim';

SELECT Ref(mc) INTO local_mcube_ref
FROM   mcubes mc
WHERE  mc.cname = 'local_cube';

global_mcube.add_dimension_mapping(dimension_mapping_ty(global_dim_ref,
      local_dim_ref, local_mcube_ref));
global_root.add_measure_mapping('revenue', 'Umsatz', local_mcube_ref);
```

Auch auf den zweiten und den letzten Schritt, den Import der Dimensionen und des Cubes haben zusätzliche Dimensionen keine Auswirkung für den Benutzer.

Allerdings gibt es sehr wohl Auswirkungen auf den Importvorgang selbst. Hat ein lokaler M-Cube eine zusätzliche Dimension, müssen die einzelnen Koordinaten der M-Relationships beim Import an das globale Schema angepasst werden, da die Koordinaten des lokalen Schemas gegenüber dem Globalen zusätzliche M-Objects beinhalten.

Weiters haben zusätzliche Dimensionen Auswirkungen auf die Werte der Kennzahlen selbst. Beispielsweise sind die Werte der Kennzahl Umsatz lokal pro Monat, Filiale, Produkt und Werbemaßname, im globalen Schema jedoch nur pro Monat, Stadt und Produkt. Die Kennzahlen müssen also beim Import durch ein roll-up auf den entsprechenden globalen Connection Level <month,city,model> aggregiert werden. Dies regelt jedoch der Importprozess selbst, der Benutzer muss dabei nichts beachten.

4.8.2. Fehlende Dimensionen

Wenn der Cube des lokalen Data Marts weniger Dimensionen hat als das globale Data Warehouse, hat dies für den Benutzer ebenso keine Auswirkung. Auch in diesem Fall werden die Mappings für die Levels *category* (*kategorie*) und *model* (*titel*), die Attribute *catMgr* (*Verantwortlicher*), *taxRate* (*Steuersatz*) und *costs* (*Listenpreis*), die Kennzahl *Umsatz* und die Dimension *Musik* ganz normal angelegt und anschließend die Dimensionen und der Cube importiert. Die Koordinaten der M-Relationships und die Connection Level der Kennzahlen werden wiederum automatisch angepasst. Die Koordinate der M-Relationship LausanneShop1 – RondoAllaTurca wird somit selbstständig auf Time – Lausanne Shop 1 – RondoAllaTurca angepasst. Für die fehlende Dimension wird stets das Wurzel Objekt der jeweiligen Dimension genommen, bei dem Connection Level wird der Top-Level der fehlenden Dimension ergänzt. Die Werte der Kennzahlen bleiben unverändert, lediglich die zugehörige Koordinate wird an das globale Schema angepasst.

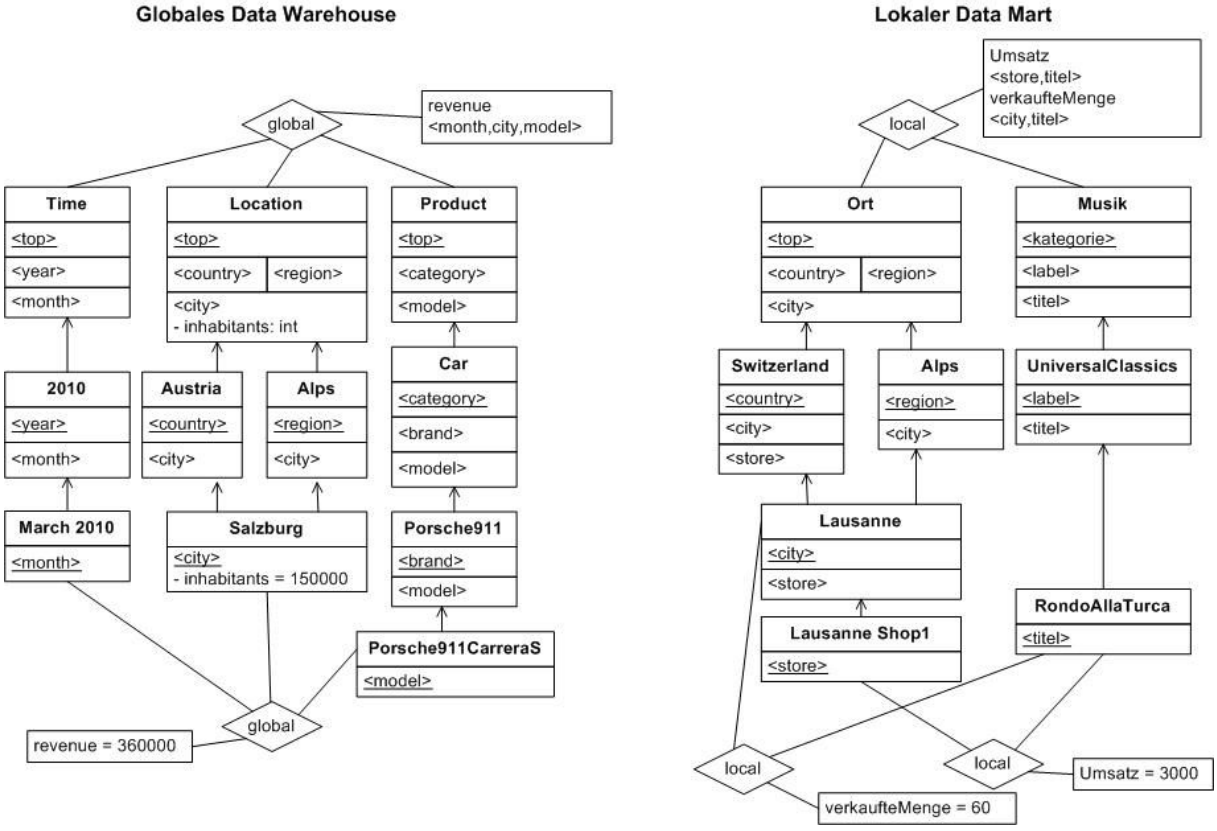

Abbildung 24: Data Mart mit weniger Dimensionen

Das oben beschriebene Verhalten gilt für den Standardfall, es gibt jedoch auch die Möglichkeit, ein Hidden-Dimension-Mapping für fehlende Dimensionen zu erstellen. Wenn beispielsweise bekannt ist, dass alle Daten des Data Marts auf das Jahr 2010 bezogen sind, so

kann durch ein Hidden-Dimension-Mapping angeben werden, dass für die fehlende Dimension Zeit das M-Object Jahr 2010 genommen werden soll. Dies hat zur Folge, dass für alle Koordinaten das Jahr 2010 anstatt der Wurzel der Zeitdimension genommen wird, und im Connection Level bei den Attributen der Level des M-Objects Jahr 2010 und nicht das Top-Level der Dimension verwendet wird.
Der folgende Code zeigt die erforderlichen Mappings für das Beispiel aus Abbildung 24, wenn alle Verkäufe auf das Jahr 2010 bezogen sind:

```
Select Value(o) Into obj From d000000001 o Where o.oname = 'Product';
Select REF(o) Into obj_ref From d000000004 o Where o.oname = 'Musik';

-- add level mappings
obj.add_level_mapping('category', obj_ref, 'kat');
obj.add_level_mapping('model', obj_ref, 'titel');

-- add attribute mappings
obj.add_attribute_mapping('category', 'catMgr', obj_ref, 'kat',
      'verantwortlicher');
obj.add_attribute_mapping('category', 'taxRate', obj_ref, 'kat',
      'steuersatz');
obj.add_attribute_mapping('model', 'costs', obj_ref, 'kat', 'listenpreis');

SELECT TREAT(Value(mc) AS mcube_c000000001_ty) INTO global_mcube
FROM   mcubes mc WHERE mc.cname = 'sales_cube';

SELECT TREAT(Value(mc) AS mrel_c000000001_ty) INTO global_root
FROM   global_cube mc WHERE mc.coordinate.equals(
    coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

SELECT REF(d) INTO global_dim_ref FROM dimensions d WHERE dname =
      'product_dim';
SELECT REF(d) INTO local_dim_ref FROM dimensions d WHERE dname =
      'musik_dim';
SELECT REF(mc) INTO local_mcube_ref FROM mcubes mc WHERE cname =
      'local_cube';
SELECT REF(o) INTO obj_ref FROM d000000002 o WHERE o.oname = 'Year2010';

-- add a dimension mapping for the music dimension
global_mcube.add_dimension_mapping(global_dim_ref, local_dim_ref,
      local_mcube_ref);

-- add a hidden dimension mapping for the dimension time
global_mcube.add_hidden_dim_mapping(local_mcube_ref, 'time_dim', obj_ref);

-- add a measure mapping for the measure 'Umsatz'
global_root.add_measure_mapping('revenue', 'Umsatz', local_mcube_ref);
```

Der Import der Dimensionen und des M-Cubes erfolgt standardmäßig, wie in den Kapiteln 4.4 bzw. 4.5 beschrieben:

```
Select Ref(o) Into global_obj_ref From d000000001 o Where oname =
      'Product';
Select Ref(o) Into local_obj_ref From d000000004 o Where oname = 'Musik';

-- import the musik dimension
```

```
dimension.import_branch(global_obj_ref, local_obj_ref,
      aliases_tty(aliases_ty('Music','Musik')),
      aliases_tty(aliases_ty('music_label','label')),
      aliases_tty(aliases_ty('composer','Komponist')));

SELECT REF(mc) INTO global_root_ref
FROM c000000001 mc
WHERE  mc.id = 'r000000001';

SELECT REF(mc) INTO local_root_ref
FROM   c000000002 mc
WHERE  mc.id = 'r000000001';

-- Import local M-Cube
mcube.import_mcube(global_root_ref, local_root_ref,
      aliases_tty(aliases_ty('qtySold','verkaufteMenge')));
```

4.8.3. Zusätzliche Measures

Mit zusätzlichen Measures (Kennzahlen) verhält es sich wie mit zusätzlichen Attributen beim Dimensionsimport. Hat der lokale Data Mart eine zusätzliche Kennzahl, wie zum Beispiel die verkaufte Menge, so sind hierfür grundsätzlich keine zusätzlichen Mappings erforderlich. Will man die Kennzahl jedoch unter einem anderen Namen ins globale Data Warehouse integrieren, so ist ein Alias zu erstellen und der *import_mcube* Methode mitzugeben. Das folgende Beispiel zeigt den entsprechenden Aufruf der Import M-Cube Methode, wenn die Kennzahl *verkaufteMenge* im globalen Data Warehouse als *qtySold* bezeichnet werden soll:

```
mcube.import_mcube(global_root_ref, local_root_ref,
      aliases_tty(aliases_ty('qtySold','verkaufteMenge')));
```

4.8.4. Unterschiedliche Granularität der Kennzahlen

Bei der Integration von Daten kann es passieren, dass eine Kennzahl im lokalen Data Mart eine andere Granularität hat als im globalen Data Warehouse. Dies ist zum Beispiel in der Problemstellung bei der Kennzahl *revenue* der Fall. Global ist diese Kennzahl für den Connection Level <*month,city,model*> definiert, lokal die Kennzahl Umsatz jedoch für den Connection Level <*month,store,model*>.

Gibt es eine Kennzahl sowohl global als auch lokal, wird in folgende vier Fälle unterschieden:

- Gleiche Granularität der Kennzahlen:
  Ist eine Kennzahl des lokalen Schemas bereits global vorhanden und der Connection Level der beiden Kennzahlen gleich, so muss sie im globalen Schema nicht

hinzugefügt werden. Die lokalen Werte der Kennzahl können somit auch global einfach gesetzt werden.

- Die lokale Kennzahl ist detaillierter:
  Wenn die lokale Kennzahl auf einem niedriger gelegenen Connection Level definiert ist als die globale, wie das in der Problemstellung der Fall ist, (<*month,store,model*> ist detaillierter als <*month,city,model*>, weil der Level *store* als Kindknoten unter dem Level *city* liegt), so wird die Kennzahl im globalen Schema auf den detaillierteren Connection Level angehoben und die Werte hierfür aus dem lokalen Schema importiert. Die vorherige globale Kennzahl auf dem Connection Level <*month,city,model*> mit allen zugehörigen Werten bleiben allerdings zusätzlich bestehen.
- Die lokale Kennzahl ist allgemeiner:
  Liegt die lokale Kennzahl auf einem höher gelegenen Connection Level, wird sie nicht im globalen Data Warehouse hinzugefügt und die zugehörigen Werte nicht importiert. Hätte die lokale Kennzahl *Umsatz* etwa den Connection Level <*year,city,model*>, würde sie nicht importiert werden, da der Level *year* über dem Level *month* liegt und es sich somit um allgemeinere, stärker aggregierte Daten handelt.
- Teilweise detailliertere, teilweise allgemeinere Granularität:
  Haben die lokale und die globale Kennzahl eine gemischte Granularität, so wird die lokale Kennzahl ebenfalls nicht weiter betrachtet und alle zugehörigen Werte nicht importiert. Dies wäre beispielsweise der Fall, wenn sich der Umsatz global auf den Connection Level <*month,city,model*> und lokal auf <*year,store,model*> beziehen würde. Damit wäre er im Bezug auf die Zeitdimension allgemeiner als im globalen Schema, im Bezug auf die Ortsdimension hingegen detaillierter.

4.8.5. Äquivalente M-Relationships

Wie bereits beschrieben können bei der Integration von Data Marts beziehungsweise ganz allgemein von Daten Probleme aufgrund von ähnlichen und teils widersprüchlichen Daten auftreten. So können nicht nur Attribute beim Dimensionsimport, sondern auch Kennzahlen beim Cube-Import unterschiedliche Werte aufweisen. Dieses Kapitel zeigt auf, wie diese behandelt werden.

Um die Problematik besser zu veranschaulichen, wird sie anhand eines konkreten Beispiels beschrieben. Dabei wird wieder vom Cube der Problemstellung mit den Dimensionen Zeit, Ort und Produkt ausgegangen. Jedoch hat er dieses Mal neben dem Umsatz auch noch eine Kennzahl *qtySold*.

Der lokale Data Mart ist ebenfalls ein Cube mit Verkaufsdaten, welcher aber dieses Mal zur Vereinfachung dieselben Dimensionen hat, und ebenfalls zwei Kennzahlen für den Umsatz und für die verkaufte Menge. Die Werte der beiden Kennzahlen unterscheiden sich jedoch mit jenen im globalen Schema (vgl. Abbildung 25).

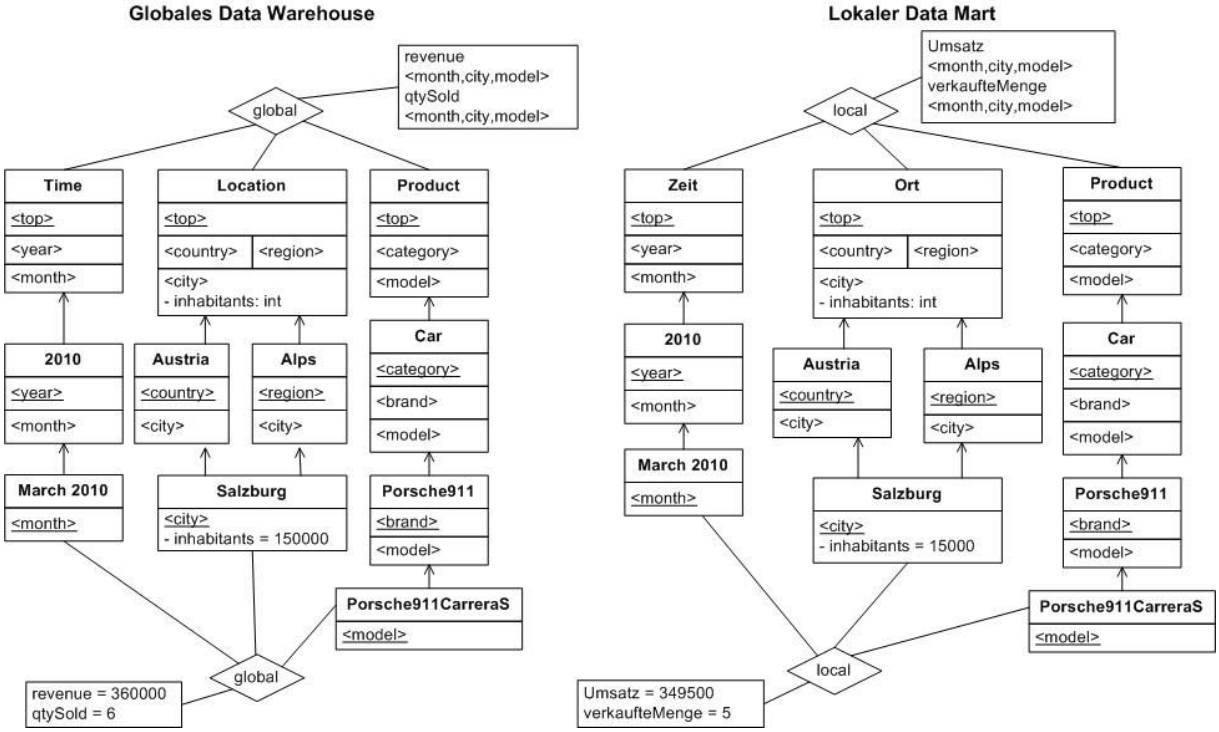

Abbildung 25: M-Cube mit äquivalenten M-Relationships

Da der lokale Data Mart dieselben Dimensionen wie das globale Data Warehouse verwendet, ist ein Dimensionsimport hier nicht erforderlich, es kann also gleich zur Definition der Mappings für den Cube Import übergegangen werden. Hierfür sind zuerst einmal zwei Measure-Mappings für die Kennzahlen Umsatz (*revenue – Umsatz*) und verkaufte Menge (*qtySold – verkaufteMenge*) erforderlich (vgl. Kapitel 4.3.1.4).

Anschließend ist ein M-Relationship-Mapping für Salzburg – März 2010 – Porsche 911 Carrera S erforderlich, da diese M-Relationship in beiden Cubes vorkommt. Bei einem standardmäßigen M-Relationship-Mapping (vgl. Kapitel 4.3.1.5) würden bei widersprüchlichen Werten der Kennzahlen jene des globalen Schemas beibehalten werden. Grundsätzlich wären M-Relationship-Mappings gar nicht erforderlich, da äquivalente M-

Relationships auch durch Vergleichen der M-Objects ihrer Koordinaten gefunden werden könnten.

Man beachte jedoch, dass in der derzeitigen Implementierung zum Importieren von Data Marts in ein globales Data Warehouse, dies aus Effizienzgründen nicht gemacht wird. Somit sind für M-Relationships, die sowohl im globalen als auch im lokalen Cube vorkommen, zwingend M-Relationship-Mappings erforderlich. Andernfalls wirft der Importvorgang einen Fehler und wird abgebrochen.

Ein einfaches standardmäßiges M-Relationship-Mapping würde folgendermaßen aussehen:

```
global_mrel.add_same_as_mapping(local_mrel_ref, mrel_priority_tty());
```

Will man jedoch selbst entscheiden, welche Werte vom globalen Cube und welche vom lokalen Cube übernommen werden sollen, kann man Prioritäten für die Kennzahlen vergeben. Dazu gibt man neben dem M-Relationship-Mapping den Namen der Kennzahl und eine Priorität an. Die Priorität selbst ist eine Zahl, welche die Werte „0“ oder „1“ annehmen kann. Der Wert „0“ würde festlegen, dass der lokale Wert der Kennzahl keine Priorität hat und somit jener des globalen Data Warehouses bleibt. Der Wert „1“ gibt an, dass der Wert des lokalen Data Marts übernommen werden soll. In unserem Beispiel soll der Umsatz aus dem lokalen Cube und die verkaufte Menge aus dem globalen Cube genommen werden. Dafür ist ein M-Relationship-Mapping für die M-Relationship (*März 2010*, *Salzburg*, *Porsche911*) mit der Priorität „1“ für den Umsatz bzw. der Priorität „0“ für die verkaufte Menge erforderlich. Zudem sind Measure-Mappings für die Kennzahlen *Umsatz* und *verkaufte Menge* zu erstellen. Die Mappings wären wie folgt anzulegen:

```
SELECT TREAT(Value(mc) AS mrel_c000000001_ty) INTO global_root
FROM   c000000001 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

SELECT value(mc) INTO global_mrel FROM c000000001 mc
WHERE  mc.coordinate.equals(coordinate_c000000001_ty('Month_3_2010',
      'Salzburg', 'Porsche911CarreraS')) = 1;

SELECT REF(mc) INTO local_mrel_ref FROM c000000002 mc
WHERE  mc.coordinate.equals(coordinate_c000000002_ty('Month_3_2010',
      'Salzburg', 'Porsche911CarreraS')) = 1;

SELECT Ref(mc) INTO local_mcube_ref FROM mcubes mc
WHERE  mc.cname = 'local_cube';

-- add the measure mappings
global_root.add_measure_mapping('qtySold', 'verkaufteMenge',
      local_mcube_ref);
global_root.add_measure_mapping('revenue', 'Umsatz', local_mcube_ref);

-- add the Same-As Mapping for the measures of Salzburg
global_mrel.add_same_as_mapping(local_mrel_ref, 'revenue', 1);
```

```
global_mrel.add_same_as_mapping(local_mrel_ref, 'qtySold', 0);
```

Alternativ könnte man, anstatt die Prioritäten einzeln zu einem Same-As Mapping hinzuzufügen, auch einfach ein Feld von Prioritäten mitgeben:

```
global_mrel.add_same_as_mapping(local_mrel_ref,
      mrel_priority_tty(mrel_priority_ty('qtySold', 0),
      mrel_priority_ty('revenue', 1)));
```

Nachdem die Mappings hinzugefügt wurden, kann der lokale Cube importiert werden. Dies passiert über die Standard *import_mcube* Methode (vgl. Kapitel 4.5):

```
SELECT REF(mc) INTO global_root_ref FROM c000000001 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000001_ty('Time', 'Location', 'Product')) = 1;

SELECT REF(mc) INTO local_root_ref FROM c000000002 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000002_ty('Time', 'Location', 'Product')) = 1;

-- Import of the local M-Cube
mcube.import_mcube(global_root_ref, local_root_ref, NULL);
```

4.8.6. Pivoting

Wie bereits in Kapitel 4.3.1.8 beschrieben, können bei der Integration von Data Marts Schema/Instanz Konflikte auftreten, die mit Pivoting oder Unpivoting zu beheben sind. Schema/Instanz Konflikte stellen beim Import von Cubes einen Sonderfall dar und müssen deshalb auch speziell behandelt werden. In der Folge wird die Problematik anhand des Beispiels in Abbildung 26 näher dargestellt, und anschließend mit Pivoting behandelt. Dabei gibt es ein globales Data Warehouse mit den Dimensionen Zeit und Ort und zwei Kennzahlen *anzStudenten* und *anzWimi*, welche Auskunft über die Anzahl der Studenten bzw. wissenschaftlichen Mitarbeiter pro Monat und pro Stadt geben. Im lokalen Data Mart gibt es hingegen drei Dimensionen, Zeit, Ort und Person, dafür jedoch nur eine Kennzahl *anzahl*, welche die Anzahl einer bestimmten Gruppe von Personen, wie zum Beispiel Studenten oder wissenschaftlichen Mitarbeitern, pro Stadt und pro Monat enthält.

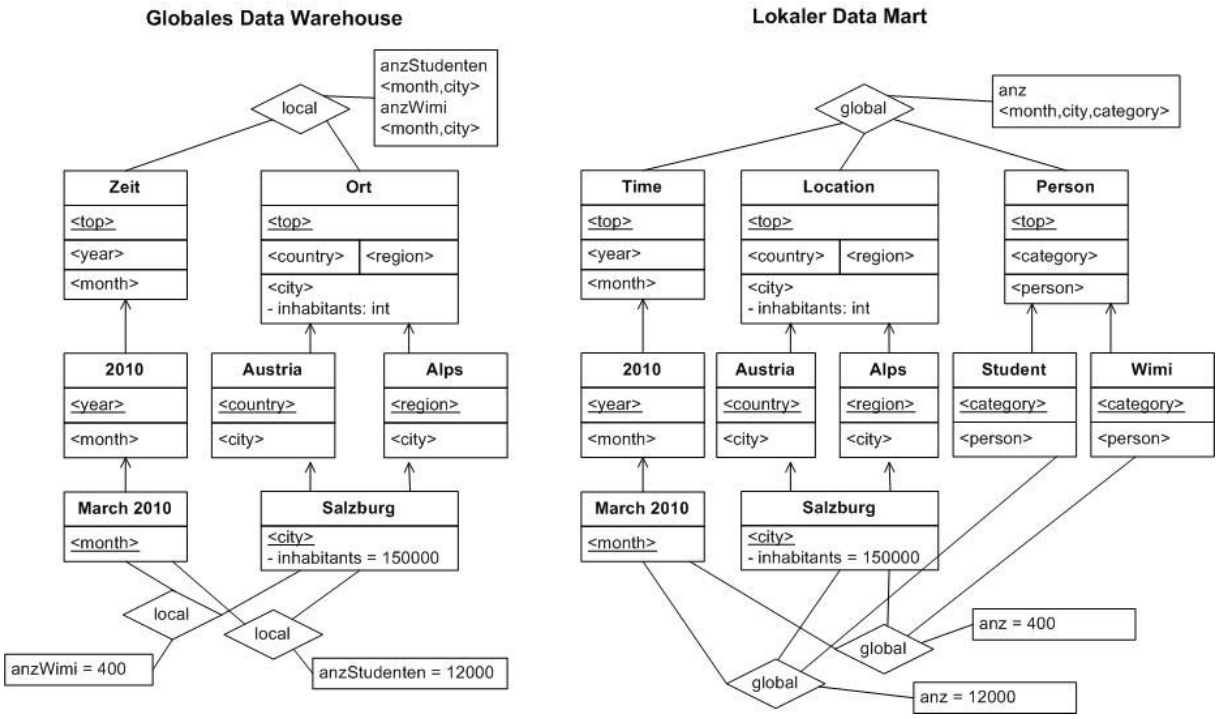

Abbildung 26: Pivoting Beispiel

Da die Kennzahl *anzWimi* der lokalen Kennzahl *Anzahl* auf dem Connection Level <month,city,category> für das Objekt wissenschaftlicher Mitarbeiter, und die Kennzahl *anzStudenten* der Anzahl für das Objekt Studenten entspricht, müssen entsprechende Pivot Mappings erstellt werden. Ein Pivot Mapping erhält den Namen der Kennzahl im globalen Schema, den Kennzahl Namen im lokalen Schema, das Objekt, dem die Kennzahl entspricht (z.B. Student, wissenschaftlicher Mitarbeiter oder nichtwissenschaftlicher Mitarbeiter) und eine Referenz auf den lokalen M-Cube, für den das Mapping gilt, als Parameter übergeben:

```
SELECT TREAT(Value(mc) AS mrel_c000000002_ty) INTO global_root
FROM   c000000002 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000002_ty('Time', 'Location', 'Person')) = 1;

SELECT TREAT(Value(mc) AS mcube_c000000002_ty) INTO global_mcube
FROM   mcubes mc
WHERE  mc.cname = 'global_cube';

SELECT REF(o) INTO obj_ref FROM d000000003 o WHERE oname = 'student';
SELECT REF(mc) INTO lcube_ref FROM mcubes mc WHERE  mc.cname =
      'local_cube';

global_root.add_pivot_mapping('anzStudents', 'anz', obj_ref, lcube_ref);

SELECT REF(o) INTO obj_ref FROM d000000003 o WHERE oname = 'wimi';

global_root.add_pivot_mapping('anzWimi', 'anz', obj_ref, lcube_ref);
```

Für den Dimensionsimport sind hier keine speziellen Mappings nötig, da beide Cubes dieselben Dimensionen Zeit und Ort verwenden.

Der Import des M-Cubes geht anschließend standardmäßig mit der *import_mcube* Methode (vgl. Kapitel 4.5):

```
SELECT REF(mc) INTO global_root_ref FROM c000000001 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000001_ty('Time', 'Location')) = 1;

SELECT REF(mc) INTO local_root_ref FROM c000000002 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000002_ty('Time', 'Location', 'Person')) = 1;

-- Import local M-Cube
mcube.import_mcube(global_root_ref, local_root_ref, NULL);
```

4.8.7. Unpivoting

Unpivoting stellt die Umkehrung von Pivoting dar und wird ebenfalls für die Behebung von Schema/Instanz Konflikten benötigt. Zur besseren Erläuterung wird das Unpivoting hier anhand des vorigen Beispiels erklärt (siehe Abbildung 26). Dieses Mal werden aber das globale Data Warehouse und der Data Mart vertauscht. Es gibt also ein globales Data Warehouse mit den drei Dimensionen Zeit, Ort und Person und einer Kennzahl *anzahl*, welche die Anzahl einer bestimmten Gruppe von Personen, wie zum Beispiel Studenten oder wissenschaftlichen Mitarbeitern, pro Stadt und pro Monat enthält. Im lokalen Data Mart gibt es nun nur mehr die Dimensionen Zeit und Ort und zwei Kennzahlen *anzStudenten* und *anzWimi*, welche Auskunft über die Anzahl der Studenten bzw. wissenschaftlichen Mitarbeiter pro Monat und pro Stadt geben.

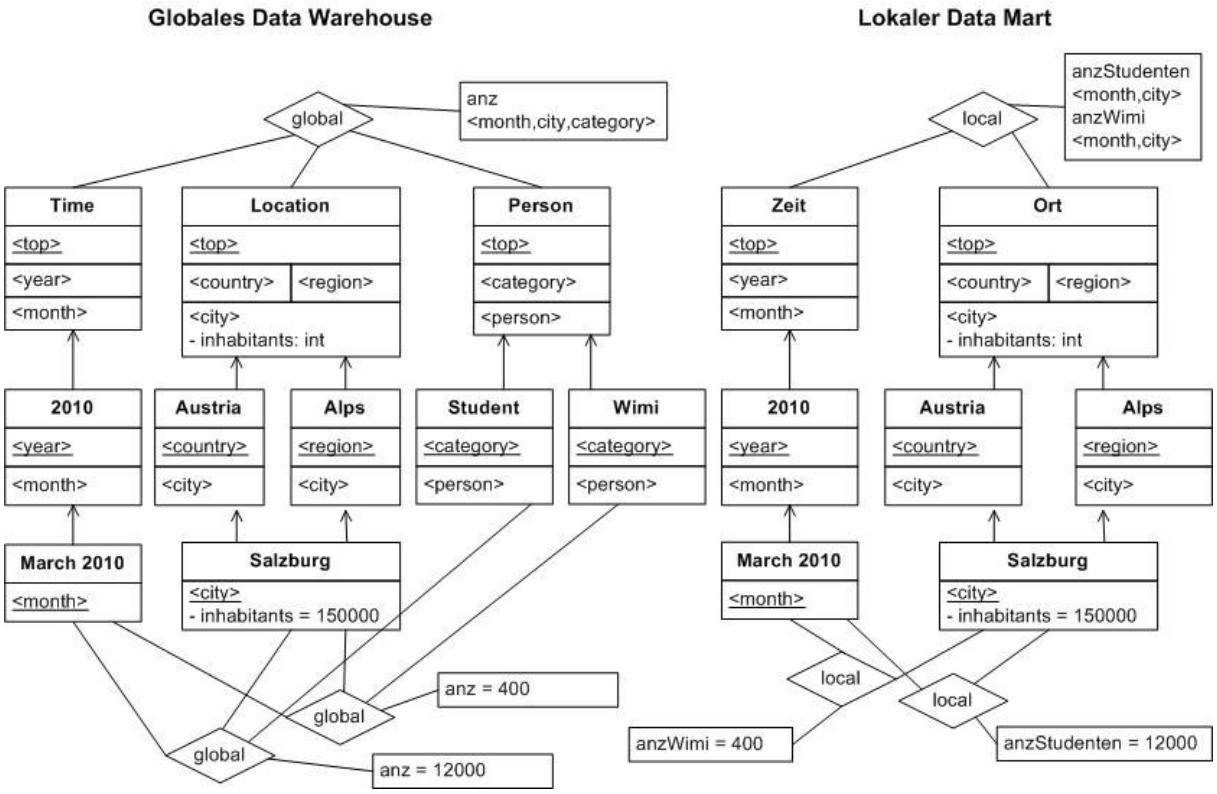

Abbildung 27: Unpivoting Beispiel

Um anzugeben, dass *anzStudenten* der Kennzahl *Anzahl* im globalen Schema mit dem Connection Level <month,city,category> für das Objekt Student und *anzWimi* der Kennzahl *Anzahl* im globalen Schema mit dem Connection Level <month,city,category> für das Objekt *Wimi* entspricht, sind Unpivot-Mappings erforderlich.

Ein Unpivot Mapping erhält ebenfalls den Namen der Kennzahl im globalen Schema, den Kennzahl Namen im lokalen Schema, das Objekt, dem die Kennzahl entspricht (z.B. Student, wissenschaftlicher Mitarbeiter oder nichtwissenschaftlicher Mitarbeiter), und eine Referenz auf den lokalen M-Cube, für den das Mapping gilt, als Parameter übergeben:

```
SELECT TREAT(Value(mc) AS mrel_c000000002_ty) INTO global_root
FROM   c000000002 mc
WHERE  mc.coordinate.equals(coordinate_c000000002_ty('Time', 'Location',
      'Person')) = 1;

SELECT TREAT(Value(mc) AS mcube_c000000002_ty) INTO global_mcube
FROM   mcubes mc
WHERE  mc.cname = 'global_cube';

SELECT REF(o) INTO obj_ref FROM d000000003 o WHERE oname = 'student';
SELECT REF(mc) INTO lcube_ref FROM mcubes mc WHERE  mc.cname =
      'local_cube';

global_root.add_unpivot_mapping('anz', 'anzStudenten', obj_ref, lcube_ref);

SELECT REF(o) INTO obj_ref FROM d000000003 o WHERE oname = 'wimi';
```

```
global_root.add_unpivot_mapping('anz', 'anzWimi', obj_ref, lcube_ref);
```

Für den Dimensionsimport sind wiederum keine speziellen Mappings nötig, da beide Cubes dieselben Dimensionen Zeit und Ort verwenden, der Import des M-Cubes geht anschließend wieder standardmäßig mit der *import_mcube* Methode (vgl. Kapitel 4.5):

```
SELECT REF(mc) INTO global_root_ref FROM c000000002 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000002_ty('Time', 'Location', 'Person')) = 1;

SELECT REF(mc) INTO local_root_ref FROM c000000001 mc
WHERE  mc.coordinate.equals(
      coordinate_c000000001_ty('Time', 'Location')) = 1;

-- Import local M-Cube
mcube.import_mcube(global_root_ref, local_root_ref, NULL);
```

4.9. Einschränkungen des Importprozesses

In den vorherigen Kapiteln wurde der Importprozess für den Standardfall und für eine Reihe von Sonderfällen beschrieben. Es können jedoch auch folgende spezielle Fälle auftreten, die derzeit nicht behandelt werden können:

- Inkonsistente Level-Hierarchien:

  Die Level-Hierarchien des globalen und des lokalen Schemas müssen konsistent sein. Es dürfen zwar im lokalen Schema zusätzliche Levels vorhanden sein beziehungsweise ein oder mehrere Levels fehlen, jedoch muss die Reihenfolge der Levels konsistent sein. Ein Beispiel hierfür wäre eine globale Produktdimension in der über dem Produkt, die Marke des Produkts und darüber die Produktkategorie ist und eine lokale Produktdimension in der jedes Produkt einer Kategorie zugeordnet wird und erst übergeordnet nach Marken gruppiert wird. Wenn nun für die Levels *Kategorie* und *Marke* Level-Mappings erstellt werden, könnte die lokale Produktdimension nicht mehr importiert werden, da die Reihenfolge der Levels in der Level-Hierarchie inkonsistent ist.
- Unterschiedliche Elternobjekte:

  Wenn ein M-Object sowohl global als auch lokal besteht, so muss es in beiden Schemata äquivalenten Eltenobjekten untergeordnet sein. Zum Beispiel darf in einer Dimension Person, in der einzelne Personen in wissenschaftliche bzw. nicht-wissenschaftliche Mitarbeiter unterschieden werden, eine Person mit dem Titel Bachelor nicht global unter dem Elternobjekt wissenschaftlicher Mitarbeiter und lokal

unter dem Elternobjekt nicht-wissenschaftlicher Mitarbeiter untergeordnet werden oder umgekehrt.

- Unpivot-Mappings:

  Eine weitere Einschränkung tritt beim Unpivot-Mapping auf. Wenn es, wie im Beispiel aus Kapitel 4.8.7, im lokalen M-Cube zwei Kennzahlen Anzahl Studenten und Anzahl wissenschaftlicher Mitarbeiter gibt und im globalen M-Cube eine Kennzahl Anzahl sowie eine zusätzliche Dimension Person mit den Objekten Student beziehungsweise wissenschaftlicher Mitarbeiter, dann kann hierfür ein Unpivot Mapping erstellt werden. Voraussetzung dafür ist jedoch, dass die Objekte Student bzw. wissenschaftlicher Mitarbeiter im globalen Schema bereits existieren. Gäbe es beispielsweise im lokalen M-Cube eine zusätzliche dritte Kennzahl Anzahl Professoren, so könnte hierfür kein Unpivot Mapping erstellt werden, da es kein Objekt *Professor* in der globalen Dimension Person gibt. Um ein entsprechendes Mapping erstellen zu können, müsste der Benutzer zuerst selbstständig ein M-Object *Professor* in der Dimension Person einfügen.

- Unvollständige Mappings:

  Eine Voraussetzung für den fehlerfreien Import eines Data Marts in ein globales, hetero-homogenes Data Warehouse ist, dass alle Mappings vollständig und korrekt gesetzt wurden. Dies gilt für alle Mapping-Typen.

5. Implementierung des Importprozesses

Während sich die vorherigen Kapitel mit der Funktionalität und den verfügbaren Mappings aus Benutzersicht beschäftigt hat, wird hier näher auf die interne Realisierung eingegangen. Dazu werden zunächst in Kapitel 5.1 die Anpassungen des Prototyps von [Schütz 2010] aufgezeigt. Es wird also genau beschrieben, wo und wie die einzelnen Mappings gespeichert werden. Dabei ist wiederum zu beachten, dass die Anpassungen an der bereits erweiterten Version des Prototyps [HH-DW] vorgenommen wurden. Die Erweiterungen des Prototyps sind in [Schütz 2011] ausführlich beschrieben. In den Kapiteln 5.2 und 5.3 werden die internen Abläufe beim Importieren der Dimensionen beziehungsweise des M-Cubes des lokalen Data Marts näher beschrieben. Anschließend wird in Kapitel 5.4 aufgezeigt, wie und wofür automatisch Mappings erstellt und wie die Aliase der Levels, Attribute und M-Objects in Mappings übersetzt werden. In Kapitel 5.6 wird der Ablauf beim Importieren von Dimensionen mit zusätzlichen Levels genauer beschrieben und gezeigt wie dabei das bestehende Schema angepasst wird. Im letzten Abschnitt wird auf die zur Verfügung gestellten Hilfsfunktionen eingegangen.

5.1. Speicherung der Mappings

Generell werden die Objekttypen der Mappings alle statisch angelegt und nicht weiter konkretisiert. Dies hat den Vorteil, dass sie sehr einfach aufgebaut und somit leicht verständlich sind. Zudem ist es für manche Mappings, wie beispielsweise den Level-, den Attribut- und den M-Object-Mappings, auch nicht nötig, sie zu konkretisieren.

Das statische anlegen der Objekttypen bringt jedoch auch einige Nachteile mit sich. Ein Nachteil ist, dass für die Mappings, die für den Import eines M-Cubes erforderlich sind, deshalb nur Referenzen auf den allgemeinen Super-Typ und nicht auf die konkretisierten Dimensionen bzw. auf den konkretisierten M-Cube verwendet werden können. Die statische Generierung der Mapping-Typen und des Import-Algorithmus hat auch zur Folge, dass die Import-Funktionalität dynamisch entwickelt werden musste, da die eigentlichen, Cube-spezifischen Datentypen erst zur Laufzeit bekannt sind. Als Konsequenz dessen ließ es sich auch nicht verhindern, dass gewisse Codepassagen sehr ähnlich beziehungsweise gar redundant vorkommen.

Ein weiterer Nachteil ist das suboptimale Laufzeitverhalten des Systems. Zwar ist es bei der aktuellen Lösung nicht erforderlich, die konkretisierten Typen dynamisch zu erstellen, jedoch

müssen bei einer generischen Lösung die Typen stets zur Laufzeit überprüft werden, was Performance-Einbußen mit sich bringt. Außerdem könnten bei einer dynamischen Generierung Indizes angelegt werden. Dies würde eine weitere Performancesteigerung bewirken. Da das Laufzeitverhalten Systems aber vorerst nicht vorrangig ist, wurde gegenwärtig darauf verzichtet die Objekttypen der Mappings dynamische zu konkretisieren.

In der Folge wird die Speicherung der einzelnen Mapping-Typen genauer beschrieben. Dabei wird gezeigt, wie diese aufgebaut sind und wo sie gespeichert werden. Wie bereits in Kapitel 4.3.2 beschrieben, werden die Mappings an drei verschiedenen Orten hinzugefügt:

- Beim M-Object
- Bei einer M-Relationship
- Beim M-Cube

5.1.1. Mappings beim M-Object

Alle Mappings, die für den Dimensionsimport erforderlich sind, werden beim M-Object gespeichert. Dies betrifft also das Level-, das Attribut- und das M-Object-Mapping. Diese werden beim M-Object gespeichert, da sie sich auf Heterogenitäten zwischen dem globalen und dem lokalen Schema auf Ebene der M-Objects beziehen. Um Mappings zu einem M-Object hinzufügen zu können, wurde der M-Object Typ selbst angepasst. In Abbildung 28 ist der neue M-Object Typ zu sehen. Allerdings ist zu beachten, dass aus Gründen der Übersichtlichkeit nur der neue hinzugefügte Teil dargestellt ist. Alle anderen Attribute des Typs mobject_ty und ihre Objekttypen sind nicht abgebildet.

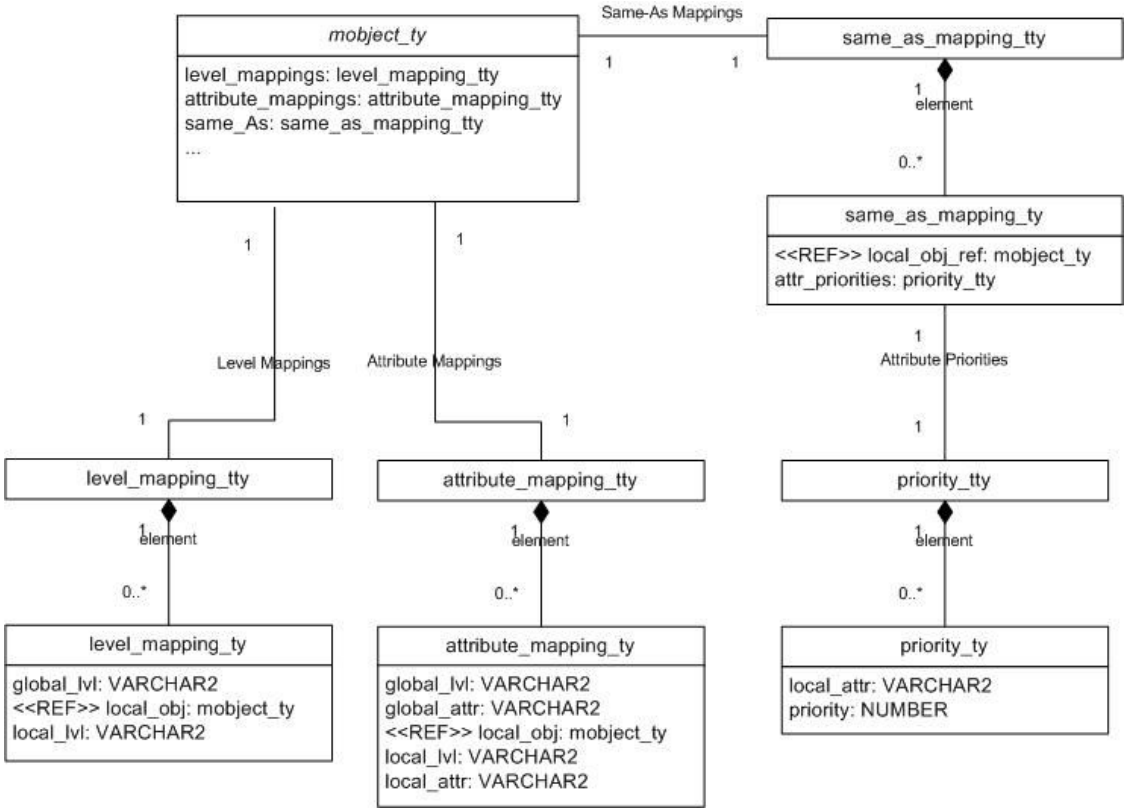

Abbildung 28: mobject_ty

Dem M-Object Typen wurden also drei Attribute für die Level-, die Attribut- und die M-Object-Mappings hinzugefügt und für die Mappings eigene Typen erstellt. Jedes M-Object enthält eine verschachtelte Tabelle mit den Level-Mappings. Ein Level-Mapping besteht aus dem Namen des Levels im globalen Schema, einer Referenz auf das Wurzelobjekt in der lokalen Dimension und dem Namen des Levels im lokalen Schema. Attribut-Mappings sind ähnlich aufgebaut. Ein Attribut-Mapping besitzt zusätzlich lediglich noch den Namen des Attributs im globalen und den Attributnamen im lokalen Schema.
Außerdem wird für jedes Objekt eine geschachtelte Tabelle mit den M-Object-Mappings inklusive der Prioritäten für die Attributwerte gespeichert. Die *Same_As_Mappings* Tabelle besteht aus einzelnen M-Object-Mappings, die wiederum aus einer Referenz auf ein lokales Objekt und einer Tabelle für die Prioritäten der Attributwerte des Objekts besteht. In dieser verschachtelten Tabelle ist ein einzelner Datensatz vom Typ priority_ty. Dieser setzt sich aus dem Attributnamen im globalen Schema und einer Priorität (= „0“, wenn der globale Wert Priorität hat, und „1“, wenn der lokale Priorität hat) zusammen.

In der Problemstellung in Kapitel 4.1 sollte eine lokale Dimension *Musik* in eine globale Produktdimension integriert werden. Dabei waren Level-Mappings für die Levels *kat* und *titel*

erforderlich. Tabelle 4 zeigt einen Ausschnitt der M-Object-Tabelle, nachdem die erforderlichen Level-Mappings vom Benutzer hinzugefügt werden, wobei für jedes Objekt der Name (Spalte *oname*) und die Level-Mappings angezeigt werden. Man beachte, dass für alle Objekte, die keine Level-Mappings besitzen, bei der Initialisierung der M-Object-Tabelle eine leere Tabelle level_mapping_tty() eingefügt wird.

Oname	Level_Mappings		
	global_lvl	local_obj	local_lvl
Product	category	REF musik_obj	kat
	model	REF musik_obj	titel
Car	level_mapping_tty()		
Porsche911	level_mapping_tty()		
Porsche911CarreraS	level_mapping_tty()		

Tabelle 4: M-Object-Tabelle mit Level-Mappings

Tabelle 5 zeigt einen Ausschnitt aus der M-Object-Tabelle, nachdem die Attribut-Mappings für die Attribute *verantwortlicher*, *steuersatz* und *listenPreis* hinzugefügt wurden, wobei dieses Mal der Übersichtlichkeit halber lediglich der Objektname und die Attribut-Mappings angeführt werden.

Oname	Attribute_Mappings				
	global_lvl	global_attr	local_obj	local_lvl	local_attr
Product	category	catMgr	REF musik_obj	kat	verantwortlicher
	category	taxRate	REF musik_obj	kat	steuersatz
	model	costs	REF musik_obj	titel	listenPreis
Car	attribute_mapping_tty()				
Porsche911	attribute_mapping_tty()				
Porsche911CarreraS	attribute_mapping_tty()				

Tabelle 5: M-Object-Tabelle mit Attribut-Mappings

Da in der ursprünglichen Problemstellung keine M-Object-Mappings erforderlich waren, werden nun in der Tabelle 6 die M-Object-Mappings anhand des Beispiels aus Kapitel 4.7.4 angezeigt. In letzterem Beispiel ging es darum, eine Dimension *Österreich* in eine globale Ortsdimension zu integrieren und dabei den Wert des Attributs Einwohner für das Objekt Salzburg zu belassen, beim Objekt Innsbruck jedoch den vorherigen Wert mit dem lokalen zu überschreiben.

Oname	Same_As		
	local_obj_ref	attr_priorities	
		attribute	priority
Location	same_as_mapping_tty()		
Austria	REF österreich_obj	priority_tty()	
Alps	same_as_mapping_tty()		
Salzburg	REF sbg_obj	inhabitants	0
Innsbruck	REF innsbruck_obj	inhabitants	1

Tabelle 6: M-Object-Tabelle mit M-Object-Mappings

5.1.2. Mappings beim M-Cube

Bei den für den M-Cube-Import erforderlichen Mappings wird in solche, die beim M-Cube gespeichert werden, und jene, die bei M-Relationships gespeichert werden, unterschieden. Die Begründung hierfür ist, dass beim M-Cube-Import die Dimension- und die Hidden-Dimension-Mappings für den gesamten Cube relevant sind, die Measure-, die M-Relationship-, die Pivot- und die Unpivot-Mappings jedoch nur für einzelne M-Relationships oder Sub-Cubes.

Für den M-Cube selbst relevant sind Mappings, die die Dimensionen des Cubes betreffen, also das Dimension- und das Hidden-Dimension-Mapping. Der Typ mcube_ty wurde daher um die zwei Attribute *dimension_mappings* und *hidden_dim_mappings* erweitert, welche jeweils verschachtelte Tabellen darstellen.

In der verschachtelten Tabelle *dimension_mappings* ist ein Tupel ein einzelnes Dimension-Mapping, wobei dieses wiederum aus der Referenz auf die globale Dimension, der Referenz auf die lokale Dimension und einer Referenz auf den lokalen M-Cube besteht. In der Problemstellung in Kapitel 4.1 verwendet das globale Data Warehouse eine Produktdimension, der lokale Data Mart aber stattdessen eine Dimension *Musik*. Somit ist für die Dimension *Musik* ein Dimension-Mapping erforderlich, mit einer Referenz auf die globale Produktdimension, einer auf die lokale Dimension *Musik* und einer auf den M-Cube des lokalen Data Marts. Die letzte Referenz ist erforderlich, um ein Mapping einem lokalen M-Cube zuordnen zu können. Wenn mehrere lokale Cubes importiert werden können, ist es erforderlich für alle Cube Importe Mappings zu erstellen. Die Referenz auf den lokalen Cube ist nötig um anzugeben für welchen lokalen Cube ein Mapping gültig ist.

In der verschachtelten Tabelle *hidden_dim_mappings* ist ein einzelner Datensatz ein Hidden-Dimension-Mapping. Dieses besteht aus einer Referenz auf den lokalen Cube, den Namen der versteckten Dimension und einer Objektreferenz. Wenn wir uns an das Beispiel aus Kapitel 4.8.2 zurückerinnern, wo im lokalen Data Mart keine Zeitdimension existiert, aber bekannt

ist, dass alle Verkäufe 2010 stattgefunden haben, kann ein Hidden-Dimension-Mapping für den lokalen Cube mit den Dimensionsnamen *time_dim* und einer Referenz auf das Objekt *Year2010* in der globalen Dimension *time_dim* erstellt werden.
Abbildung 29 zeigt die Anpassungen am *mcube_*_ty*. Dabei ist zu beachten, dass auch hier aus Gründen der Übersichtlichkeit nur die neu hinzugefügten Attribute dargestellt sind.

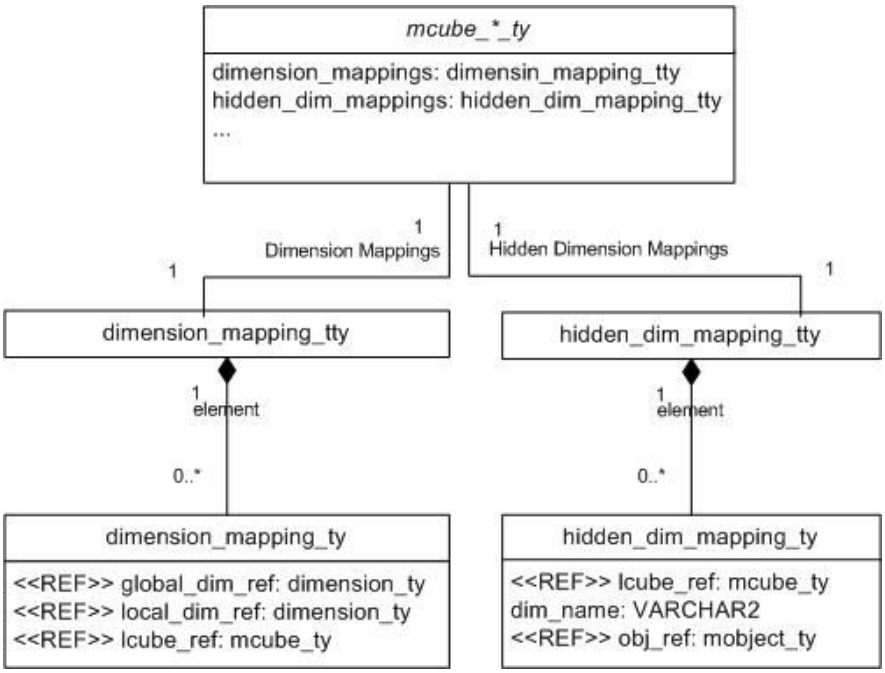

Abbildung 29: mcube_*_ty

Tabelle 7 zeigt einen Ausschnitt der M-Cubes-Tabelle, nachdem ein Dimension-Mapping für die Produkt- und die Musik Dimension erstellt wurde, wobei aus Gründen der Übersichtlichkeit wieder nur die Namen der M-Cubes (Attribut *cname*) und die Dimension-Mappings angezeigt werden. Bei allen Cubes, die keine Dimension-Mappings besitzen, ist das Attribut d*imension_mappings* nur eine leere Tabelle vom Typ *dimension_mapping_tty*.

Cname	Dimension_Mappings		
	global_dim_ref	local_dim_ref	lcube_ref
SalesCube	REF product_dim	REF musik_dim	REF local_cube
LocalCube	dimension_mapping_tty()		

Tabelle 7: M-Cube Tabelle mit Dimension-Mappings

Tabelle 8 zeigt einen Ausschnitt der M-Cubes-Tabelle mit den Namen der M-Cubes und den Hidden-Dimension-Mappings, nachdem ein Mapping für die fehlende Zeitdimension aus dem oben beschriebenen Beispiel angelegt wurde. Auch hier haben M-Cubes ohne entsprechenden Mappings nur eine leere Tabelle, dieses Mal jedoch vom Typ *hidden_dim_mapping_tty*.

Cname	Hidden_Dim_Mappings		
	lcube_ref	dim_name	obj_ref
SalesCube	REF local_cube	time_dim	REF year2010_obj
LocalCube	hidden_dim_mapping_tty()		

Tabelle 8: M-Cube Tabelle mit Hidden-Dimension-Mappings

5.1.3. Mappings bei den M-Relationships

Abgesehen von den Dimension- und den Hidden-Dimension-Mappings werden für den Import von M-Cubes auch Measure-, Pivot-, Unpivot- und M-Relationship-Mappings benötigt. Letztere betreffen nicht den M-Cube selbst und werden somit bei den M-Relationships gespeichert, besser gesagt bei jener M-Relationship, wo der lokale Cube eingefügt wird. Um die Mappings speichern zu können, musste das bestehende Schema überarbeitet und dem mrel_*_ty Typ zusätzliche Attribute hinzugefügt werden. Abbildung 30 zeigt die zum Cube-spezifischen mrel_*_ty neu hinzugefügten Attribute und die damit verknüpften neuen Mapping-Typen. Der Typ mrel_*_ty hat nun vier zusätzliche Attribute:

- Measure_mappings:

  Die Measure-Mappings werden als verschachtelte Tabelle gespeichert, wobei jeder Datensatz dieser Tabelle ein Measure-Mapping vom Typ *measure_mapping_ty* ist. Dieser Typ wiederum besteht aus den Namen der Kennzahl im globalen bzw. im lokalen Schema und einer Referenz auf den lokalen Cube, damit der Import Algorithmus weiß, für welchen Cube das Mapping relevant ist, wenn mehrere Cubes importiert werden.

- Same_mrels:

  Jede M-Relationship im globalen Schema kann auch in einem oder mehreren lokalen Cube(s) vorkommen. Wenn eine M-Relationship auch lokal vorkommt, muss ein M-Relationship-Mapping hierfür erstellt werden. M-Relationship-Mappings werden in der verschachtelten Tabelle *Same_mrels* gespeichert, wobei diese aus der Referenz auf die lokale M-Relationship und einer Tabelle für die Prioritäten der Kennzahl-Werte besteht. In letzterer ist ein einzelner Datensatz vom Typ *mrel_priority_ty*, welcher aus dem Kennzahl Namen im globalen Schema und einer Priorität besteht (analog zu den Attributprioritäten ist die Priorität = „0“, wenn der globale Wert Priorität hat, und „1“, wenn der lokale Priorität hat).

  M-Relationship-Mappings wären grundsätzlich gar nicht erforderlich, da äquivalente M-Relationships auch durch Vergleichen der M-Objects ihrer Koordinaten gefunden werden könnten. Da dies jedoch ineffizient wäre, weil jedes M-Object unter einem

anderen Namen im globalen Schema gespeichert wird und hierfür somit zusätzlich für jedes einzelne Objekt die M-Object-Mappings überprüft werden müssten, wurde für M-Relationships, die sowohl im globalen als auch im lokalen Cube bestehen, ein M-Relationship-Mapping-Typ erstellt.

- Pivot_mappings

  Pivot-Mappings werden in der geschachtelten Tabelle *pivot_mappings* gespeichert. Diese Tabelle ist vom Typ *pivot_mapping_tty* und einzelne Datensätze darin Pivot-Mappings vom Typ *pivot_mapping_ty*. Der Typ *pivot_mapping_ty* besteht aus dem Namen der Kennzahl im globalen Schema, den lokalen Namen der Kennzahl, der Referenz des M-Objects und einer Referenz auf den lokalen Cube. Die Referenz auf den lokalen Cube ist wiederum nötig, damit der Import Algorithmus weiß, für welchen Cube das Mapping relevant ist, wenn mehrere Cubes importiert werden.

- Unpivot_mappings:

  Unpivot-Mappings werden in der verschachtelten Tabelle *unpivot_mappings* gespeichert. Sie sind wie die Pivot Mappings vom Typ *pivot_mapping_tty*.

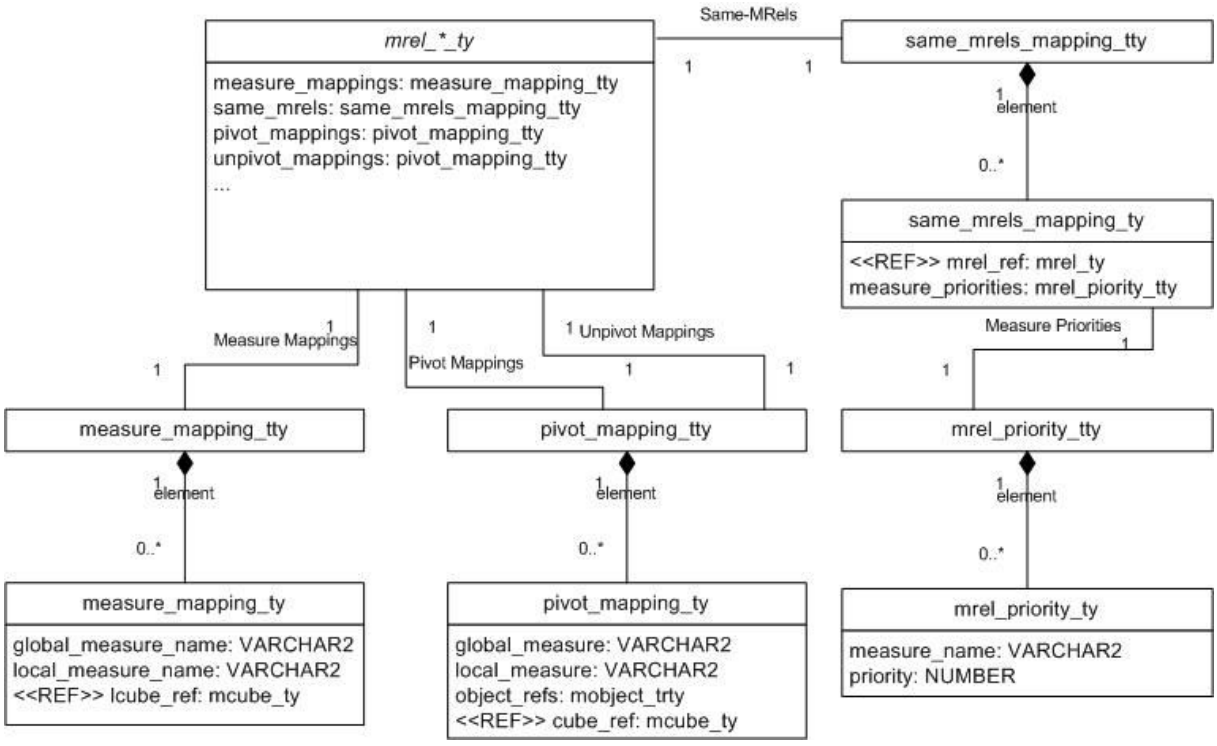

Abbildung 30: mrel_*_ty

In der Problemstellung galt es, ein Measure-Mapping für den Umsatz zu erstellen, da die entsprechende Kennzahl global als *revenue* bezeichnet wurde. Tabelle 9 zeigt die M-Relationship-Tabelle, nachdem das Measure-Mapping hinzugefügt wurde. Der

Übersichtlichkeit halber werden nur die ID und die Koordinate der M-Relationship und das entsprechende Measure-Mapping dargestellt. Hat eine M-Relationship kein Measure-Mapping, so ist das Attribut eine leere Tabelle vom Typ *measure_mapping_tty*.

ID	Coordinate	Measure_Mappings		
		global_measure_name	local_measure_name	lcube_ref
r000000001	coordinate_c000000001_ty('Time','Location','Product')	revenue	Umsatz	REF local_cube

Tabelle 9: M-Relationship-Tabelle mit Measure-Mapping

Im Beispiel aus Kapitel 4.8.5 galt es für die im lokalen Cube äquivalente M-Relationship der Verkäufe des Produkts Porsche 911 Carrera S in Salzburg im März 2010 ein M-Relationship-Mapping zu erstellen. Dabei sollte der Wert des Umsatzes des globalen Data Warehouses beibehalten, jener der verkauften Menge allerdings aus dem lokalen Data Mart genommen werden. Tabelle 10 zeigt einen Ausschnitt der M-Relationship-Tabelle mit dem hinzugefügten M-Relationship-Mapping. Man beachte, dass das *Same_Mrels* Attribut für alle M-Relationships ohne M-Relationship-Mapping eine leere Tabelle vom Typ *same_mrel_mapping_tty* ist. Für alle M-Relationships mit M-Relationship-Mapping, jedoch ohne Measure Prioritäten, ist der *measure_priorities* Wert eine leere Tabelle vom Typ *mrel_priority_tty*.

ID	Coordinate	Same_Mrels		
		mrel_ref	measure_priorities	
			measure	priority
r00000001	coordinate_c000000001_ty('Time','Location','Product')	REF local_mrel	mrel_priority_tty()	
r00000002	coordinate_c000000001_ty('Month_3_2010', Salzburg','Porsche911CarreraS')	REF local_mrel	qtySold	0
			revenue	1

Tabelle 10: M-Relationship-Tabelle mit M-Relationship-Mappings

Tabelle 11 zeigt einen Ausschnitt der M-Relationship-Tabelle mit den für das Beispiel aus Kapitel 4.8.6 erforderlichen Mappings. Dabei wurden Pivot Mappings für die lokale Kennzahl *anzahl* erstellt, die im globalen Schema den Kennzahlen Anzahl wissenschaftlicher Mitarbeiter und Anzahl von Studenten entspricht.
Tabelle 12 zeigt abschließend einen Ausschnitt der M-Relationship-Tabelle, nachdem die für das Beispiel aus Kapitel 4.8.7 erforderlichen Unpivot-Mappings eingefügt wurden.

ID	Coordinate	Pivot Mappings			
		global_measure	local_measure	obj_refs	cube_ref
r000000001	coordinate_c000000001_ty('Time','Location')	anzWimi	anzahl	REF wimi_obj	REF local_cube
		anzStudents	anzahl	REF student_obj	REF local_cube
r000000002	coordinate_c000000001_ty('Time','Innsbruck')	pivot_mapping_tty()			

Tabelle 11: M-Relationship-Tabelle mit Pivot Mappings

ID	Coordinate	Unpivot Mappings			
		global_measure	local_measure	obj_refs	cube_ref
r000000001	coordinate_c000000001_ty('Time','Location','Person')	anzahl	anzWimi	REF wimi_obj	REF local_cube
		anzahl	anzStudents	REF student_obj	REF local_cube
r000000002	coordinate_c000000001_ty('Time','Innsbruck','Person')	pivot_mapping_tty()			

Tabelle 12: M-Relationship-Tabelle mit Unpivot Mappings

5.2. Import der Dimensionen

Der Import der Dimensionen erfolgt semi-automatisch durch Ausführen der *import_branch* Methode. Der Benutzer muss, bevor er eine Dimension importieren kann, die erforderlichen Mappings erstellen, und anschließend die *import_branch* Methode selbst aufrufen. Diese befindet sich im Package *dimension* und bekommt den Einfügepunkt in der globalen Dimension, das Wurzelobjekt der lokalen Teildimension, die importiert werden soll, und die Aliase für die M-Objects, die Levels und die Attribute als Parameter. Der Einfügepunkt in der globalen Dimension ist jenes M-Object in der globalen Dimension, unter dem die lokale Dimension eingefügt werden soll. Der Import selbst erfolgt anschließend automatisch. In zukünftigen Versionen ist es vorgesehen, dass bei Konflikten der Benutzer eine Meldung bekommt und korrigierend eingreifen kann. Derzeit wird jedoch bloß eine Fehlermeldung ausgegeben, der Integrationsprozess selbst läuft allerdings weiter, wenn kein gravierender Fehler in Form einer unvorhergesehenen/unerwarteten Exception auftritt.

Die Methode *import_branch* befindet sich im Package *dimension*. Sie befindet sich also nicht im dynamisch generierten Typ dimension_*_ty. Das Konkretisieren der Import Funktionalität hätte den Vorteil, dass die konkreten mobject_*_ty Typen verwendet werden könnten und der Objekttyp zur Laufzeit geprüft werden würde. Für den Import der Dimensionen ist es allerdings vollkommen ausreichend, den allgemeinen mobject_ty zu verwenden, da dieser

ohnehin nur verwendet wird, um anzugeben, für welche lokale Dimension ein Mapping gilt. Ansonsten ist der Import der Dimensionen typunabhängig.

Der folgende Abschnitt soll den Algorithmus beim Importieren einer Dimension grob beschreiben:

1. Überprüfen der lokalen Level-Namen

   Im ersten Schritt wird die Methode *correct_level_names* aufgerufen. Diese stellt sicher, dass kein Level-Name der zu importierenden Teildimension die Unique Induction Regel verletzt. Sollte dies bei einem Level der Fall sein, wird der Name des Levels mit der Methode *get_unique_short_name* im Package *identifiers* umbenannt, und der neue Name temporär gespeichert, in dem ein Alias für den neuen Level-Namen angelegt wird.

2. Importieren des lokalen Wurzelobjekts

   Im zweiten Schritt wird das lokale Wurzelobjekt, das der *import_branch* Methode mitgegeben wurde, importiert. Der Import eines Objekts erfolgt mit der *import_mobject* Methode. Nähere Information zum Importieren eines M-Objects befindet sich in Kapitel 5.2.1.

3. Durchlaufen aller Kindobjekte der lokalen Wurzel

   Nachdem die lokale Wurzel importiert wurde, werden mittels eines Oracle-Cursors alle direkten und transitiven Kindobjekte der Wurzel durchlaufen, und für jedes Objekt wird die *import_mobject* Methode zum Importieren aufgerufen. Man beachte, dass die Kindobjekte in geordneter Art und Weise durchlaufen werden, das heißt es werden zuerst alle Objekte des Top-Levels der Dimension importiert, danach alle des Second-Top-Levels usw. Dies ist erforderlich, da beim Einfügen von Objekten die Elternobjekte mitgegeben werden und somit bereits in der globalen Dimension existieren müssen.

4. Setzen der Attributwerte

   Nachdem alle Objekte der zu importierenden Teildimension im globalen Schema eingefügt wurden, wird die Methode *set_attribute_values* aufgerufen, um die Attributwerte zu setzen. Details zum Setzen der Attributwerte findet man im Kapitel 5.2.2.

5. Importieren der Metadaten

   Abschließend wird die Methode *set_attribute_metadata* aufgerufen, um die Metadaten der lokalen Dimension zu importieren. Diese ruft die Methode *set_metadata* für die lokale Wurzel auf und durchläuft anschließend alle Objekte der lokalen

Teildimensionen, die eine Metadaten Tabelle besitzen, und ruft für diese die *set_metadata* Methode auf.

Set_metadata ist für das Importieren der Metadaten eines Objekts verantwortlich. Man beachte, dass im derzeitigen System die Metadaten zwar importiert werden, jedoch hierfür keine Heterogenitäten berücksichtigt werden. Aktuell werden also noch keine unterschiedlichen Einheiten bei den Kennzahlen berücksichtigt.

5.2.1. Importieren eines M-Objects

Zum Importieren eines einzelnen Objekts wird die Methode *import_mobject* verwendet. Diese geht folgendermaßen vor:

1. M-Object Namen dem globalen Schema anpassen:
   Im ersten Schritt werden die M-Object Aliase, die der *import_branch* Methode mitgegeben wurden, durchsucht und, falls ein Alias für das M-Object gefunden wird, der Objektname mit dem Alias ausgetauscht.
2. Überprüfen, ob das Objekt bereits im globalen Schema existiert:
   Existiert ein Objektname bereits im globalen Schema, und gibt es für das Objekt kein M-Object-Mapping, so wird dem Benutzer mittels *dbms_output* eine Warnung ausgegeben, dass der Name bereits existiert. Anschließend wird das Objekt mit der Methode *get_unique_short_name* im Package *identifiers* umbenannt und das Objekt in weiterer Folge mit dem neuen Namen eingefügt. Der neue Name wird nirgends explizit gespeichert, da ohnehin über das später automatisch abgeleitete M-Object-Mapping ein Bezug zwischen dem integrierten Objekt in der globalen und dem Originalobjekt in der lokalen Dimension hergestellt werden kann.
3. Objekt ins globale Schema einfügen:
   Existiert ein Objekt noch nicht in der globalen Dimension (wenn kein M-Object-Mapping für das Objekt gefunden wird), so wird es neu eingefügt, ansonsten setzt der Algorithmus gleich bei Punkt 4 (*Objekt im globalen Schema anpassen*) fort:
   3.1. Aktualisieren der Elternobjekte
      Bevor ein Objekt in die globale Dimension eingefügt werden kann, müssen die Parent-M-Objects angepasst werden. Hierfür werden alle lokalen Elternobjekte durchlaufen und jedes lokale Elternobjekt mit dem äquivalenten Objekt in der globalen Dimension ersetzt. Wenn beispielsweise ein M-Object *Salzburg* importiert werden soll, welches lokal ein Parent-M-Object *Österreich* hat, so

muss das Parent-M-Object *Österreich* durch das Objekt *Austria* der globalen Dimension ersetzt werden damit das M-Object *Salzburg* in der globalen Dimension angelegt werden kann.

3.2. Anpassen des Top-Levels

Im zweiten Schritt wird der Level-Name des Top-Levels des Objekts an das globale Schema angepasst. Hierfür wird geprüft, ob für den Level ein Mapping besteht. Existiert ein Mapping, wird der Name des Levels ausgetauscht. Wenn nicht, wird geprüft, ob ein Alias für den Level besteht, und wenn ein Alias existiert, dieses als Name verwendet. Wenn zum Beispiel der Level *city* lokal als *Stadt* bezeichnet ist, muss der Top-Level des M-Objects *Salzburg* beim Importieren von *Stadt* auf *city* geändert werden.

3.3. Aktualisieren der Level-Hierarchie

Dieser Schritt ist ähnlich dem vorherigen, nur dass hier anstatt des Top-Levels jeder Level-Name in der Level-Hierarchie des einzufügenden Objekts an das globale Schema angepasst wird. Dafür wird die Level-Hierarchie durchlaufen und für jeden Level-Namen geprüft, ob ein Mapping bzw. ein Alias für den Level definiert wurde.

3.4. Hinzufügen von Levels zum globalen Schema

In diesem Schritt wird die Methode *add_lvl_to_global_schema* aufgerufen. Diese prüft, ob die lokale Dimension zusätzliche Levels hat, die im globalen Schema nicht existieren, und fügt diese gegebenfalls ins globale Schema ein. Nähere Information dazu befindet sich im Kapitel 5.6.

3.5. Hinzufügen von lokal fehlenden Levels

Hier werden die aktualisierten Elternobjekte des einzufügenden Objekts durchlaufen und geprüft, dass das neue Objekt auch tatsächlich am Second-Top-Level des Elternobjekts ist. Wenn dies nicht der Fall ist, also wenn in der lokalen Dimension ein oder mehrere Level fehlen, so wird der fehlende Level in die aktualisierte Level-Hierarchie eingefügt, ein *<parentLevel>_unknown* Objekt für den lokal fehlenden Level in der globalen Dimension angelegt und als Parent des neu einzufügenden Objekts gesetzt. Dies ist beispielsweise der Fall, wenn eine lokale Ortsdimension mit den Levels Stadt und Land existiert, es im globalen Schema jedoch die Levels Stadt, Bundesland und Land gibt. Wenn nun eine Stadt Salzburg mit dem Elternobjekt Österreich aus dem lokalen Schema eingefügt werden soll, so würde zuerst das Parent-M-Object Österreich geprüft

werden, dieses hat im globalen Schema als zweithöchsten Level allerdings das Bundesland und nicht die Stadt. Also wird in die Level-Hierarchie der lokalen Stadt Salzburg das Bundesland eingefügt und ein Objekt *Österreich_bundesland_unknown* in der globalen Dimension erstellt (wenn dieses nicht bereits existiert) und der Stadt Salzburg als Elternobjekt zugewiesen. Das Objekt *Österreich_bundesland_unknown* hat wiederum Österreich als Elternobjekt.

3.6. Prüfen ob alle Elternobjekte gesetzt wurden

Nachdem die Level-Hierarchie korrigiert wurde, wird geprüft, ob alle Elternobjekte gesetzt wurden. Wenn dies nicht der Fall ist, wird ein *<parentLevel>_unknown* Objekt erstellt und dem zu importierenden Objekt zusätzlich als Parent hinzugefügt. Dies kann beispielsweise sein, wenn eine Ortsdimension importiert werden soll, die lediglich aus einer Stadt und einem Land besteht, das globale Schema hingegen jede Stadt einem Land und einer Region zuweist. Somit würde eine neu zu importierende Stadt aus dem lokalen Schema im globalen Schema einer unbekannten Region zugewiesen werden (Objekt *region_unknown*). Existiert dieses *<parentLevel>_unknown* Objekt noch nicht, wird es in die globale Dimension eingefügt.

3.7. Einfügen des Objekts ins globale Schema

Nachdem alle Heterogenitäten an das globale Schema angepasst wurden, kann das Objekt schlussendlich in die globale Dimension eingefügt werden.

3.8. Erstellen eines M-Object-Mappings für das neue Objekt

Für jedes importierte Objekt wird ein M-Object-Mapping mit einer Referenz auf das lokale Objekt erstellt. Nähere Information dazu findet man in Kapitel 5.4.1.

3.9. Hinzufügen der Attribute des neuen Objekts

Hierfür wird die Methode *add_attributes* aufgerufen. Diese prüft, ob das Objekt im lokalen Schema ein oder mehrere Attribute definiert. Wenn ja, werden der Attributname und der Level, für den es definiert wurde, an das globale Schema angepasst. Anschließend wird der Datentyp des Objekts herausgesucht und das Attribut, sofern es in der globalen Dimension noch nicht existiert, dem neu in die globale Dimension eingefügten Objekt hinzugefügt.

3.10. Hinzufügen von Mappings

Dieser Punkt wird nur für das Wurzelobjekt der lokalen Dimension ausgeführt. Hierbei werden die M-Object-, die Level- und die Attribut-Aliase, die der

import_branch Methode mitgegeben wurden, in Mappings übersetzt. Der genaue Ablauf dabei wird in Kapitel 0 beschrieben.

3.11. Speichern der Änderungen

Abschließend werden die Änderungen in der globalen Dimension persistiert.

4. Objekt im globalen Schema anpassen:

Existiert ein Objekt bereits in der globalen Dimension (wenn ein M-Object-Mapping für das Objekt gefunden wird), so wird es zwar nicht erneut eingefügt, allerdings wird die Level-Hierarchie angepasst und geprüft, ob es im lokalen Schema zusätzliche Attribute definiert.

4.1. Aktualisieren der Level-Hierarchie

In diesem Schritt wird die Level-Hierarchie durchlaufen und für jeden einzelnen Level-Namen geprüft, ob ein Mapping für den Level besteht und gegebenfalls der Name geändert. Für jeden Level-Namen für den kein Mapping existiert, wird geprüft, ob ein Alias für die Level-Namen besteht. Wenn ja, wird das Alias als neuer Level-Name verwendet.

4.2. Überprüfen der Level-Hierarchie des Objekts im globalen Schema

Hierbei wird die Level-Hierarchie des globalen Objekts angepasst. Dafür wird zuerst geprüft, ob alle Levels der aus dem vorherigen Schritt aktualisierten lokalen Level-Hierarchie auch im globalen Schema existieren. Anschließend wird geprüft ob es im globalen Schema zusätzliche Levels gibt. Wenn Änderungen an der Level-Hierarchie nötig sind, werden die Änderungen in diesem Schritt durchgeführt.

4.3. Anpassen der Level-Hierarchie

Nachdem eine korrekte Level-Hierarchie auf Basis des lokalen und des globalen Schemas erstellt wurde, wird diese dem bereits in der globalen Dimension existierenden Objekt zugewiesen. Zusätzlich gehören daraufhin auch die Level Positionen und der Level Hierarchie Cache neu berechnet. Die Level Positionen wurden von [Schütz 2011] eingeführt um die Reihenfolge der Levels in der Level-Hierarchie zu spezifizieren. Die Aktualisierung der Level Positionen und des Level Hierarchie Caches erfolgt allerdings in der aktuellen Version noch nicht und ist somit in zukünftigen Versionen zu ergänzen.

4.4. Hinzufügen der Attribute des neuen Objekts

Hierfür wird wieder die Methode *add_attributes* aufgerufen, welche prüft, ob das Objekt im lokalen Schema ein oder mehrere Attribute definiert. Wenn ja, werden

der Attributname und der Level, für den es definiert wurde, an das globale Schema angepasst, d.h. es wird geprüft, ob ein Mapping für den Attributnamen bzw. den Level, auf den das Attribut eingeführt wurde, existiert. Anschließend wird der Datentyp des Objekts herausgesucht und das Attribut, sofern es in der globalen Dimension noch nicht existiert, dem neu in die globale Dimension eingefügten Objekt hinzugefügt.

4.5. Hinzufügen von Mappings

Dieser Punkt wird wiederum nur für das Wurzelobjekt der lokalen Dimension ausgeführt. Dabei werden die Aliase, die der Methode *import_branch* mitgegeben wurden, in Mappings übersetzt. Der genaue Ablauf dabei wird in Kapitel 5.4 beschrieben.

4.6. Speichern der Änderungen der globalen Dimension

5.2.2. Setzen der Attributwerte

Attribute werden zwar gleich beim Einfügen eines Objekts in der globalen Dimension hinzugefügt, die Werte der Attribute werden jedoch erst gesetzt, nachdem alle M-Objects importiert wurden, da dies vorher nicht möglich ist. Der Grund hierfür ist, dass ein Objekt Attribute für niedrigere Levels definieren kann, diesen Attributen allerdings erst bei Objekten auf den entsprechenden Level die Werte zugewiesen werden. Da die Elternobjekte eines einzufügenden Objekts bereits im globalen Schema existieren müssen, werden die Objekte jedoch geordnet ins globale Schema importiert. Es werden daher zuerst alle Objekte des Top-Levels der Dimension importiert, danach alle des Second-Top-Levels usw. Da aber Objekte Attribute auch für niedrigere Levels definieren können, existieren die jeweiligen Objekte, für die die Werte dann tatsächlich gesetzt werden, beim Hinzufügen des Attributs noch nicht. Es können zu diesem Zeitpunkt also keine Werte für die Objekte gesetzt werden. Deswegen werden die Werte aller Attribute erst nachdem alle Objekte importiert wurden gesetzt. Ein Beispiel für die Problematik ist die zu importierende Dimension *Musik* aus der Problemstellung. Diese definiert im M-Object *Musik* das Attribut *komponist*. Das Attribut selbst wird, gleich nachdem das Objekt in der Produktdimension eingefügt wurde, dem globalen Schema hinzugefügt. Der Wert *Mozart* für das M-Object *RondoAllaTurca* kann aber noch nicht gesetzt werden, da das Objekt *RondoAllaTurca* zu diesem Zeitpunkt noch nicht existiert. Dieses wird erst erstellt, nachdem alle M-Objects der Levels *kat* und *label* importiert wurden.

Um die Attributwerte zu setzen, wird die Methode *set_attribute_values* verwendet. Diese ruft zum Setzen der Werte zuerst für das lokale Wurzelobjekt und anschließend in einer Schleife für jedes Objekt in der zu importierenden Teildimension die Methode *set_values* auf.
Die Methode *set_values* durchläuft zuerst alle Attributtabellen eines Objekts und anschließend alle Attribute der jeweiligen Tabelle. Danach werden der Name des Attributes und des Levels, für das es definiert wurde, an das globale Schema angepasst, d.h. es wird geprüft, ob ein Mapping für den Attributnamen bzw. den Level, auf den das Attribut eingeführt wurde, existiert. Anschließend wird zum Setzen des Wertes die Methode *set_attribute_value* aufgerufen. Diese liest die Attributwerte der lokalen Dimension und prüft, ob es für das Attribut im globalen Schema einen Wert gibt. Hierbei können folgende Fälle auftreten:

- Im globalen Schema hat das Objekt noch keinen Wert für das Attribut definiert:
  In diesem Fall werden der Name des Objekts für das der Attributwert gesetzt werden soll und der Wert des Attributs selbst einfach einer Liste mit den Attributwerten hinzugefügt.
- Im globalen Schema hat das Objekt bereits einen Wert für das Attribut und es gibt ein M-Object-Mapping für das Objekt inklusive dem Attribut mit Priorität „1“:
  Gibt es im globalen Schema für das Attribut des Objekts bereits einen Wert, so werden die Attributprioritäten des M-Object-Mappings durchsucht. Wird ein Mapping mit der Priorität „1“ für das Attribut gefunden, werden der Name des Objekts, für das der Attributwert gesetzt werden soll, und der Wert des Attributs selbst einfach einer Liste mit den Attributwerten hinzugefügt.
- Gibt es im globalen Schema bereits einen Wert für das Attribut des Objekts und es existiert allerdings kein M-Object-Mapping für das Attribut des Objekts bzw. ein M-Object-Mapping mit der Priorität „0“, so wird der lokale Wert nicht gesetzt. Im globalen Schema bleibt also weiterhin der alte Wert bestehen.

Am Ende der Methode *set_attribute_value* wird die Methode *bulk_set_attribute* mit dem Attributnamen und der Liste der Attributwerte aufgerufen um die Werte zu setzen.

5.3. Import des M-Cubes

Der Import des M-Cube erfolgt semi-automatisch durch Ausführen der *import_mcube* Methode. Das heißt, der Benutzer muss, bevor er eine Dimension importieren kann, die erforderlichen Mappings erstellen und anschließend die *import_mcube* Methode aufrufen. Diese befindet sich im Package *mcube* und bekommt den Einfügepunkt im globalen M-Cube, die Wurzel M-Relationship des lokalen Teil-Cubes, der importiert werden soll, und die Kennzahl-Aliase als Parameter übergeben. Der Import selbst erfolgt anschließend automatisch.
Ebenso wie bei den Dimensionen ist auch hier in zukünftigen Versionen vorgesehen, dass der Benutzer bei Konflikten eine Meldung bekommt und korrigierend eingreifen kann. Derzeit ist dies jedoch nicht möglich, da die Implementierung rein in PL/SQL erfolgt ist und PL/SQL nicht interaktiv ist. Aktuell wird also lediglich eine Warnung ausgegeben, der Integrationsprozess selbst läuft jedoch weiter, wenn kein gravierender Fehler in Form einer unvorhergesehenen/unerwarteten Exception auftritt.
Die Methode *import_mcube* ist dynamisch generiert, da die konkreten Typen des lokalen Cubes, der lokalen M-Relationships etc. erst zur Laufzeit bekannt sind. Der Super-Typ kann nicht verwendet werden, weil es für die Koordinaten und die Connection-Levels im erweiterten Prototyp von [Schütz 2010] keinen allgemeinen Super-Typ gibt. Die Methode *import_mcube* befindet sich im Package *mcube*. Sie ist also ebenso wie die *import_branch* Methode nicht im konkretisierten, Cube-spezifischen Typ. Dadurch können auch hier keine konkreten Typen verwendet werden und die Objekttypen werden nicht dynamisch zur Laufzeit geprüft. Die Verwendung der konkreten Typen würde also die Vorteile mit sich bringen, dass Cube-spezifische Typen verwendet werden könnten. Es könnten jedoch, auch wenn die *import_mcube* Methode im konkretisierten Typ realisiert wird, nur für den globalen Cube die konkreten Typen verwendet werden. Jene des lokalen Typs wären wiederum unbekannt. Nichts desto trotz wäre diese Variante für den Benutzer semantisch verständlicher. Es gilt also zu überdenken, ob die Import Funktionalität für M-Cubes in zukünftigen Versionen nicht im konkretisierenden Typ realisiert werden sollte.

Der folgende Abschnitt beschreibt den Algorithmus beim Importieren eines Cubes abstrakt. Der Algorithmus besteht grundsätzlich aus fünf Schritten:

- Dem Prüfen, ob der globale und der lokale Cube dieselben Dimensionen verwenden
- Dem Import der M-Relationships

- Dem Hinzufügen der Kennzahlen
- Dem Setzen der Werte für die Kennzahlen
- Dem Hinzufügen der Mappings

Prüfen der Dimensionen:

Im ersten Schritt wird lediglich geprüft, ob der globale und der lokale Cube dieselben Dimensionen in derselben Reihenfolge haben, also ob die beiden Cubes jeweils dieselbe Dimension an derselben Stelle besitzen. Dies wird geprüft, da die einzelnen Koordinaten des lokalen Cubes nicht an das globale Schema angepasst werden müssen, wenn die beiden Cubes dieselben Dimensionen in derselben Reihenfolge aufweisen. Wenn die einzelnen M-Objects der lokalen Koordinate gleich im globalen Schema übernommen werden können, bringt dies einen erheblichen Performancegewinn mit sich.

Importieren der M-Relationships:

Die einzelnen M-Relationships werden mit der Methode *import_mrelations* importiert. Diese durchläuft im ersten Schritt mit einem PL/SQL Cursor alle lokalen M-Relationships und prüft, ob diese im zu importierenden Teil-Cube liegen oder nicht. Wenn sie darin liegen wird die jeweilige M-Relationship importiert, anderenfalls nicht. Danach holt sich der Algorithmus die M-Objects der Koordinate der lokalen M-Relationship und bildet daraus eine Koordinate im globalen Cube, wenn der globale und der lokale Cube dieselben Dimensionen verwenden. Ansonsten werden die M-Objects der Koordinate an das globale Schema angepasst. Dabei wird ihre Reihenfolge richtig gestellt, eventuell einzelne M-Objects ausgetauscht (wenn der lokale Cube eine andere Dimension verwendet) oder gar zusätzliche M-Objects zur Koordinate hinzugefügt oder einzelne gestrichen, wenn der lokale Cube mehr oder weniger Dimensionen hat als jener des globalen Schemas.
Anschließend wird geprüft, ob die Koordinate im globalen Cube bereits existiert. Falls dies der Fall ist, wird ihr ein M-Relationship-Mapping mit einer Referenz auf die M-Relationship im lokalen Cube hinzugefügt. Andernfalls werden die Koordinate und die Referenz der lokalen M-Relationship gespeichert.
Wenn alle M-Relationships durchlaufen sind, werden alle zuvor gespeicherten, also alle, die noch nicht im globalen Cube existieren, mit der Methode *bulk_create_mrel* angelegt. Zuletzt werden in einer FORALL Schleife die M-Relationship-Mappings gesetzt. Eine FORALL Schleife verwendet ein bulk-bind Konzept, um mehrere SQL Statements auf einmal abzusetzen. Die SQL Statements werden zwar trotzdem nacheinander abgesetzt, jedoch wird

der SQL Engine gleich zu Beginn eine Collection mit allen Werten mitgegeben und es muss somit nicht nach jedem Statement zur PL/SQL Engine zurückgewechselt werden. Dies bringt einen Performancegewinn mit sich. Allgemein ist eine FORALL Schleife folgendermaßen aufgebaut:

```
FORALL Index IN Untergrenze..Obergrenze
    SQL_Statement
```

Hinzufügen der Kennzahlen:

Zum Hinzufügen der Kennzahlen wird die Methode *add_measures* verwendet. Dabei wird zunächst über alle M-Relationships des lokalen Cubes iteriert, die Fakttabellen definieren. Danach werden die einzelnen Fakttabellen importiert wenn die M-Relationship für die sie definiert wurden innerhalb des zu importierenden Teil-Cubes liegt. Anschließend wird für die M-Relationship, die das Fakt bzw. die Kennzahl definiert, wiederum die Koordinate an das globale Schema angepasst.

Wenn die M-Relationship aus der an das globale Schema angepassten Koordinate gebildet wurde, wird über die einzelnen Fakttabellen selbst und in weiterer Folge über die einzelnen Kennzahlen der Fakttabelle iteriert. Anschließend holt sich der Algorithmus die einzelnen Mappings und passt den Namen der Kennzahl anhand der Measure-, der Pivot- und der Unpivot-Mappings an das globale Schema an. Analog zu den Attributen wird auch hier, wenn kein Mapping gefunden wurde, geprüft, ob es ein Alias für die Kennzahl gibt.

Im nächsten Schritt wird der Connection Level, auf dem die Kennzahl definiert wurde, geholt und an das globale Schema angepasst. Dabei kann analog zum Anpassen der Koordinaten die Reihenfolge der einzelnen Level richtig gestellt werden, einzelne Level ausgetauscht (wenn der lokale Cube eine andere Dimension verwendet) oder gar einige Level hinzugefügt oder gestrichen werden, wenn der lokale Cube mehr oder weniger Dimensionen hat als jener des globalen Schemas.

Wenn der Name und der Connection Level der Kennzahl bekannt sind, wird geprüft, ob sie bereits im globalen Cube existiert. Wenn nicht, wird die Kennzahl mit der Methode *add_measure* einfach hinzugefügt. Ansonsten werden der Connection Level der Kennzahl im globalen Schema und jener des lokalen Schemas verglichen. Ist die Kennzahl lokal auf einem konkreteren Connection Level, also wenn sie lokal detaillierter ist, wird sie im globalen Cube auf den konkreteren Connection Level angehoben, andernfalls nicht. Man beachte, dass die detaillierteren Werte der Kennzahl später zusätzlich hinzugefügt werden, die alte Kennzahl des globalen Cubes mitsamt seinen Werten bleibt jedoch weiterhin bestehen.

Im nächsten Schritt wird der Name der Fakttabelle temporär in einer Liste gespeichert. Zuletzt wird die Liste mit den Fakttabellen des zu importierenden Teil-Cubes durchlaufen und für jede die Methode *set_measure_values* zum Setzen der Werte der Kennzahlen aufgerufen.

Setzen der Werte der Kennzahlen:

Die Werte einer Kennzahl werden mit der Methode *set_measure_values* gesetzt. Diese wird für eine spezielle Fakttabelle aufgerufen und durchläuft alle Fakten beziehungsweise Kennzahlen der Tabelle.
Anschließend wird wieder die Koordinate der M-Relationship, für die die Kennzahl definiert wurde, an das globale Schema angepasst und die M-Relationship im globalen Cube gesucht.
Nachdem die M-Relationship mit Hilfe der an das globale Schema angepassten Koordinate gebildet wurde, werden die Mappings gelesen und der Name der Kennzahl anhand der Measure-, der Pivot- und der Unpivot-Mappings an das globale Schema angepasst, d.h. es wird geprüft, ob ein Mapping für den Namen der Kennzahl existiert. Auch hier wird, wenn kein Mapping gefunden wurde, geprüft, ob es ein Alias für die Kennzahl gibt und der Name der Kennzahl gegebenfalls geändert.
Im nächsten Schritt wird geprüft, ob für die Kennzahl an der entsprechenden Koordinate im globalen Schema bereits ein Wert besteht. Wenn nicht, wird der lokale Wert einfach mit der Methode *set_measure* gesetzt. Ansonsten wird geprüft, ob es für die M-Relationship ein M-Relationship-Mapping mit der Priorität „1“ für die Kennzahl gibt oder nicht. Gibt es ein Mapping mit der Priorität „1“, wird der globale Wert mit dem lokalen überschrieben. Andern falls wird der globale Wert beibehalten.

Hinzufügen der Mappings:

Im letzen Schritt wird die Methode *add_measure_mappings* aufgerufen. Diese erstellt für jedes Kennzahl-Alias ein Mapping. Dabei werden die einzelnen Aliase durchlaufen und ein Mapping mit der Methode *add_measure_mapping* hinzugefügt.

Allgemein anzumerken ist außerdem, dass der Import des M-Cubes überwiegend in dynamischen PL/SQL Methoden implementiert wurde, da jeder M-Cube eigene Typen für die Koordinaten, M-Relationships etc. hat und deshalb der Typ der einzelnen Variablen erst zur Laufzeit bekannt ist. Ein allgemeiner Typ konnte nicht angegeben werden, da es einen solchen nicht überall gibt. Zum Beispiel hat jeder Cube für seine Koordinaten einen eigenen

Typ, es gibt aber keinen Basis Typ für die Koordinaten, da diese für keine generischen Attribute und Methoden verwendet werden.
Der Nachteil dieser Methode ist, dass einzelne Code Segmente, wie zum Beispiel das Anpassen der Koordinaten an das globale Schema, mehrmals sehr ähnlich vorkommen, und nur Unterschiede in den Datentypen aufweisen. Aufgrund der unbekannten Datentypen und der nicht bekannten Anzahl der Dimensionen eines Cubes, wurde hierfür aber noch keine bessere Lösung gefunden. Die Problematik ist jedoch bekannt und soll in zukünftigen Versionen behoben werden.

5.4. Automatisch abgeleitete Mappings

In den bisherigen Abschnitten wurden stets nur jene Mappings betrachtet, die vom Benutzer zu erstellen sind, es gibt jedoch auch Mapping-Typen, die automatisch erstellt werden. Um welche Mappings es sich dabei handelt und wie beziehungsweise wo diese erstellt werden, wird in diesem Kapitel beschrieben.

5.4.1. M-Object-Mappings

Wie bereits in den Kapiteln 4.3.1.3 und 4.7.4 beschrieben, können M-Object-Mappings für M-Objects erzeugt werden. Diese M-Object-Mappings enthalten eine Referenz auf das lokale Objekt und eine Tabelle mit Attributen und den Prioritäten der Attributwerte, und werden benötigt, um später wieder einen Bezug zwischen dem Objekt im globalen und im lokalen Schema herstellen zu können.
M-Object Mappings werden allerdings auch teilweise automatisch abgeleitet. Beim Importieren wird für jedes Objekt, das importiert werden soll, geprüft, ob es bereits ein vom Benutzer definiertes M-Object-Mapping hat, ansonsten wird eines automatisch erstellt, um später wieder einen Bezug zwischen den importierten M-Object im globalen und dem Originalobjekt im lokalen Schema herstellen zu können. Wird ein Same-As Mapping automatisch erstellt, so wird ihr für die Attributprioritäten nur eine leere Tabelle mitgegeben. Der zugehörige Code ist unten abgebildet:

```
IF (global_obj.has_same_as_mapping(local_obj_ref) = false) Then
    global_obj.add_same_as_mapping(local_obj_ref, priority_tty());
End IF;
```

5.4.2. M-Relationship-Mappings

M-Relationship-Mappings enthalten analog zu den M-Object-Mappings eine Referenz auf die M-Relationship im lokalen Cube und eine Tabelle mit ihren Kennzahlen und den Prioritäten der Werte der Kennzahlen. Für jede M-Relationship, die importiert werden soll und die global bereits existiert, wird wieder geprüft, ob sie bereits ein M-Relationship-Mapping hat, andernfalls wird eines mit einer leeren Tabelle für die Kennzahl Prioritäten erstellt:

```
IF (mrel.has_same_as_mapping(local_mrel_ref) = false) Then
    mrel.add_same_as_mapping(local_mrel_ref, mrel_priority_tty());
End IF;
```

Für alle M-Relationships, die im globalen Cube erst angelegt werden müssen, werden die Same-As Mappings gesammelt am Ende der *import_mrelations* Methode erstellt. Um die Methode effizient zu gestalten, werden die M-Relationship Mappings mit Hilfe einer FORALL-Schleife gesetzt:

```
FORALL i IN 1..coordinates.Count
    Insert Into Table(Select rt.same_mrels From <global_mcube.mrel_table>
      rt Where rt.coordinate.equals(coordinates(i)) = 1) Values(mrels(i));
```

5.4.3. Mappings aus Parametern der Import Methode

Mappings können, egal ob für Levels, Attribute, Objekte oder Kennzahlen, nur erstellt werden, wenn das jeweilige Level, Attribut, Objekt bzw. die jeweilige Kennzahl auch im globalen Schema existiert. Wenn nicht, muss der Import Methode ein Alias mitgegeben werden.

M-Object-Aliase

Wenn ein M-Object unter einem anderen Namen ins globale Schema eingefügt werden soll, so ist dafür ein M-Object-Alias zu erstellen und dieses der *import_branch* Methode mitzugeben. Das Objekt wird dann automatisch von der *import_mobject* Methode umbenannt, indem ein neuer Name für das Objekt erzeugt und das Objekt unter diesem neuen Namen ins globale Schema eingefügt wird. Für M-Object Aliase wird allerdings kein Mapping erstellt, da ohnehin über das M-Object-Mapping ein Bezug zwischen dem globalen und dem lokalen Objekt hergestellt werden kann.

Level-Aliase

Für Level, die in der lokalen Dimension eingeführt werden oder für lokal zusätzliche Levels, sind vom Benutzer Level-Aliase zu erstellen, wenn diese unter einem anderen Namen in die globale Dimension integriert werden sollen. Beim Importieren des lokalen Wurzelobjekts wird automatisch die Methode *add_mappings* aufgerufen. Diese erstellt für alle Level- und Attribut-Aliase ein Mapping, wenn vorher kein Mapping existiert hat.

Attribut-Aliase

Analog zu den Levels sind für Attribute, die in der lokalen Dimension eingeführt werden oder für lokal zusätzliche Attribute vom Benutzer Attribut-Aliase zu erstellen und der *import_branch* Methode mitzugeben, wenn diese unter einem anderen Namen in die globale Dimension integriert werden sollen. Für die Attribut-Aliase wird ebenfalls in der Methode *add_mappings* ein Mapping erstellt, wenn vorher kein Mapping existiert hat.

Kennzahl-Aliase

Wie bei den Levels und Attributen beim Dimensionsimport, kann auch beim M-Cube-Import für Kennzahlen, die im globalen Cube noch nicht existieren, kein Mapping erstellt werden. Wenn die Kennzahl unter einem anderen Namen in den globalen Cube integriert werden sollen, muss der Benutzer ein Kennzahl-Alias erstellen und der *import_mcube* Methode mit geben. Diese ruft, nachdem der lokale Cube importiert wurde die Methode *add_measure_mappings* auf, welche automatisch für alle Kennzahl-Aliase ein Mapping erstellt.

5.4.4. Sonstige Mappings

Wie bereits in Kapitel 4.2 angeführt, müssen grundsätzlich für alle Levels, Attribute usw., die sowohl im globalen als auch im lokalen Schema vorkommen, Mappings erstellt werden, da Namensgleichheit nicht gleich Äquivalenz impliziert. Um den Benutzer zu unterstützen, wird jedoch bei Namensgleichheit zwischen den globalen und den lokalen Schema für die einzelnen Attribute, Levels usw. automatisch ein Mapping erstellt, sofern vom Benutzer für das Attribut, Level usw. kein Mapping oder Alias definiert wurde. Die automatisch generierten Mappings werden ebenfalls mit der Methode add_mappings hinzugefügt.

5.5. Anpassen der Koordinaten

Grundsätzlich hat jede M-Relationship eines M-Cubes eine Koordinate, welche die Position innerhalb des Cubes spezifiziert. Diese Koordinate besteht wiederum aus einer Reihe von M-Objects. Beispielsweise gibt es in der Problemstellung in Kapitel 4.1 im globalen Data Warehouse eine M-Relationship mit der Koordinate März 2010 – Salzburg – Porsche 911 Carrera S, für welche wiederum eine Kennzahl *revenue* mit dem Wert 360000 definiert ist.
Wenn nun der globale und der lokale M-Cube verschiedene Dimensionen verwenden oder die Dimensionen eine unterschiedliche Reihenfolge haben, müssen die Koordinaten der M-Relationships des lokalen Cubes an das globale Schema angepasst werden, bevor die M-Relationships in den globalen M-Cube integriert werden können. Die Koordinaten werden vom System automatisch angepasst. Das heißt, es werden die M-Objects der Koordinate an das globale Schema angepasst. Dabei wird ihre Reihenfolge richtig gestellt, eventuell einzelne M-Objects ausgetauscht (wenn der lokale Cube eine andere Dimension verwendet) oder gar zusätzliche M-Objects zur Koordinate hinzugefügt oder einzelne gestrichen, wenn der lokale Cube mehr oder weniger Dimensionen hat als jener des globalen Schemas. Es müssen daher keine Mappings vom Benutzer erstellt werden.

5.6. Hinzufügen von Levels zum globalen Schema

Einer speziellen Betrachtung bedarf es auch, wenn Dimensionen mit zusätzlichen Levels importiert werden, da in diesem Fall das Schema der globalen Dimension angepasst werden muss. Standardmäßig wird beim Importieren die Methode *add_lvl_to_global_schema* aufgerufen. Diese prüft, ob der Top-Level des zu importierenden M-Object im globalen Schema bereits existiert oder nicht. Wenn der Top-Level erstellt werden muss, wird folgendermaßen vorgegangen:

1. Aktualisieren der globalen Level-Hierarchie

   Im ersten Schritt wird der neue Level in die Level-Hierarchie der globalen Dimension eingefügt. Anschließend wird die Level-Hierarchie für alle Objekte in der betroffenen Teildimension ebenfalls aktualisiert. Für das Beispiel aus Kapitel 4.7.1 würde dies bedeuten, dass beim Importieren des Bundeslandes Oberösterreich die Methode *add_lvl_to_global_schema* das neue Level *bundesland* hinzufügen würde. Daraufhin würde die Level-Hierarchie der Dimension *Location* und die Level-Hierarchie aller Objekte in der Dimension, die direkte oder transitive Nachfolger des M-Objects *Österreich* sind, aktualisiert werden.

2. Einfügen eines ‘<newLevel>_Unknown‘ Objektes

   Um die Level-Hierarchie konsistent zu halten, müssen die Elternobjekte aller M-Objects des Levels unter dem neu eingefügten Level aktualisiert werden. Wenn es zum Beispiel neben der Stadt Salzburg in der globalen Dimension auch eine Stadt Graz gegeben hätte, dürfte dieses also nicht mehr *Austria* als Elternobjekt haben, da sich der Level Stadt nun nicht mehr an zweithöchster Stelle in der Level-Hierarchie von Österreich befindet. Daher muss ein Objekt *Austria_state_unknown* erstellt und der Stadt Graz als Elternobjekt zugewiesen werden. *Austria_state_unknown* selbst bekommt *Austria* als Elternobjekt. In diesem Schritt wird also ein Objekt erstellt, dass in weiterer Folge den Objekten des weiter unten folgenden Levels als Parent zugewiesen wird. Der Name des neu erstellten Objekts setzt sich aus dem Elternobjekt des neuen Levels (*Austria*), dem neuen Level (*bundesland* bzw. *state*) und dem Suffix *unknown* zusammen. Wenn der Name länger als 30 Byte ist, wird er mit der Methode *get_unique_short_name* im Package *identifiers* umbenannt.

   Zum Einfügen des ‘<newLevel>_Unknown‘ Objektes wird die Methode *insert_missing_level_obj* verwendet. Falls die lokale Dimension mehrere zusätzliche Levels hat, also zum Beispiel jede Stadt einem Bezirk zugeordnet ist, jeder Bezirk einem Bundesland und jedes Bundesland einem Land, dann wird die Methode *insert_missing_level_obj* rekursiv aufgerufen, abermals die Level-Hierarchie aktualisiert und ein ‘<newLevel>_Unknown‘ Objekt in die globale Dimension eingefügt.

3. Aktualisieren der Elternobjekte

   Im dritten Schritt werden für alle Objekte der globalen Dimension, deren Top-Level sich direkt unter dem neuen Level befindet, die Elternobjekte aktualisiert. Beispielsweise würde für alle Städte aus Österreich das Land Österreich durch das Objekt Austria_state_unknown ersetzt werden.

5.7. Hilfsmethoden

Für den Dimensionsimport gibt es zusätzlich ein Package *Import_auxili*ary, welches drei Hilfsmethoden beinhaltet, die mehrmals benötigt werden:

- get_name_alias
- get_global_level_name
- get_local_level_name

Die Methode *get_name_alias* wird benötigt, um zu prüfen, ob für einen Namen der lokalen Dimension ein Alias definiert wurde oder nicht. Die Methode nimmt den lokalen Namen und ein Feld von Aliasen als Parameter und gibt das Alias zurück. Wenn kein Alias gefunden wird, wird einfach der lokale Name zurückgegeben. Die Methode kann sowohl für Level- als auch für Attribut und M-Object Aliase verwendet werden.

Get_global_level_name wird benötigt, um für einen Level der lokalen Dimension den Namen für die globale Dimension herauszufinden. Dazu durchsucht die Methode die Mappings des globalen Schemas und ändert den Namen, wenn ein Mapping gefunden wurde. Andernfalls wird geprüft, ob ein Alias für das Level besteht. Wenn ja, wird der Level-Name für das globale Schema geändert, ansonsten wird derselbe Level-Name wie in der lokalen Dimension verwendet.

Die Funktion *get_local_level_name* ist die Umkehrung von *get_global_level_name*. Hier wird für einen Level des globalen Schemas der Name des lokalen Schemas herausgesucht und zurückgegeben.

6. Ergebnisse

Die im Zuge dieser Arbeit entstandene Prototyp-Implementierung ist in der Lage, einen Data Mart mit hetero-homogenem Schema in ein globales Data Warehouse zu integrieren unter der Voraussetzung, dass der lokale Data Mart dasselbe Datenmodell verwendet. Dabei wird für die Speicherung des Data Warehouses und der Data Marts der Prototyp eines Data Warehouse Management Systems in einer objekt-relationalen Oracle Datenbank verwendet, welcher von [Schütz 2010] erstellt wurde. Da die verwendeten Data Marts autonom sein können, werden zur Abbildung von Schema-Heterogenitäten eine Reihe verschiedener Mapping-Typen, wie etwa das Level-, das Attribut- oder das M-Object-Mapping, zur Verfügung gestellt.
Der Integrationsprozess zum Import des Data Marts ist semi-automatisch und besteht aus drei Schritten: der Definition der Mappings, dem Import der Dimensionen und dem Import des M-Cubes. Im ersten Schritt muss der Benutzer für alle Schema-Heterogenitäten und für alle ähnlichen beziehungsweise widersprüchlichen Daten Mappings erstellen. Die dafür zur Verfügung stehenden Mappings wurden in Kapitel 4.3.1 vorgestellt. Im zweiten Schritt werden die Dimensionen importiert und im letzten Schritt kann der M-Cube selbst importiert werden. Der Import der Dimensionen bzw. des M-Cubes erfolgt automatisch, nachdem die jeweiligen Methoden aufgerufen worden sind.

Der in dieser Arbeit präsentierte Prototyp ist in PL/SQL implementiert. Er ist in der Lage, die in Kapitel 1.1 angeführten, Schema-Heterogenitäten zu behandeln. Es gibt jedoch, da die Implementierung rein in PL/SQL erfolgte, keine grafische Benutzerschnittstelle. Außerdem gibt es auch ein paar Einschränkungen in Bezug auf das Schema des lokalen Data Marts, wie zum Beispiel, dass die Level-Hierarchien des globalen und des lokalen Schemas konsistent sein müssen (vgl. Kapitel 4.9). Weiters müssen für alle Schema-Heterogenitäten korrekte Mappings definiert werden. Eine Liste der Einschränkungen befindet sich in Kapitel 4.9. Neben den Einschränkungen gibt es auch ein paar Punkte, die im derzeitigen System noch nicht realisiert, sondern erst in zukünftigen Versionen vorgesehen sind:

- Unterschiedliche Einheiten:
  Im derzeitigen System ist es zwar möglich, unterschiedliche Einheiten für Attribute zu definieren, jedoch werden diese noch nicht berücksichtigt. Beispielsweise können die Preise im globalen Data Warehouse in Euro angegeben werden, im lokalen Data Mart hingegen in Dollar oder Schweizer Franken. Die in Form von Metadaten hinterlegte

Währung wird zwar ins globale Schema importiert, allerdings in weiterer Folge nicht berücksichtigt.

- Fehler-/Konfliktbehandlung:

  Grundsätzlich wurde versucht, gegebenfalls anfallende Fehler abzufangen und die Prototyp Implementierung robust zu gestalten, jedoch können Fehler nie gänzlich ausgeschlossen werden. Wenn im Zuge des Imports dennoch ein gravierender Fehler in Form einer unvorhergesehenen/unerwarteten Exception auftritt, führt dies derzeit zum Abbruch des Imports. In zukünftigen Versionen sollte der Benutzer jedoch durch eine Benutzerschnittstelle die Möglichkeit haben, im Fehlerfall einzugreifen. Dasselbe gilt bei Namenskonflikten. Wenn also beispielsweise ein M-Object-Mapping vergessen wird anzulegen, wird das M-Object derzeit einfach umbenannt und unter einem anderen Namen eingefügt. Wenn ein M-Relationship-Mapping fehlt, wird ein Fehler geworfen. In Folgeversionen sollte der Benutzer auch hier über eine Benutzerschnittstelle die Möglichkeit haben einzugreifen und den Konflikt zu beheben.

- Logging Mechanismus:

  Derzeit gibt es keinen Logging Mechanismus oder dergleichen. Falls ein Konflikt auftritt und beispielsweise ein M-Object umbenannt wird, weil ein M-Object-Mapping fehlt, wird bloß eine Meldung mit *dbms_output* von PL/SQL geworfen. Warnungen oder zusätzliche Information über den Verlauf des Integrationsprozesses werden allerdings nicht persistiert oder gar archiviert. In späteren Versionen sollten hierfür Fehler geworfen und die entsprechenden Meldungen in einer eigenen Log Tabelle abgespeichert werden.

- Import zusätzlicher Dimensionen:

  Der Import von zusätzlichen Dimensionen, die im globalen Data Warehouse gar nicht existieren, wurde als trivial erachtet, da es sich dabei um ein einfaches Kopieren der Daten handelt. Die Prototyp Implementierung stellt deshalb hierfür keine Funktionalität zur Verfügung.

- Performance-Optimierung:

  Im derzeitigen System ist keine Optimierung hinsichtlich der Performance erfolgt, da diese bei kleinen Datenmengen nicht zwingend nötig ist, und eine Performance-Optimierung den Umfang dieser Arbeit sprengen würde. Dennoch sollten in zukünftigen Versionen Performance-Messungen erfolgen und Teile, die eine schlechte Performance aufweisen, optimiert werden, da der Import bei größeren Datenmengen

sehr lange dauern kann, insbesondere wenn sehr viele Schema-Heterogenitäten bestehen.

- Mappings:

  Derzeit werden sämtliche Mappings, die den Dimensionsimport betreffen, beim Wurzelobjekt in der globalen Dimension und sämtliche, die bei M-Relationships gespeichert werden, bei der Wurzel M-Relationship im globalen Cube gespeichert. Um den Prinzipien der Objektorientierung zu entsprechen, müssten jedoch alle Mappings von sämtlichen Vorgängern eines M-Objects bzw. einer M-Relationship berücksichtigt werden und nicht nur jene, die bei der Wurzel gespeichert sind. Für die Funktionsweise des Integrationsprozesses stellt dies kein Problem dar, solange die betroffenen Mappings bei der Wurzel gespeichert werden. Dennoch sollte auch dies in zukünftigen Versionen noch überarbeitet werden.

Literaturverzeichnis

[Bauer & Günzel 2009] Bauer A., Günzel H., „Data Warehouse Systeme: Architektur, Entwicklung, Anwendung“, 4. Auflage, dpunkt.verlag, 2009

[Berger 2009] Berger S., „FedDW: a Model-Driven Approach for Querying Federations of Autonomous Data Marts“, Dissertation, Institut für Wirtschaftsinformatik - Data & Knowledge Engineering, Johannes Kepler Universität Linz, 2009

[Breslin 2004] Breslin M., “Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models.”, Business Intelligence Journal, 2004

[Calì et al. 2004] Calì A., Lembo D., Rosati R., Ruzzi M., “Experimenting Data Integration with DIS@DIS”, In Lecture Notes in Computer Science, Vol.3084, S. 653-673, 2004

[Chamoni & Gluchowski 2006] Chamoni P., Gluchowski P., „Analytische Informationssysteme. Business Intelligence–Technologien und –Anwendungen”, Springer Verlag GmbH, Berlin Heidelberg, 2006

[Codd 1990] Codd E.F., „The relational model for database management: version 2”, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990

[Codd 1993] Codd E.F., “Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandat”, E.F. Codd & Associates, White-Paper, 1993

[Goeken 2006] Goeken M., „Data-Warehouse-Systeme”, Deutscher Universitäts-Verlag, Wiesbaden, 2006

[Golfarelli et al. 1998] Golfarelli M., Maio D., Rizzi S., “The dimensional fact model: A conceptual model for data warehouses”, International Journal of Cooperative Information Systems, Vol. 7, Nr. 2-3, S. 215-247, 1998

[Inmon 2005] Inmon W.H., „Building the Data Warehouse. 4.Auflage”, Wiley, 2005

[Kemper & Eickler 2006] Kemper A., Eickler A., „Datenbanksysteme“, Oldenbourg, München, 2006

[Kimball & Ross 2004] Kimball R., Ross M., “The Data Warehouse Toolkit: The Complete Guide to Dimensional Modelling”, 3.Auflage, John Wiley & Sons, New York, 2004

[Koch 2001] Koch C., “Data integration against multiple evolving autonomous schemata”, Dissertation, TU Wien, Wien, Österreich, 2001

[Lange 2006] Lange N., “Geoinformatik in Theorie und Praxis”, Springer, Berlin Heidelberg, 2006

[Lenzereni 2002] Lenzerini M., “Data integration: a theoretical perspective,” In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, S. 233-246, 2002

[Levy 2000] Levy A.Y., “Logic-based techniques in data integration,” KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE, S. 575-595, 2000

[Lufter 1999] Lufter J., „Objektrelationale Datenbanksysteme“, Informatik-Spektrum, Vol.22, Nr.4, S. 288-290, 1999

[Marx Gómez et al. 2009] Marx Gómez J., Rautenstrauch C., Cissek P., „Data Warehouse-Architektur“, Springer, Berlin Heidelberg, 2009

[Miller et al. 2001] Miller R.J., Hernandez M.A., Haas L.M., Yan L. Howard Ho C.T., Fagin R., Popa L., “The Clio project: managing heterogeneity,” ACM Sigmod Record, Vol.30, Nr.1, S. 78-83, 2001

[Navrade 2007] Navrade F., „Strategische Planung mit Data-Warehouse-Systemen“, Gabler, Wiesbaden, 2008

[Neumayr et al. 2010] Neumayr B., Schrefl M., Thalheim B., “Hetero-Homogeneous Hierarchies in Data Warehouses APCCM '10: Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling, Vol.110, S. 61-70, 2010

[Oracle 2010] Oracle Corporation, “Oracle Database Object-Relational Developer's Guide, 11g Release 2 (11.2)”, 2010

[Rautenstrauch 2004] Rautenstrauch C., „Strategisches Informationsmanagement. Manuskript zur Vorlesung“, Otto-von-Guericke Universität, Magdeburg, 2004

[Sattler & Saake 2004] Sattler K, Saake G, „Data Warehouse Technologien. Manuskript zur Vorlesung“, Otto-vonGuericke Universität, Magdeburg, 2004

[Schmidt-Volkmar 2008] Schmidt-Volkmar P., „Data Warehouse und Data Mart. Betriebswirtschaftliche Analyse auf operationalen Daten“, Gabler, Wiesbaden, 2008

[Schütz 2010] Schütz C., „Extending data warehouses with hetero-homogenous dimension hierarchies and cubes“, Diplomarbeit, Institut für Wirtschaftsinformatik - Data & Knowledge Engineering, Johannes Kepler Universität Linz, 2010

[Schütz 2011] Schütz C., „Extending data warehouses with hetero-homogenous dimension hierarchies and cubes: Improvements on the original proof-of-concept prototype in Oracle“, Internes Arbeitspapier (vorläufige Version), Institut für

Wirtschaftsinformatik - Data & Knowledge Engineering, Johannes Kepler Universität Linz, 2011.
Verfügbar unter http://www.dke.uni-linz.ac.at/research/projects/hh-dw.html

[Stahlknecht & Hasenkamp 2006] Stahlknecht P., Hasenkamp U., „Systementwicklung“, Springer, Berlin Heidelberg, 2006

Informationsmaterial

[HH-DW] Schütz C., „Prototyp eines hetero-homogenen Data Warehouses“, 2011, verfügbar unter: http://www.dke.uni-linz.ac.at/research/projects/hh-dw.html