JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Dissertation zur Erlangung des akademischen Grades
Dr. rer. soc. oec.
im Doktoratsstudium der Sozial- und Wirtschaftswissenschaften

FedDW: a Model-Driven Approach for Querying Federations of Autonomous Data Marts

Angefertigt am

Institut für Wirtschaftsinformatik -
Data & Knowledge Engineering
Johannes Kepler Universität Linz

Eingereicht von

Mag. Stefan Berger

Betreut von

o. Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl
a. Univ.-Prof. Dr. Josef Küng

Linz, Juni 2009

Johannes Kepler Universität Linz, Altenberger Straße 69, 4040 Linz, Austria

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht verwendet und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen deutlich als solche kenntlich gemacht habe.

Linz, Juni 2009

(Stefan Berger)

Danksagung

“Was wir am Nötigsten brauchen, ist ein Mensch, der uns zwingt, das zu tun, was wir können.”

Ralph Waldo Emerson (1803–82), amerikanischer Philosoph und Dichter

---

An dieser Stelle bedanke ich mich bei allen Personen, die in unterschiedlicher Form ihren Anteil am Gelingen dieser Dissertation haben.

Zuerst sei Michael Schrefl erwähnt, dessen schier unerschöpfliches Ideen-Reservoir und fachliches Wissen wesentlich beigetragen haben, die in dieser Dissertation präsentierten Ansätze im Detail auszuarbeiten. Er hatte zu jeder Tages- und Nachtzeit ein offenes Ohr, wenn ich Unterstützung brauchte. In zahlreichen Diskussionen schaffte Michael Schrefl, meine wissenschaftliche Neugier zu fördern, meinen Ehrgeiz anzustacheln, sowie konstruktive Ratschläge zu liefern. Sehr wertvoll war für mich die große Freiheit bei der Erfüllung der wissenschaftlichen Ziele, die mir genügend Raum für persönliche Entfaltung ließ.

Weiters danke ich Josef Küng, der die Zweitbetreuung dieser Arbeit übernommen hat. Seine Rückmeldungen gaben mir zusätzliche Anregungen und Motivation bei der Fertigstellung der Dissertation.

Großer Dank gilt meinen KollegInnen Christian Eichinger, Margit Brandl, Michael Karlinger, Bernd Neumayr, Katharina Grün, Mathias Goller und Günter Preuner, die in den vergangenen Jahren für ein stets positives und angenehmes Arbeitsklima am Institut für Data & Knowledge Engineering gesorgt haben. Ich danke ihnen für die oftmalige Hilfsbereitschaft beim Korrekturlesen von Publikationen sowie für die in vielen Diskussionen eingebrachten Vorschläge, Ideen und frischen Sichtweisen. Weiters danke ich meinen Diplomanden Wolfgang Brunneder, Thomas Rossgatterer, Peter Schweinzer und Lorenz Maislinger, die Teile dieser Dissertation in prototypischen Werkzeugen implementierten.

Meiner Familie bin ich dankbar, dass sie mir das Studium an der Universität ermöglicht und mich finanziell sowie moralisch auf meinem Weg unterstützt haben. Ich danke meinen Freunden, insbesondere Mathias Goller, dass sie für Ausgleich gesorgt und meine “Probleme” in das rechte Licht gerückt haben.

Ein besonderer Dank gebührt Karin, die in den vergangenen Monaten mit bewundernswerter Geduld auf viel gemeinsame Zeit verzichtet hat. Karin gab mir die notwendige Unterstützung und Motivation, indem sie mich stets bestärkt hat, meinen Weg zu beschreiten. Ihr liebevoller Rückhalt war für das Gelingen dieser Arbeit von unschätzbarem Wert.

Kurzfassung

Information ist längst zum wichtigsten Gut des modernen Wirtschaftssystems geworden. Um im internationalen Wettbewerb zu bestehen, sind moderne Unternehmen auf aktuelle und exakte Information als fundierte Grundlage strategischer Entscheidungen angewiesen. Analytische Informationssysteme – Data Warehousing und OLAP Technologien – sind im letzten Jahrzehnt dank immens gesteigerter Rechenleistung und Speicherkapazität von Computern zu Standardtechnologien geworden. Im Falle von Unternehmens-Zusammenschlüssen ist hingegen die Frage zu beantworten, wie die Daten aus bestehenden Data Marts effizient gemeinsam genutzt werden können.

Die Integration von Data Marts führt nicht nur angesichts sehr großer Datenmengen zu neuen Herausforderungen. Zum einen sind Data Marts nach dem multi-dimensionalen Modell entworfen, das eine ausdrucksstärkere Aufbereitung der Daten in OLAP-Anwendungen ermöglicht. Dafür steigt die Wahrscheinlichkeit, dass Fakten und Dimensionen unabhängiger Schemata heterogen sind. Zum anderen enthalten Data Marts oft vertrauliche Daten, auf die kein unbeschränkter Zugriff möglich ist. Deshalb sind bewährte Ansätze zur Integration von Datenbanken für analytische Informationssysteme unzureichend.

Diese Dissertation behandelt “FedDW”, einen modell-basierten, föderierten Ansatz zur Integration von Data Marts auf logischer Schemaebene. FedDW definiert ein globales, multi-dimensionales Schema zwischen autonomen, heterogenen Data Marts. Das globale Schema steht direkt für OLAP-Anfragen der Analysten zur Verfügung. Alle Heterogenitäten zwischen den Data Marts behebt das System transparent mit Hilfe semantischer Mappings. So erweitert sich die Datenbasis für strategische Entscheidungen, ohne dass die Benutzer die heterogenen Schemata der autonomen Data Marts exakt kennen müssen.

Der FedDW-Ansatz bietet zahlreiche Vorteile. Erstens, die Integration logischer Schemata belässt bestehenden Data Mart Systemen volle Autonomie, da alle Daten in den ursprünglichen Systemen bleiben. Zweitens, die Autonomie erleichtert den Schutz sensibler Daten. Drittens konvertiert FedDW alle Daten und Metadaten in ein kanonisches Modell, um Implementierungsplattformen unterschiedlicher Hersteller zu unterstützen. Viertens, FedDW definiert semantische Mappings “von lokal zu global”. Das globale Schema bleibt dadurch stabil, was die Robustheit und Erweiterbarkeit des föderierten Systems begünstigt.

Die Dissertation stellt zwei Prototypen vor, die den FedDW-Ansatz erfolgreich implementieren. “Global Schema Architect” modelliert die semantischen Mappings, basierend auf UML. Das “Query Tool” beantwortet OLAP-Anfragen im globalen Schema direkt aus den autonomen Data Marts.

Abstract

In today's economy, timely access to accurate business information has become an often critical key success factor. Due to rapidly increasing processing power and storage capacity, Data Warehousing and OLAP have emerged to standard technologies in strategic business decision support. Business cooperations, mergers or acquisitions commonly entail the integration of business information among preexisting Data Marts.

The integration of analytical Data Marts poses new challenges for two reasons. First, Data Marts conform to the multi-dimensional model, which increases the expressiveness of data models for business analysts but also causes potentially more heterogeneity. Second, analytical data is often confidential. If privacy policies restrict access to sensitive data, physical integration of Data Marts with well-established techniques is out of question.

The FedDW approach introduced in this thesis provides model-driven design of Data Mart federations. FedDW provides a global "mediated", multi-dimensional schema across the analytical data stores of several autonomous and heterogeneous Data Marts. Thus, FedDW allows strategic analysts to run Business Intelligence applications over larger repositories of data across organizational boundaries, enabling better business decisions.

The advantages of FedDW are manifold. First, FedDW integrates multi-dimensional data at the logical schema level while the underlying Data Marts remain autonomous. Second, the privacy of confidential or sensitive data is ensured by FedDW's conceptual architecture. Every participating organization is entitled to decide which business Data Mart(s) to disclose within the federation. Third, FedDW is system independent because it represents all multi-dimensional schemas, data and the mappings in an internal "canonical" data model. Fourth, FedDW uses source-to-target mappings from autonomous Data Marts to the federated layer. Thus, the global schema remains stable despite possible changes of local Data Mart schemas, and the federation is easier to extend.

This thesis demonstrates the viability of the FedDW Data Mart integration approach with two prototypes. Global Schema Architect supports visual, semi-automatic integration of logical, multi-dimensional Data Mart schemas with a UML-based notation. In turn, FedDW's Query Tool transparently answers user queries against the global schema. The tool ships and reconciles local, heterogeneous Data Mart data according to the semantic matches generated with the Global Schema Architect.

Contents

Chapter 1

Introduction

Contents

This chapter introduces the topics of this thesis in general, motivating the business case for Federated Data Warehouses in Section 1.1. Section 1.2 describes the main challenges addressed and Section 1.3 briefly reviews related work. Based on the objectives stated in Section 1.1, the so-called "FedDW" approach taken in this thesis is briefly explained in Section 1.5. Next, Section 1.6 introduces the prototype implementation of FedDW's tool suite. Finally, Section 1.7 summarizes the overall outline of this thesis.

1.1 Motivation

Data Warehouses (DWs) and Decision Support Systems (DSS) have evolved into indispensable tools for strategic decision makers. In contrast to database systems, a DW is optimized for analytical workload rather than transactional data processing [Bauer and Günzel, 2006, Chaudhuri and Dayal, 1997]. To this end, an organization's DW collects and consolidates the data on all subject areas that are considered helpful for the support of strategic business decisions. Decision makers access the DW data to gain a clearer picture about the organization, which enables better, more well-grounded decisions.

Typically, the Data Warehouse represents the enterprise-wide "single source of truth" and corporate memory of all business process data [Inmon, 2005]. Business processes both produce and consume data. Various *operational systems*—databases, cash registers, inventory management systems, etc.—record data about the states and outcomes of business processes. The DW, in turn, collects and stores all relevant data from those disparate sources. It overcomes existing heterogeneities among the operational data sources during the so-called *ETL process* (i.e., Extraction, Transformation, Loading) that converts all data to the reconciled multi-dimensional schema.

Data Marts (DMs) are more specific repositories of business data designed on top of the DW, often on a coarser level of detail. The usual purpose of a Data Mart is to deliver particular subsets of the DW data to a particular group of users, e.g. the sales division [Kimball, 2002]. Both the Data Warehouse and Data Marts typically conform to the multi-dimensional data model, organizing the items of interest (*"measures"*) in *data cubes*, i.e. within the analysis space along several axes (*"dimensions"*) that represent different business perspectives [Inmon, 2005]. The cube metaphor of the multi-dimensional data model is very illustrative since it is coherent with the analysts' intuitive understanding of business data.

When accessing the DW's data repository, business analysts and decision makers perform aggregations over the multi-dimensional schema in order to compute financial ratios. Typically, these operations—denoted as *On-Line Analytical Processing* (OLAP)—allow to change the perspective on data, drilling up and down the hierarchies of dimensions, and so on [Chaudhuri and Dayal, 1997]. Moreover, so-called Business Intelligence (BI) tools assist the analysts in generating spreadsheet based reports, graph visualizations, etc.

Decision making means setting intentional management actions that change the organizational environment, based on knowledge gained from the DW with BI applications. The decision making process is cyclic since the consequences of every previous decision are reflected in the DW through the regular updates of its information, that in turn will influence future decisions [Thomsen, 2002]. For example, assume that the sales management of a trading company decides to promote a new coffee brand "XY" with a special 20 % discount for two weeks, due to dissatisfying sales figures over the past four weeks' period. The decision will either lead to increased sales of "XY coffee", or the sales manager will ultimately conclude to better remove the product from the assortment.

Nowadays, in answer to high competitiveness of the modern economy, many organizations integrate their businesses. Corporations of any scale commonly merge, acquire others or strategically cooperate with competitors, suppliers and/or customers to obtain strategic advantages. Thus, the number of large to very large private and public organizations is steadily growing.

Corporations may integrate their businesses either horizontally or vertically. *Horizontal integration* is characterized by several independent companies co-operating within some particular market sector. Usually, horizontal business integration aims at achieving economies of scale and/or economies of scope. In contrast, *vertical integration* means that some enterprise cooperates with other organizations along the production chain, but in different economic sectors. As such, vertical integration concerns the upstream suppliers and/or downstream buyers. The goal of vertical business integration is to give each participating organization an edge on information within their respective markets to improve competitiveness of all partners.

Mergers and acquisitions of large corporations often entail the tedious integration of their existing Data Warehouses. If the cooperating organizations share their data, both sides usually benefit from the wider pool of information upon which to base their strategic business decisions. With current technology, however, typical DW integration projects involve the migration of existing systems to the chosen target DW, or even the implementation of a new, integrated DW. Such projects usually take several months or even years to complete [Kimball, 2002].

1.2 Challenges

Timely access to accurate and comprehensive business data is the key factor of success in business integration scenarios. Moreover, efficiency of business processes and accuracy of strategic business decisions are critical to any organization's economic survival. Data Warehouses help large scaled organizations to thoroughly investigate their business processes and locate potential cost savings.

Thus, data integration across autonomous organizations is the necessary prerequisite for any successful business integration. The integration of database systems is the traditional approach that has been researched for several decades [Halevy et al., 2006]. Federated Database Systems [Sheth and Larson, 1990] are the most prominent example of data integration systems.

Integrating Data Warehouses provided by different corporations or public organizations widens the knowledge base for the partners' business analysts, thus enabling better founded strategic decisions. The concrete goals of DW integration projects depend on the organizational background. As far as the private sector is concerned, for example, retail trade companies commonly share the chain stores' sales data with their food suppliers. Consequently, both the retail trader and its food suppliers may forecast the demand more accurately and safe costs. Considering the public sector, health insurance organizations are often divided into autonomous sub-organizations per federal state. Analysis over the data of all sub-organizations enables—among other applications—more effective fraud analysis and better control of treatment costs.

However, the integration of DW schemas and data is tedious and error-prone [Kimball, 2002]. Schema and data integration in the multi-dimensional model is more complex than in the relational model because the multi-dimensional model is semantically richer. It introduces the additional *dimension* entity, whereas the multi-dimensional *fact* entity corresponds with the relation entity of the "traditional" relational data model. In particular, the possibility of organizing dimensions hierarchically along hierarchies of aggregation levels introduces new classes of potential heterogeneity that DW integration must account for.

In the highly competitive modern markets, though, corporations simply cannot afford to wait for several months or years until the integrated Data Warehouse is finally operational. Multi-dimensional schema and data integration is less complex when focused on smaller Data Marts instead of complete Data Warehouses. Using this strategy, the efforts of integration concentrate on a single, relatively small data repository. The scale of problems to solve remains easier manageable.

If autonomous organizations share their analytical data stores, the integration of DMs is better performed on the logical level by establishing a federation. The simplest solution for integrating autonomous DMs—on the physical level, copying the data from the sources to the global DW—is often not feasible due to access restrictions or technical limitations (e.g., network topology). Under such constraints, the advantages of the federated approach are well known from the field of databases. In federated database systems, the dedicated access layer works on top of autonomous data sources to hide heterogeneity (e.g., data models, query languages, and so forth) from the applications and users [Sheth and Larson, 1990, Litwin et al., 1990]. The local databases participating in the federation can be queried with a uniform language. Tightly coupled federated systems additionally provide a global schema expressed in the common, "canonical" data model [Sheth and Larson, 1990].

Besides the obvious advantages of business integration and Data Mart integration, privacy policies often constrain access to sensitive data in an organization's DW. It is therefore necessary to ensure the confidentiality of sensitive data before sharing access to the DW with the strategic partners. For example, retail trade companies could agree to share aggregated sales data—e.g., on a weekly basis—with their food suppliers, but refuse access to detailed sales figures. As far as the public sector is concerned, legal obligations have to be respected. For instance, analysis over the Data Warehouses of health insurance sub-organizations must ensure the confidentiality of patients' personal data (e.g., their medical records).

The main challenges addressed by the FedDW approach are (1) integration of multi-dimensional data at the logical schema level, and (2) query processing against the global schema. In particular, this thesis analyzes and classifies the conflicts at the schema and at the instance level of multi-dimensional data that commonly occur among autonomous Data Marts. Moreover, queries against the global schema of a FedDW Data Mart federation are answered from a virtual instance of the global schema that represents the current snapshot of the reconciled distributed data.

In order to integrate the facts and dimensions of multi-dimensional schemas, Data Warehouse administrators and designers need tool support. In particular, DW practitioners lack tools that adequately assist in (1) the design of integrated, mediated schemas, and (2) the definition of mappings between different schemas.

Although some promising approaches have employed UML profiles to model DW schemas—e.g., [Luján-Mora et al., 2006]—or mappings between DW schemas—e.g., [Luján-Mora et al., 2004]—comprehensive tools that support both these tasks of Data Mart integration have not been proposed yet.

Processing and answering queries against the global mediated schema is closely related to query answering over views. Relations in distributed database systems are typically union compatible across autonomous nodes. Therefore, the global query plan is relatively easy to determine. Distributed systems provide a component called *query coordinator* to manage query and data shipping between the nodes [Özsu and Valduriez, 1999]. In the case of federated systems, however, query answering must also consider that conflicts between autonomous nodes—concerning the schemas and/or the tuples—may have to be repaired. Besides that, federated systems need *query rewriting* capabilities to coordinate some global query plan as well, transforming the original query into a series of partial queries against the nodes.

1.3 State of the Art

Data Warehouse integration is more challenging than relational data integration because of the additional *dimension* entity. Compared to the relational model—which provides the "relation" as its only conceptual entity—the multi-dimensional model distinguishes facts and dimensions, whereby the facts link to the dimensions. In particular, so called "(data) cubes" store facts in measure attributes that are categorized by the attributes of several dimensions. The dimensions, in turn, can be organized in hierarchies of aggregation levels. Thus, mappings of multi-dimensional schemas have to consider (1) the attributes in facts and dimensions, (2) the dependencies between facts and dimensions, and (3) the aggregation hierarchies.

Current research on multi-dimensional integration is inspired by several previous approaches, mainly in the field of distributed databases [Özsu and Valduriez, 1999] and federated databases [Sheth and Larson, 1990]. In general, federated systems rely on *semantic mappings* to repair heterogeneities across the schemas and data. At query time, the mappings enable the translation of distributed data across the different schemas into a common representation. This approach—called *mediated schema*—enables a uniform query interface that frees the user from querying each data source and converting the result data manually [Doan and Halevy, 2005]. Typically, the schema and data heterogeneities in federated databases remain transparent to the user.

Two different strategies for describing an integrated schema over heterogeneous data sources with mappings are known from database research: global-as-view (GAV) and local-as-view (LAV). In both approaches, the actual data is physically stored among distributed autonomous sources. While GAV systems specify the global schema as view expressions over the sources, LAV systems conversely describe the sources as views over the global schema [Halevy et al., 2006]. It is known from theoretical database research that GAV mappings facilitate query processing, whereas LAV mappings are easier to maintain and evolve [Lenzerini, 2002, Halevy, 2001].

Federated data integration systems are more flexible compared to the physical migration of existing data into a mediated, *global schema* designed from the existing schemas. Integrating only the logical schemas to build the federation—instead of physically unifying existing data—allows the participating organizations to remain autonomous. This means that every organization discloses parts of their logical Data Warehouse schemas (e.g., by defining a "public" Data Mart), but is still entitled to change the local, logical schema. Especially if confidential and sensible data is shared through the federated system, such autonomous schema and data management is advantageous.

In contrast, *loosely coupled* federations of Data Marts without a global schema need other mechanisms for overcoming heterogeneity. A common approach has been to extend query languages with conflict resolution features. For example, Mangisengi et al. have proposed an XML-based query language, the operators of which allow the ad-hoc integration of autonomous Data Marts [Mangisengi et al., 2003]. The XML layer of their approach is responsible for the exchange of meta-data between autonomous schemas and supports some simple transformation of the data. The work of [Abelló et al., 2002], in turn, simply defines relationships based on structural similarity between Data Marts that enable drill-across queries. Clearly, the disadvantage of such systems is that the user is responsible for repairing all conflicts within the query.

Data Mart integration with semantic mappings among autonomous, logical schemas addresses two major aspects: dimension integration and fact integration. The *dimension integration* problem is more complex than traditional, relational schema integration because dimension attributes are structured in hierarchies of aggregation levels. In contrast, *fact integration* corresponds to the traditional challenges of schema matching and data matching among relational entities [Doan and Halevy, 2005]. However, the interdependencies among dimensions and facts complicate the fact integration problem as well since fact schemas reference dimension attributes as foreign keys. This way, combinations of dimension attribute values uniquely identify "coordinates" of cells in cubes [Golfarelli et al., 1998].

Several authors in the fields of distributed and federated Data Warehousing have addressed the integration of dimension schemas as an isolated problem. For example, Torlone and Panella developed the visual tool "DaWaII" for dimension integration [Torlone and Panella, 2005]. DaWaII allows the user to specify mappings between similar dimensions and check them for correctness, based on the concept of dimension compatibility [Cabibbo and Torlone, 2005]. Lately, in order to automatically discover semantic mappings among autonomous dimension schemas, Banek et al. have proposed a matching algorithm that combines structural schema comparison with linguistic heuristics to compute a numeric similarity measure [Banek et al., 2007].

In contrast, fact integration has not received much attention. While the integration techniques developed for databases (e.g., see [Zhao and Ram, 2007]) also provide basic operations for the Data Warehousing field, these techniques do not address the interplay between fact and dimension integration. Due to the interconnections between facts and dimensions in the multi-dimensional model, a comprehensive approach for Data Mart integration must consider that mappings among autonomous dimensions potentially affect the facts connected with these dimensions.

Altogether, traditional data integration techniques and systems are insufficient for the integration of Data Marts, as this thesis will show in Chapter 3. The numerous heterogeneities that occur in the multi-dimensional model call for (1) extended federated system architectures for multi-dimensional schemas and data, and (2) a semantic mapping mechanism powerful enough to handle all multi-dimensional heterogeneity. With existing technology, mappings among facts and dimensions of multi-dimensional schemas can only be defined separately, using different tools that lack comfortable interfaces and base upon diverse data models. The manual integration of facts with existing database technology is known to be very laborious and error-prone [Kimball, 2002].

1.4 Objectives

The primary goal of this thesis is to enable the model-driven integration of autonomous, heterogeneous Data Marts. To address the challenges discussed in Section 1.2, the thesis presents the novel approach *"FedDW"*. In particular, the thesis pursues the following goals:

- **Integration at the logical schema level:** Since Data Warehouses contain huge amounts of data, enable the integration of multi-dimensional Data Marts at the logical schema level, using semantic mappings. Thus, avoid the time-consuming migration of existing data at the physical level.
- **Autonomy of federated Data Marts:** Ensure that the organizations participating in a federation of Data Marts can retain their schema and data management autonomy. In particular, the privacy of confidential and sensitive data must be respected.
- **Comprehensive methodology:** Define a systematic and comprehensive classification of heterogeneities that occur among the schemas and extensions of autonomous, multi-dimensional Data Marts. Address the entire range of heterogeneities that may occur at the schema and at the instance level of the multi-dimensional data model. Provide solutions to repair all of those within the semantic mappings between logical schemas.
- **Tool Support:** Provide model-driven tool support for DW designers and business analysts, facilitating all stages of the Data Mart integration project. Design the tool suite as user-friendly as possible, provide easily comprehensible and intuitive user interfaces. Moreover, comply with open, official standards (e.g., the Common Warehouse Metamodel CWM) to support the DW products and platforms of multiple vendors. Use off-the-shelf technology for tool implementation. Test the practical viability of the FedDW approach and tools.

This thesis focuses on enabling the model-driven integration of autonomous Data Marts, but not on performance optimization of the query mechanism. To process queries over several, autonomous Data Marts—given the semantic mapping between logical schemas—the FedDW approach presented in this thesis uses only an elementary query plan.

The primary goal of our work is to demonstrate the viability of Model-Driven Architecture based concepts for two distinct phases of Data Mart integration: (1) definition of semantic mappings (including schema modelling of a global schema), and (2) basic OLAP query processing, using these mappings. Query rewriting techniques or sophisticated optimizing algorithms of the query plan are out of the scope of this thesis. Therefore, the test results present experimental results of a case study, but no detailed performance studies.

1.5 Contributions of the FedDW Approach

Integration of Data Marts at the logical schema level offers numerous advantages. First, the federated architecture integrates multi-dimensional data while the underlying systems remain autonomous. Schema and data management autonomy is particularly important if protection of sensitive data is an issue. Second, FedDW's conceptual architecture addresses the privacy concerns of participating organizations. The multi-tier schema management of the FedDW approach gives every organization the autonomy to exclude confidential or sensitive data from the export schema. Third, FedDW's internal "canonical" data model supports various Data Mart implementation platforms. All multi-dimensional schemas, data and the mappings are represented on the logical level. Fourth, the semantic mappings between autonomous Data Marts and the global schema are defined in *source-to-target* direction. These source-to-target mappings combine the advantages of the global-as-view and local-as-view integration paradigms—i.e., straightforward generation of query plans (GAV) with extensibility of existing mappings (LAV). FedDW's global schema remains stable despite possible changes of the local Data Mart schemas. If some local schema evolves, the according mapping becomes invalid and has to be updated analogously, but the global schema as well as the other mappings are not affected.

This thesis extends the State of the Art in two areas. For the Federated DW administrators, FedDW supports the tasks of designing the global schema and defining mappings among the heterogenous schemas of autonomous Data Marts with a comprehensive mapping and conversion language. These concepts are implemented in a visual integration tool called *"Global Schema Architect"* (GSA) that supports both, global schema design and semantic mapping design among the heterogenous schemas of autonomous Data Marts. From the users' viewpoint, FedDW provides a global cube with integrated schema and processes the user queries based on the mappings defined between the Data Marts and the global schema. All heterogeneity among the DMs remains transparent to users. *FedDW Query Tool* implements these concepts and provides a meta-data interface to the Global Schema Architect, enabling the business analysts to use the mappings designed by the Federated DW administrator. Thus, FedDW supports the entire workflow of Data Mart integration to enable faster business integration across organizational boundaries.

FedDW enables visual, model-driven integration of multi-dimensional Data Mart schemas with an easy-to-comprehend notation that is based on the Unified Modelling Language (UML). The UML-based notation of the Global Schema Architect's user interface depicts facts and dimensions of the autonomous Data Marts in an intuitive and user-friendly manner. The GSA user configures the Data Mart federation by both, designing the global multi-dimensional schema and specifying semantic mappings between the Data Marts and the global schema. While the visual design environment of the GSA tool supports a rich palette of conversion operators that address schema and instance conflicts among facts and dimensions, user interaction is reduced to a minimum possible extent.

FedDW's query processing algorithm uses algebraic optimization of the conversion operators within mappings, and data shipping from the Data Marts to the federation layer in order to compute the query result. This approach depends on several optimization strategies in order to achieve acceptable performance—e.g., caching of facts, replication of dimensions. While the prototype of FedDW Query Tool introduced in this thesis only implements a basic query processing algorithm, demonstrating the viability of the approach, there is also enough potential for additional performance optimization, as discussed in the final Part IV of the thesis.

1.6 Prototype implementation of FedDW Tools

In order to demonstrate the viability and practicality of the FedDW approach for Data Mart integration proposed in this thesis, we developed the *FedDW tool suite*, comprising prototypes of "Global Schema Architect" (GSA) and "Query Tool". GSA is a visual, model-driven design environment for the integration of autonomous Data Marts, while FedDW Query Tool executes SQL OLAP queries over the global, federated Data Marts designed with GSA. Together, FedDW's Global Schema Architect and Query Tool provide the functionality of a Federated DW System, according to the reference architecture introduced in Chapter 6 of this thesis.

The prototypes of the FedDW tool suite demonstrate a possible object-oriented implementation of the concepts proposed in Parts III and IV of this thesis. The goal of our implementation was to proof our concept, not to optimize the performance of query processing with FedDW Query Tool across the Data Marts in the Federated DW system. Investigating the potential of both, various strategies for determining an optimal query plan, and different query processing algorithms is an interesting direction for future research on Federated Data Warehouses. As discussed in our conclusions (Chapter 10), the implementation architecture of the FedDW tool suite presented in Chapter 9 allows numerous strategies for performance optimization (e.g., caching global cubes, exploiting the SQL OLAP query for optimization of the query plan, and so forth).

1.7 Outline

The remainder of this thesis is organized into four parts. Part I (Chapters 2 and 3) motivates the need for Federated Data Warehouse systems, describes a typical scenario in which a Federated DW system is an adequate technology, and reviews related work that enable Federated DW systems. Part II (Chapters 4, 5 and 6) develops the basic concepts for Federated DW systems, presenting a conceptual model for multi-dimensional data—that unifies the common concepts of Data Warehouse schemas—and analyzing the heterogeneities that can occur among autonomous Data Marts that conform to this model. Moreover, Chapter 6 defines the basic architecture of Federated DW systems. Part III (Chapters 7 and 8) introduces our approach to Data Mart integration that is named *"FedDW"*, presenting the methodology we propose for repairing heterogeneity among the schemas and instances of autonomous Data Marts, and defining the syntax and semantics of a query language to supports the integration methodology. Finally, Part IV (Chapters 9 and 10) explains our proof-of-concept implementation of Federated DW technology and presents test results. We conclude the thesis by highlighting the most challenging research questions left up for future work.

Chapter 2 – Case Study presents a use case from the health care sector, motivating the need for Federated Data Warehouse systems.

Chapter 3 – State of the Art discusses the enabling technologies of Federated DW systems. The chapter summarizes contributions of previous work in the field of Data Warehousing and related fields—e.g., federated databases, data integration—towards the integration of multi-dimensional data sources. It explains the preliminaries of, and identifies detailed requirements for the FedDW approach.

Chapter 4 – Conceptual Data Model formally defines the typical multi-dimensional concepts data mart, cube, measure, dimension, hierarchy, aggregation level, roll-up attribute, and non-dimensional attribute at the conceptual level, i.e. independent of any implementation aspects. Moreover, the data model introduces the dimension functions *members* and *level*. Thus, the FedDW conceptual model represents the formal foundation for a systematic classification of heterogeneity in multi-dimensional systems.

Chapter 5 – Taxonomy of Conflicts introduces the classification of heterogeneities in the multi-dimensional model. It analyzes in depth a taxonomy of heterogeneities among autonomous Data Marts in five categories, defined along the two dimensions *modelling scope* (schema – instance) and *model entity* (dimension – fact), plus the "cross-categorial" schema versus instance category.

Chapter 6 – Federated DW Architecture introduces the reference architecture for Federated DW systems, which is based on the "classical", general five-level architecture for federated databases [Sheth and Larson, 1990]. This reference architecture provides four-tiered multi-dimensional schemas (component schema, export schema, import schema, application schema). The separation of schemas to several layers supports the integration of several Data Marts at the logical schema level, while these retain full autonomy for local schema and data management. Compared to previous approaches in the field of multi-dimensional source integration, we propose to add (i) a stable, global schema, (ii) the Meta-data Dictionary, and (iii) the Dimension Repository to Federated DW systems.

Chapter 7 – Integration Methodology proposes a general methodology for the conjoint integration of dimensions and facts among multi-dimensional data sources. The integration methodology systematically addresses the categories of heterogeneity classified in Chapter 5. Its general idea is to produce source-to-target semantic mappings (i.e., from the autonomous Data Marts to a stable, global schema), such as demanded in the reference architecture. The proposed integration paradigm is the *minimum use* strategy between autonomous, multi-dimensional schemas and extensions, which is quite restrictive since it only keeps the elements of import schemas shared among all local Data Marts. Moreover, Chapter 7 defines the conversion operators of the Dimension Algebra and Fact Algebra, which provide the "building blocks" of FedDW's semantic mappings.

Chapter 8 – SQL-MDi Language introduces the novel language SQL-MDi, which allows the integration of logical, relational Data Mart schemas. SQL-MDi provides high-level conversion clauses for the heterogeneities analyzed in Chapter 5. The language is suited for both, the ad-hoc integration of multi-dimensional Data Marts, and the permanent definition of semantic mappings in the Federated DW reference architecture. The conversion clauses available in SQL-MDi correspond exactly to the operators of the Dimension/Fact Algebra. Thus, the Dimension Algebra and Fact Algebra represent one possible, procedural implementation of the SQL-MDi clauses. Chapter 8 defines the syntax of SQL-MDi precisely, and illustrates its use with numerous examples.

Chapter 9 – Prototype Implementation presents the prototype implementation of the FedDW tool suite, comprising the Global Schema Architect and Query Tool. The motivation behind these two tools is to facilitate the business analysts' tasks. Ideally, the users want to integrate and query autonomous Data Marts without having to write SQL-MDi code of the semantic mappings by hand. FedDW Global Schema Architect demonstrates that the model-driven architecture is well suited for the Data Warehousing domain. FedDW Query Tool, in turn, acts as the mediator of the Federated DW reference architecture. It processes queries across autonomous Data Marts, using the SQL-MDi statements designed with the Global Schema Architect.

Chapter 10 – Conclusions concludes this thesis. The Chapter summarizes the concepts for model-driven integration of autonomous, heterogeneous Data Marts proposed in the FedDW approach, and gives an outlook on interesting and challenging future research questions.

Part I

The Need for Federated Data Warehouse Systems

Chapter 2

Case Study

This Chapter presents a small, but illustrative use case from the public health care sector that highlights the problems and challenges of Data Warehouse integration in practice. Throughout the remaining Chapters, this thesis will refer to the case study for illustrating both, the challenges of Data Mart integration, and the proposed solutions. The following Chapter introduces basic notions of the multi-dimensional model, and explains the possible classes of heterogeneity among dimensions and facts. For better clarity and easier presentation, the case study defines two Data Marts, illustrating heterogeneity at the schema and at the instance level in isolation. Thus, the case study helps providing the basic understanding of the challenges addressed in the remainder of this thesis.

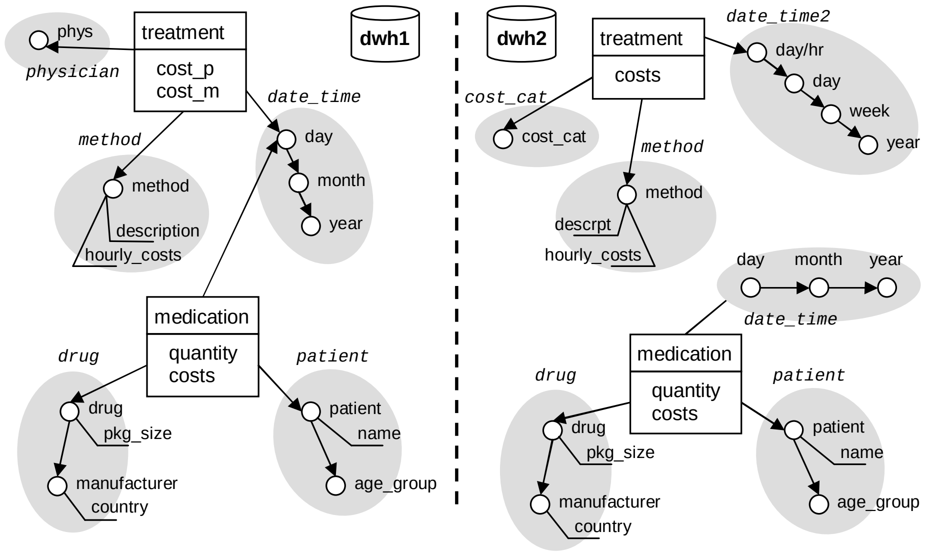


Figure 2.1: Health insurance – conceptual schemas of local Data Marts.

Data integration across autonomous organizations is a necessary prerequisite for any kind of business integration. The integration of database systems is the traditional approach that has been researched for several decades. Halevy et al. survey recent progress achieved by the data integration community and list future challenges [Halevy et al., 2006]. Federated Database Systems [Sheth and Larson, 1990] are the most prominent example of data integration systems (see Chapter 3).

Lately, due to the advent of more processing power and broadband connectivity over the Internet, the integration of data stemming from independent Data Warehouses or Data Marts of various organizations is becoming both increasingly interesting and important. The integration of autonomous DWs allows cooperating organizations to mutually share their "corporate memories". Successful DW integration opens up a larger pool of information, broadening the knowledge base for the decision makers in all participating organizations.

Without the appropriate methodology and tool support, DW integration is a tedious and error-prone task, though. As an illustrative example of the practical difficulties consider the conceptual DW schema of a fictitious health insurance organization, consisting of independent sub-organizations within several Federal States governed by a federal association. For simplicity, our scenario considers only two sub-organizations, both of which autonomously operate a Data Mart, as depicted in Figure 2.1. The schema is instantiated at two distinct nodes, named dwh1 and dwh2. Let us assume the health insurance's federal association wants to access the data of both DMs dwh1 and dwh2 using the global schema depicted in Figure 2.2.

The schemas in Figures 2.1 and 2.2 are specified in the *"Dimensional Fact Model" (DFM)* notation proposed by [Golfarelli et al., 1998]. DFM is a graphical notation for conceptual DW models. It allows to visualize the *facts* and *dimensions* of data cubes as well as the dependencies within a data cube. Rig-

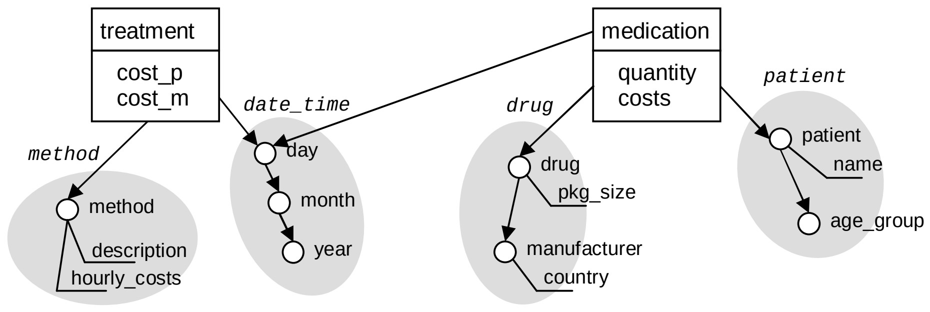


Figure 2.2: Global conceptual schema of health insurance DM federation.

orous definitions of the entities and concepts used in the multi-dimensional canonical data model of Federated DWs follow in Chapter 4 of this thesis.

The example schema defines two cubes, treatment and medication, each with three dimensions describing the facts. Notice that the DFM allows to "reuse" or "share" dimensions among multiple facts—e.g., date_time in Figure 2.1 [Golfarelli et al., 1998]. Both cubes exemplify numerous conflicts at the schema and at the instance level, that we introduce and detail in Chapter 5. We use two different cubes only for presenting different conflicts—in practice, all conflict categories could easily occur in a single cube.

Figures 2.3 and 2.4 illustrate an example instantiation of the conceptual DW schema at both sites dwh1 and dwh2. In the example, we use the postfix '_dim' to name dimension tables in order to avoid confusion with identical level attribute names. Notice that the two conceptual schemas are realized as so-called relational *star schema*. This means, that the physical DW schema consists of one table per cube ("fact table") and one table for each dimension, containing all the level and non-dimensional attributes ("dimension table"). In contrast, relational *snowflake schemas* define a separate table for each aggregation level of dimensions [Thomsen, 2002, Inmon, 2005].

Throughout the thesis we assume the conceptual schema be implemented physically as star schema for pragmatic reasons. This assumption allowed for an easier implementation of the FedDW prototype tool suite (see Chapter 9, pp. 143), since it facilitates the automatic recognition of fact and dimension tables when importing the meta-data of relational Data Marts. Moreover, as has been generally recognized, the star schema leads to faster query response times: compared to the snowflake schema, the star schema requires less join operations, and is therefore regarded more efficient from the OLAP point of view [Bauer and Günzel, 2006, Thomsen, 2002].

It is worth noting, though, that this simplifying assumption concerns only the implementation layer of both, the Data Marts and the FedDW tool prototypes (see Chapter 9, pp. 143). Conceptual schemas of Data Marts represent all aggregation levels of dimensions, whereby the internal implementation model—star or snowflake schema—is irrelevant [Golfarelli et al., 1998]. Thus, the ideas proposed in this thesis apply straightforwardly to snowflake schemas as well (cf. Chapter 4 Conceptual Data Model, pp. 51). The tools can be extended without much effort to support both star and snowflake schemas.

dwh1::medication (patient [l_patient], drug [l_drug], date_time [day]; qty, cost)

patient	*drug*	*date_time*	qty	cost
p1	'A'	25-01-06	1	68.4
p2	**'A'**	**03-02-06**	5	342.0
p3	'B'	17-02-06	4	728.0

dwh1::drug_dim (drug $\mapsto$ manufacturer)

[drug]	pkg_size	[manufacturer]	Country
'A'	**25 pcs.**	'Roche'	CH
'B'	40 pcs.	**'Novartis'**	CH
'C'	250 ml.	'Merck'	US

dwh1::patient_dim (patient $\mapsto$ age_group)

[patient]	name	[age_group]
p1	Doe, John	20-30
p2	Miller, Alice	40-50
p3	O'Neill, James	30-40

dwh1::date_time (day $\mapsto$ month $\mapsto$ year)

[day]	[month]	[year]
25-01-06	01-2006	2006
03-02-06	02-2006	2006
17-02-06	02-2006	2006
23-02-06	02-2006	2006
25-02-06	02-2006	2006

dwh2::medication (patient [l_patient], drug [l_drug], date_time [day]; qty, cost)

patient	*drug*	*date_time*	qty	cost
p5	'AA'	03-02-06	2	148.0
p6	'B'	14-02-06	3	624.3
p2	**'A'**	**03-02-06**	1	70.8

dwh2::patient_dim (patient $\mapsto$ age_group)

[patient]	name	[age_group]
p1	Doe, John	20-30
p2	Miller, Alice	40-50
p3	O'Neill, James	30-40

dwh2::drug_dim (drug$\mapsto$ manufacturer)

[drug]	pkg_size	[manufacturer]	Country
'A'	**30 pcs.**	'Roche'	CH
'B'	40 pcs.	**'Bayer'**	DE

dwh2::date_time2 (day/hr $\mapsto$ day $\mapsto$ week $\mapsto$ year)

[day/hr]	[day]	[week]	[year]
03-02-06 10:30	03-02-06	w05/06	2006
14-02-06 15:20	14-02-06	w07/06	2006
23-02-06 08:00	23-02-06	w08/06	2006
23-02-06 09:00	23-02-06	w08/06	2006
24-02-06 14:00	24-02-06	w08/06	2006

Figure 2.3: "Medication" fact tables and "drug" dimensions—[ALL] levels omitted—of the local Data Marts dwh1, dwh2 (case study).

dwh1::treatment (method [l_method], date [day], phys [l_phys]; cost_p, cost_m)

method	*date*	*phys*	cost_p	cost_m
X-ray	23-02-06	'Dr. A'	356.0	425.0
CT	23-02-06	'Dr. C'	125.2	1742.0
CT	25-02-06	'Dr. F'	473.0	903.8

dwh1::date_time (day $\mapsto$ month $\mapsto$ year)

[day]	[month]	[year]
25-01-06	01-2006	2006
03-02-06	02-2006	2006
17-02-06	02-2006	2006
23-02-06	02-2006	2006
25-02-06	02-2006	2006

dwh1::phys_dim (phys)

[phys]
'Dr. A'
'Dr. C'
'Dr. F'

dwh1::method_dim (method)

[method]	**description**	hourly_costs
X-ray	X-ray radiogram	2.360
CT	tomography	5.200

dwh2::treatment (method [l_method], date/hr [day/hr], cost_cat [l_cost_cat]; cost-$)

method	*date/hr*	*cost_cat*	cost-$
X-ray	23-02-06 08:00	personnel	480.0
X-ray	23-02-06 09:00	material	613.0
CT	24-02-06 14:00	material	624.5

dwh2::date_time2 (day/hr $\mapsto$ day $\mapsto$ week $\mapsto$ year)

[day/hr]	[day]	[week]	[year]
03-02-06 10:30	03-02-06	w05/06	2006
14-02-06 15:20	14-02-06	w07/06	2006
23-02-06 08:00	23-02-06	w08/06	2006
23-02-06 09:00	23-02-06	w08/06	2006
24-02-06 14:00	24-02-06	w08/06	2006

dwh1::cost_cat_dim (cost_cat)

[cost_cat]
personnel
material

dwh1::method_dim (method)

[method]	**descrpt**	hourly_costs
X-ray	X-ray radiogram	2.360
CT	tomography	5.200

Figure 2.4: "Treatment" fact tables and "time" dimensions of the local Data Marts dwh1, dwh2 (case study).

In the multi-dimensional model, every dimension definition consists of (1) an arbitrary number of *aggregation levels*, including the implicit [ALL]-level, (2) a *roll-up hierarchy* between the levels, i.e. a lattice on the levels, such that the dimension's instances (called members) form a tree, and (3) optional *non-dimensional attributes* to model the properties of dimension instances more precisely. For example, the patient dimension is composed of (1) the levels [l_patient] and [age_group], (2) the roll-up hierarchy {l_patient $\mapsto$ age_group $\mapsto$ ALL} and (3) the non-dimensional attribute name (of a patient). Formal definitions of the conceptual constructs of multi-dimensional schemas follow in Chapter 4.

The example Data Mart instantiations (see Figures 2.3 and 2.4) demonstrate a situation that is commonly found in practice. Obviously, both DMs represent a similar part of the real world, but the medication and treatment fact tables of dwh1 and dwh2 cannot be unified without previous transformations of their schemas or their data. On the one hand, the schemas of the medication cubes are identical and even match the global medication schema (see Figures 2.1 and 2.2), whereas their instances are heterogeneous (illustrated in Figure 2.3). On the other hand, the treatment cubes mismatch already at the schema level. Figure 2.1 reveals the heterogeneities among the two fact schemas and the two date dimension schemas.

In particular, the following heterogeneities exist among dwh1 and dwh2. Both medication cubes (see Figure 2.3) conform to the same schema, but contain numerous conflicts among their instance sets:

(1) The primary keys of some subset of the facts—given in bold font—are identical. Thus, the medication costs for patient 'p2' with drug 'A' on Feb. 3, 2006 remain unclear among the two cubes (*overlapping facts*).

(2) Both drug dimensions contain an instance named 'A', which is, however, erroneously named 'AA' at dwh2 (*value conflict in dimension attribute*).

(3) Finally, the roll-up hierarchies of the dimension instances (shown at the bottom of Figure 2.3) contain conflicting mappings for drug 'B': it rolls-up to manufacturer 'Novartis' in dwh1, as opposed to 'Bayer' in dwh2 (different *roll-up functions*).

In contrast, the two treatment cubes (see Figure 2.4) are heterogeneous among their logical schemata, affecting the dimensions and facts alike. Besides these schema level conflicts, though, the data cubes dwh1::treatment and dwh2::treatment are homogeneous at the instance level. The following list details the conflicts among the treatment cubes:

(1) Fact table dwh1::treatment defines two measures to distinguish different cost categories ('cost_p' for personnel costs and 'cost_m' for material costs) whereas at dwh2 the *instances* of dimension cost_cat identify the cost category (*schema–instance conflict*).

(2) Ignoring the cost_cat dimension (contained as implicit information in the cost_p and cost_m attributes), the data cubes differ in their respective number of dimensions: i.e., the phys (physician) dimension of dwh1 is not modelled at dwh2 (*dimensionality conflict*).

(3) The domain of the cost-attributes is incompatible among the treatment data cubes. While dwh1 records cost figures in Euros, dwh2 contains treatment costs in US-$ (*conflicting measure attribute domain*).

(4) The aggregation hierarchy of the date_time dimension contains four levels at dwh2 (day/hr $\mapsto$ day $\mapsto$ week $\mapsto$ year), compared to only three at dwh1 (day $\mapsto$ month $\mapsto$ year). Moreover, the domains of the [month] and [week] levels are different: months versus weeks of years (*diverse hierarchy*).

(5) Finally, the lowest aggregation level of the date_time dimensions is more fine-grained at dwh2: it records date plus time (day/hr), compared to only date without time at dwh1 (heterogeneous *base level domain*).

Only after repairing all the aforementioned conflicts at the schema, at the instance, and at the schema–instance level, the global schema as given in Figure 2.2 could be instantiated successfully. Figure 2.5 depicts the global medication and treatment cubes as they would be computed from the given local DMs dwh1 and dwh2. Notice that in a federated architecture the global cubes (named g::medication and g::treatment) are available for analytical queries, but need not necessarily be stored physically by the federated system. Instead, a federated DW system would typically compute global cubes "on the fly".

g::medication

patient	*drug*	*date*	SUM (qty)	SUM (cost)
p1	'A'	25-01-06	1	68.4
p2	**'A'**	**03-02-06**	**6**	**412.8**
p3	'B'	17-02-06	4	728.0
p5	'A'	03-02-06	2	148.0
p6	'B'	14-02-06	3	624.3

g::treatment

method	*date*	SUM (cost_p)	SUM (cost_m)
X-ray	23-02-06	676.0	833.6
CT	23-02-06	125.2	1742.0
CT	24-02-06	416.3	0.0
CT	25-02-06	473.0	903.8

g::drug_dim (drug $\mapsto$ manufacturer)

[drug]	pkg_size	[manufacturer]	Country
'A'	**25 pcs.**	'Roche'	CH
'B'	40 pcs.	**'Bayer'**	CH
'C'	250 ml.	'Merck'	US

g::date (day $\mapsto$ month $\mapsto$ year)

[day]	[month]	[year]
25-01-06	01-2006	2006
03-02-06	02-2006	2006
14-02-06	null	2006
17-02-06	02-2006	2006
23-02-06	02-2006	2006
24-02-06	null	2006
25-02-06	02-2006	2006

g::patient_dim (patient $\mapsto$ age_group)

[patient]	name	[age_group]
p1	Doe, John	20-30
p2	Miller, Alice	40-50
p3	O'Neill, James	30-40

g::method_dim (method)

[method]	**description**	hourly_costs
X-ray	X-ray radiogram	2.360
CT	tomography	5.200

Figure 2.5: Global fact tables and dimensions—[all] levels omitted—computed from the local Data Marts dwh1, dwh2

In the remainder of this thesis, we will use the case study to illustrate the fundamental problems in DM integration scenarios and discuss possible solutions. We emphasize that the case study presented herein is a simple example compared to real-world settings, that are usually more complex and complicated. Nevertheless, the case study demonstrates all essential conflict classes that may occur in practice, and the appropriate solutions. For an easier presentation, we intentionally kept the example at the minimum possible size.

Concluding the introduction to challenges in Data Mart integration, the following Table 2.1 gives an overview of possible classes of heterogeneities in the multi-dimensional data model. In practice, several of these conflicts commonly occur in combination, which motivates the need for a systematic methodology, helping to recognize heterogeneity among multi-dimensional schemas and data correctly. Each of the conflict classes given in Table 2.1 has been exemplified above in the case study, and will be defined in detail in Chapter 5.

Table 2.1: Overview of heterogeneities among Data Warehouses.

	Facts	Dimensions
Schema vs. Instance	Fact context as dimension members	Dimension members as contextualized facts
Schema level	• Naming conflicts (cube name, dimension attributes, measures) • Dimensionality (diverse dimension attributes, number of dimensions) • Diverse measures • Domain conflict (measures)	• Naming conflicts (dimension name, levels, non-dimensional attributes) • Diverse levels and/or hierarchies • Diverse non-dimensional attributes • Domain conflicts (level attributes, non-dimensional attributes)
Instance level	• Value conflicts (dimension attributes, measures) • Overlapping fact subsets • Disjoint fact subsets	• Heterogeneous roll-up functions • Non-dimensional value conflicts • Overlapping member sets

Chapter 3

State of the Art

Contents

This Chapter gives basic definitions of the notions “Data Warehouse” and “Federated system”, and summarizes previous effort made towards the development of Federated Data Warehouse Systems. It considers the integration of multi-dimensional database systems as the general problem and shows that, in particular, Data Warehouse integration is viewed as special case of traditional database integration. It summarizes the development chronologically, starting with the basics of Data Warehouses and Federated Database Systems. The Chapter surveys two major fields, data source integration and model-driven development. Based on the shortcomings of current approaches, the final Section formulates the requirements for improvements of existing work.

The present chapter reviews the enabling technologies of Federated Data Warehouses. The FedDW approach proposed in this thesis is primarily inspired by federated database architectures. In order to handle the richer semantics of the multi-dimensional model introduced by its additional dimension entity, however, our approach requires several enhancements of well-known concepts for the integration of autonomous and heterogeneous data sources.

This chapter is divided into several sections, that explain the fundamentals of Data Warehousing and OLAP (3.1), and survey two major research areas related to our approach towards Federated Data Warehousing: *data integration*, as well as *model-driven development* with the *Model Driven Architecture (MDA)* and *Unified Modelling Language (UML)* standards.

Data integration is a mature discipline with several decades of experience, spanning the fundamentals of federated databases (3.2), which lead to two other research areas with specifically rich experience: *physical integration* (3.3) and *logical integration* (3.4). Roughly, all data integration approaches belong to one of these two categories that utilize fundamentally different concepts (see Figure 3.1). In recent years, the community has applied these techniques also to the multi-dimensional model (3.5).

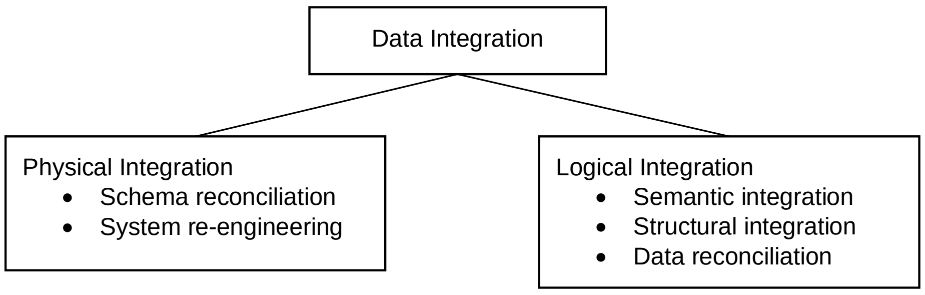


Figure 3.1: Tasks and vocabulary of data integration [Koch, 2001].

Terminology used in the data integration field is often inconsistent. In particular, many authors use the term “data integration” to denote logical integration, whereas “schema integration” refers to physical integration approaches [Doan and Halevy, 2005, Schmitt and Saake, 2005, Koch, 2001]. To avoid confusion, this thesis uses “*physical integration*” and “*logical integration*” to distinguish the concepts depicted in Figure 3.1. With “data integration”, in turn, we subsume all these concepts.

In contrast, model-driven development of information systems is a young research discipline. MDA is an official standard released by OMG, the *Object Management Group* [(OMG), 2003b]. Model-driven development abstracts from the internal code of systems, separating the *platform-independent* specification of system functionality from its actual, *platform-specific* implementation. The most important idea of MDA is the automatic transformation between platform-independent and platform-specific models. Thus, the model-driven development approach facilitates life-cycle management of information systems by improving the reusability of formal system specifications. The UML is commonly used to realize model-driven development of information systems. Section 3.7 summarizes previous applications of the UML for the Data Warehousing domain.

Finally, in Section 3.8 we summarize the current state-of-the-art in all the aforementioned fields. The chapter concludes by identifying the shortcomings of previous approaches towards Federated Data Warehouses. Based on these observations, we formulate the requirements for an ameliorated approach.

3.1 Data Warehousing and OLAP

This section defines the basic notions in the Data Warehousing field that are relevant for this thesis. Basically our work regards a Data Warehouse (3.1.1) or Data Mart (3.1.2) as *black box*, concentrating on how these systems—particularly, at the level of Data Marts—can be integrated under a federated architecture. Nevertheless, we summarize the methods and technology used for Data Warehouses as starting point for the realization of Federated Data Warehouses. Furthermore, we briefly explain On-line Analytical Processing (OLAP; 3.1.3).

3.1.1 Data Warehouses

Corporate Data Warehouses (DWs) are analytical information systems, i.e. databases that are optimized for analytical decision support rather than for high transactional throughput. More specific, “a Data Warehouse is a subject oriented, integrated, non-volatile and time variant collection of data in support of management’s decisions” [Inmon, 2005]. Thus, a DW represents an integrated collection of historical business data, supporting the business analysts in the decision making process. In that sense, the DW represents the “corporate memory” of all business data that is relevant for strategic decisions [Kimball, 2002, Inmon, 2005].

In contrast, *operational data sources* focus on efficient support for the processing of data workload in day-to-day business transactions. The operational data sources are designed around the organizational functions of a company (i.e., its business processes—e.g., sales management, marketing, customer services, and so forth). Data models for relational databases such as the Entity/Relationship (ER) model or the relational model optimize these systems for maximum throughput of relatively simple insert/update/delete transactions [Navathe and Elmasri, 2004].

The first distinguishing feature of DWs is *subject orientation.* This means, the modelling of DW schemas emphasizes the subjects (entities) that are of primary interest to business analysts and decision makers. These subjects are denoted the *dimensions* of the corporate DW. Typically, each dimension provides various *levels of granularity.* This focus on dimensional data has led to the notion of *multi-dimensional data model* and the *data cube* analogy. Indeed, the various dimensions in DWs span a n-dimensional space as container for other, non-dimensional values—called the *facts* of the cube. Facts of DWs, in turn, represent the interesting business “benchmarks”, i.e. mostly numerical, measurable attributes or simply *measures* of business processes. Thus, the facts represent transactional data from the operational data sources within the dimensional context of the DW [Golfarelli et al., 1998].

The second feature of DWs is *integration* or reconciliation of various operational data sources. More often than not, operational data sources have been developed and implemented independently before the corporate DW is introduced. Therefore, the *universe of discourse* is often represented differently among these systems—with respect to terminology behind attribute names, value domains of attributes, character encodings, etc. The conceptual schema of the DW contains the common, homogeneous representation of the source conceptual schemas [Golfarelli et al., 1998].

In this context, the DW resolves the existing heterogeneities between the various operational systems at the schema level and at the instance level, as classified by [Lee et al., 1995, Kim and Seo, 1991]. This DW design approach is called *data-driven* or *data-oriented* [Mazón et al., 2007, Mazón and Trujillo, 2007]. Recently, some authors have criticized the data-driven DW design approach for often overlooking the actual requirements of business users, and proposed *requirements-driven* (or sometimes called *goal-driven*) methods for better meeting user expectations [Prakash et al., 2004, Pardillo and Trujillo, 2008].

The third characteristic feature of DW systems is the *non-volatility* of data. A DW loads massive amounts of business data from operational systems, providing *read-only access* for analytical purposes. Usually, DWs prohibit any updates of their data [Kimball, 2002, Inmon, 2005]. In contrast, operational data sources are transaction oriented, which means that these systems permanently allow to insert, update or delete each tuple [Navathe and Elmasri, 2004].

Finally, the fourth distinguishing feature of DWs is *time variance*, i.e. emphasis on the time reference of factual data. By introducing the time dimension, a DW is able to keep historical versions of data, and thus allows analytical queries over different time periods [Golfarelli et al., 1998]. This is again clearly contrary to operational sources, in which usually data is modified without preserving older versions. Therefore, operational data sources store only the current "snapshot" of every tuple in the universe of discourse.

3.1.2 Data Marts

Data Marts (DMs) are more specific, usually departmental data repositories that are architecturally equivalent to the Data Warehouse concept, which we have summarized in the previous subsection. Two alternative design methods for DMs have been proposed: the *top-down* approach [Inmon, 2005] and the *bottom-up* approach [Kimball, 2002]. The difference between these approaches is which repository should be developed first: the corporate Data Warehouse or the departmental Data Mart(s). Inmon prefers to develop the corporate DW as the organization's comprehensive data repository first, and derive the DMs from the DW later on. Kimball, however, advocates exactly the opposite, arguing that departmental DMs are easier to develop due to the reduced problem size, and the DW can be derived later.

Although theoretically the top-down method [Inmon, 2005] is considered the more elegant solution (e.g., see [Jukic, 2006, Bonifati et al., 2001]), the bottom-up approach [Kimball, 2002] is easier to implement from the practical viewpoint and therefore commonly used in enterprises [Watson et al., 2001]. The two approaches have been heavily discussed, but it seems that both have their merits and weaknesses [Breslin, 2004]. For the purpose of this thesis, though, it is irrelevant *how* the organizational Data Warehousing infrastructure was built.

3.1.3 On-Line Analytical Processing (OLAP)

On-Line Analytical Processing (OLAP) is a category of software technology that provides executives, analysts and managers with aggregated business information via a "slice, dice and rotate" method of end user data access, augmenting or replacing more complicated query languages (e.g., SQL for relational systems). This slice and dice data access method gives the user consistently fast, consistent, interactive access to a wide variety of data views, organized by key selection criteria that match the dimensions of the corporate DW. "OLAP performs multi-dimensional analysis of enterprise data including complex aggregations, trend analysis and modelling. Derived from end-user requirements, OLAP enables end-users to perform ad hoc analysis of data in multiple dimensions, thereby giving them the insight and understanding they need for better decision making." [The OLAP Council, 2009]

The term "OLAP" was originally coined by E. F. Codd et al. who were the first to realize that conventional, relational querying is ill-suited as data model for analytical activities performed by business analysts. Instead, OLAP software must provide the multi-dimensional view on data that is important for decision support. Thus, Codd et al. proposed twelve rules to evaluate the proper functionality of OLAP products [Codd et al., 1993]. Later on, in 1995, Codd added six more rules to account for additional analytical requirements [Bauer and Günzel, 2006, p. 101]. To facilitate the evaluation of OLAP products, Pendse and Creeth—the founders of the "OLAP Report" project—simplified the 18 rules of Codd et al. to five key criteria *FASMI* (Fast Analysis of Shared Multidimensional Information) [Pendse and Creeth, 1995].

For inspecting and analyzing multi-dimensional data, OLAP tools provide high-level operations that conform to the "slice, dice and rotate" metaphor [The OLAP Council, 2009]. Since DWs allow only read access to data, but prohibit any updates, the result of these operations is a new cube, which conceptually is a view over the original cube data. The following high-level operations are commonly distinguished [Han and Kamber, 2000, pp. 39–104], [Bauer and Günzel, 2006, Chaudhuri and Dayal, 1997]:

- *Roll-up and drill-down*: the roll-up operation aggregates the fact values of the cube by changing to a higher aggregation level in one of the cube's dimensions The drill-down operation is the reverse of roll-up.
- *Slice and dice*: the slice operation performs selection on the members (i.e. the tuples) of one of the cube's dimensions, retrieving only the members satisfying the predicates given in the selection's condition. The dice operation is a slight variation of slice, extending the selection predicates on more than one of the cube's dimensions.
- *Drill across*: the drill across operation performs queries against more than one cube simultaneously, combining the fact data of cubes on their common dimensions. Thus, drill across can only be applied on sets of cubes that share at least one dimension. Sometimes the drill across operation is also called *cube join*.
- *Pivot*: the pivot operation rotates the visual ordering of the cube axes, providing an alternative presentation of the cube. However, pivot leaves the fact data presented in the cube unchanged.

3.2 Federated and Multidatabase Architectures

Historically, multi-database architectures have been the first approach towards the integration of distributed (autonomous) data sources. In general, multi-database systems are "light-weight" collections—without an integrated schema—of distributed and possibly heterogeneous databases. According to the classification of [Sheth and Larson, 1990], federated database systems slightly refine multi-database systems, allowing the component database systems to retain autonomous.

In non-federated multi-database systems, all heterogeneity between the data sources is not transparent. Heterogeneity may occur at several levels, e.g. the universe of discourse, its representation within data models, constraints on schemas, and implementation of the model (including the functionality offered by the database system, such as the query language and network interfaces). This leaves the user responsible for the integration task at query time. In the worst case, data has to be integrated using different query languages, and translated between several data models [Sheth and Larson, 1990].

Federated database systems are further categorized into tightly or loosely coupled systems [Sheth and Larson, 1990]. On the one hand, *tightly coupled* federated databases are administrated by some global authority as a single entity—not meaning that the loss component systems loose their autonomy. On the other hand, *loosely coupled* federated databases are managed locally. Typically, tight coupling of autonomous component databases might include the definition of an integrated schema, such that the federation logically appears to the users like a distributed database (see Figure 3.2).

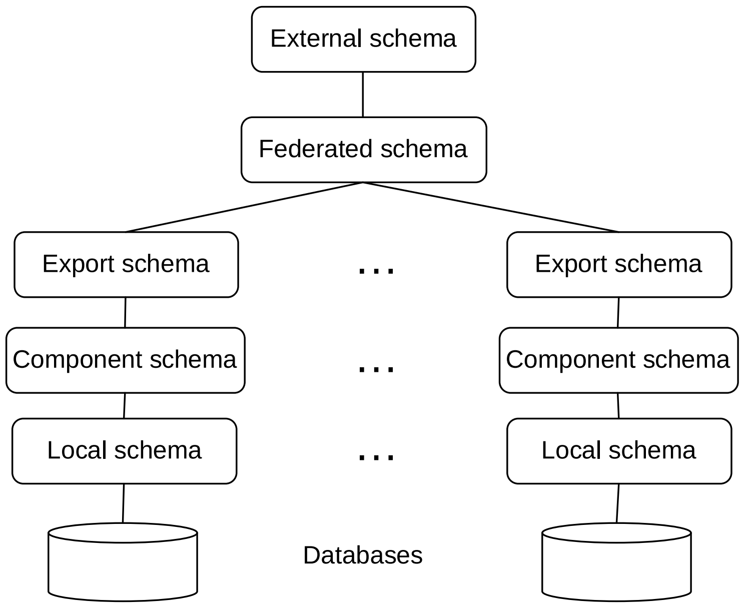


Figure 3.2: Five-tier schema architecture of federated databases [Sheth and Larson, 1990].

Autonomy in the sense of [Sheth and Larson, 1990] refers to several meanings, i.e. design autonomy, communication autonomy, association autonomy, and execution autonomy. In particular, *design autonomy* entitles the local administrators of component systems to choose the universe of discourse, its representation, as well as the data models and query languages. *Communication autonomy* and *association autonomy* permits local systems to decide about their "isolation level", i.e. the visibility of the database interface respectively the services it offers within the federation. Finally, *execution autonomy* allows the re-ordering of operators within queries sent to the component systems.

3.3 Physical Integration

Physical integration refers to the process of building a new information system with an integrated schema from heterogeneous source systems. Thus, the integrated schema is derived from the existing schemas of the original information systems [Schmitt and Saake, 2005, Batini et al., 1986]. As depicted in Figure 3.1, physical integration is one of two possible approaches for integrating heterogeneous data sources.

Sometimes, physical integration is referred to as *data migration* [Kimball, 2002]. The design of an integrated information system involves the re-engineering of original, heterogeneous schemata in order to obtain one global schema. In order to implement the integrated schema, all data has to be migrated—i.e., copied and transformed—from the source systems to the newly built, integrated information system.

Clearly, physical integration is the straightforward approach to data integration. While the physical integration process leads to the simplest possible architectures of integrated systems, the design process itself is not necessarily easier than for logical integration (cf. 3.4). The advantage of physically integrated information systems is that all heterogeneity—e.g., concerning the universe of discourse, the data model, and query languages—is repaired *a priori*. Thus, queries against the integrated schema are simple to process since they can be sent directly to the global information system implementing the integrated schema.

During the physical integration process, all heterogeneity among the existing data sources must be repaired. Usually this process is error-prone and laborious, because numerous possible heterogeneities must be considered, as sketched above. Lee et al. have given the detailed classification of particular conflicts in the relational model [Lee et al., 1995], refining the previous work of [Kim and Seo, 1991]. In order to facilitate the integration tasks from the user's point of view, several visual tools have been proposed that use machine learning techniques to infer the existing conflicts between relational schemas from user interactions. The most notable of these systems are SPHINX [Barbancon and Miranker, 2007] and Clio [Yan et al., 2001].

It has been recognized that the integration of heterogeneous schemas among various, independent information systems is a tedious and error-prone task [Seligman et al., 2002]. To address this issue, various *automated schema matching* techniques have been proposed [Rahm and Bernstein, 2001, Evermann, 2009]. Particularly challenging is the discovery of complex matches—i.e., 1:n or n:m mappings between relations. To this end, Xu and Embley propose an integration framework based on data frames and snippets of domain ontologies [Xu and Embley, 2006, Embley et al., 2004]. Their framework combines *element matching* (i.e., it analyzes typical, expected data values in attributes) with *structure matching* (i.e., it inspects the schema elements with the help of WordNet [Miller, 1995]) to generate source-to-schema mappings. The iMAP approach uses similar techniques for detecting *match candidates*—both simple 1:1 and complex 1:n matches—in a semi-automatic framework. The main contribution of iMAP is its *explanation module*, which generates *dependency graphs* for the match candidates together with numeric rankings of estimated similarity [Dhamankar et al., 2004]. This approach reduces the danger of false matches since the user, presumably an expert of the underlying application domain, is ultimately responsible for choosing the appropriate match from the candidates list of matches. Nevertheless, the automatic generation of match candidates considerably facilitates the user's workload.

To implement a global, physically integrated information system, full and unrestricted access to the local, heterogeneous data sources is necessary. Therefore, it is an appropriate approach for permanent integration of data sources. For the Data Warehousing domain, however, physical integration is often not feasible for privacy of confidential data (e.g., in the case of business cooperations the partners certainly will only share non-critical business data). Moreover, the sheer volume of data stored in DW systems often causes an unmanageably high complexity of data migration projects, even among the divisions of a single company [Kimball, 2002].

3.4 Logical Integration

Integration of heterogeneous data at the *logical level*—without migrating the data physically to an integrated system, as explained in the previous section—is the alternative approach to data integration (cf. Figure 3.1). Basically, *data integration systems* provide access to several data sources by repairing all heterogeneity at the level of logical entities [Doan and Halevy, 2005]. For that purpose, logical conversion functions—so-called *mappings*—specify how to obtain a common representation of all data across the heterogeneous sources [Lenzerini, 2002]. Logical data integration has been a major challenge for the database community over the past decades (e.g., see the surveys of [Halevy et al., 2006, Doan and Halevy, 2005]). In the following subsections we briefly summarize the current state of this research area.

Historically, the first approaches towards the integration of autonomous data sources developed extension for query languages, completely delegating the integration task to the users—like in multi-database systems (3.4.1). Later, these approaches were refined to the specification of procedural or declarative mappings between heterogeneous source systems, leading to two major approaches: *global-as-view* respectively *local-as-view* (see 3.4.2 and 3.4.3). Recently, an approach called *both-as-view* has been proposed in an attempt to combine the advantages of the two previous techniques (3.4.4). Furthermore, the more sophisticated problem of applying previous solutions to Data Warehouse integration has drawn the attention of researchers (3.4.5).

3.4.1 Multi-system Query Languages

In order to improve the usability of multi-database systems, early approaches towards logical integration have developed *multi-database query languages* for manipulating data across database boundaries [Grant et al., 1993]. In multi-database systems, the user is responsible for overcoming heterogeneity when formulating the query (cf. 3.2). Multi-database query languages provide transparency from the interfaces and query languages used internally by the data sources. Thus, these languages add an abstraction layer to multi-databases. From the user's perspective, multi-database query languages provide at least the advantage of a uniform language across the heterogeneous data sources. Nevertheless, conflict detection and resolution remains the user's responsibility.

Multi-database languages extend standard query languages with resolution features for schema and data heterogeneity. The first of these languages—named MDSL—was introduced by [Litwin and Abdellatif, 1986]. Later, the MSQL language refined MDSL functions for the SQL standard [Litwin et al., 1989]. Other prominent examples of multi-database languages are SchemaSQL [Lakshmanan et al., 2001], Federated Interoperable SQL (FISQL) [Wyss and Robertson, 2005], and nD-SQL [Gingras and Lakshmanan, 1998]. In what follows, we briefly summarize these languages, and also present higher-order logics and reflection based query techniques.

MSQL.

MSQL (Multidatabase SQL) allows to query and manipulate sets of relations—so-called *multirelations* [Litwin et al., 1989]—across multiple databases. MSQL extends standard SQL [(ISO), 1992], defining multirelational operators that correspond to SQL's clauses with an added prefix 'M' (e.g., MSELECT, MPROJECT, MJOIN). Internally, MSQL is translated to equivalent sequences of standard SQL operators that produce the desired result. Thus, MSQL is compatible to existing relational database systems that implement the SQL standard.

The *multirelational algebra* formalizes MSQL semantics [Grant et al., 1993]. The multirelational algebra represents a procedural implementation of the MSQL operators. In turn, the multirelational algebra operators map to the relational algebra [Codd, 1970]. This approach ensures that each multirelational operation delivers the same result as the analogous relational operation.

SchemaSQL.

SchemaSQL is a *second-order* relational query language, extending the SQL standard with variables over database meta-data and additional aggregation functions [Lakshmanan et al., 2001]. Standard SQL allows only variables containing tuples of relations, not the variables of relations themselves [(ISO), 1992]. The meta-data variables of SchemaSQL allow for powerful data transformation features. In order to generate integrated views over several data sources, SchemaSQL provides the concept of *output schemas* (CREATE VIEW clause). Moreover, SchemaSQL extends vertical aggregation functions known from the SQL standard with horizontal aggregation. Thus, SchemaSQL is ideally suited for querying across multiple, autonomous data sources.

nD-SQL.

Similar to SchemaSQL, nD-SQL extends the SQL standard with meta-data variables and additional aggregation features. The nD-SQL language introduces powerful schema transformations—e.g., variables ranging over attribute names and other meta-data—and even supports multi-dimensional data [Gingras and Lakshmanan, 1998]. Aggregation hierarchies in dimensions cannot be represented, though, which seems too restrictive in most practical scenarios.

The main difference distinguishing nD-SQL from its "predecessor" SchemaSQL is the underlying data model. nD-SQL introduces so-called *concepts* ("global relations"), with an associated list of *criteria* ("global attributes"). These constructs are used for *semantic integration* of the logical entities across heterogeneous data sources. For that purpose, the WHERE clause of nD-SQL provides two additional keywords, ISA and HASA, for assigning concepts to local relations and criteria to local attribute names, respectively. Analogously to SchemaSQL, nD-SQL allows for the generation of integrated views. The FOR keyword within the SELECT clause of nD-SQL declares a dynamic output schema [Gingras and Lakshmanan, 1998].

Federated Interoperable SQL.

Federated Interoperable SQL (FISQL) adds second order language operators to an extended model of relational data—the *Federated Relational Model*—together with transformation functions between the data and meta-data levels. Compared to SchemaSQL and other previous proposals, FISQL adds considerably more functionality. In particular, FISQL supports nested queries instead of view mechanisms, which allows to chain several FISQL queries to more complex expressions. Moreover, transformations of data into meta-data (or vice versa) are generally available in FISQL, whereas SchemaSQL and other languages allow only a single column of data to generate meta-data [Wyss and Robertson, 2005]. Thus, FISQL is the most powerful language for query processing across heterogeneous data sources proposed so far.

The algebraic query language *Federated Interoperable Relational Algebra (FIRA)* is the functionally equivalent procedural implementation of FISQL. To ensure its compatibility with SQL-based data sources, the FIRA is downwards compatible with the relational algebra. However, the *compositionality* of FIRA defines a more comprehensive closure than the relational algebra: a FIRA query maps a set of databases to a single database [Wyss and Robertson, 2005]. In contrast, relational algebra maps a (set of) relation(s) to a single output relation [Codd, 1970].

The outstanding contribution of the FISQL/FIRA approach is a considerable extension of data/meta-data transformation features, compared to SchemaSQL and other languages. In addition to the normal relational operators [Codd, 1970], FIRA comprises operators for *promoting* data to meta-data, *demoting* meta-data to data, and *dereferencing* columns of data. The dereference concept allows to access the values of some column directly, tuple by tuple, as if the values were columns themselves. Thus, FISQL/FIRA treats both the data and meta-data of relational data sources as completely equivalent first class citizens of its underlying data model [Wyss and Robertson, 2005].

This involves two important enhancements of FISQL/FIRA semantics in comparison to previous approaches. First, the internal model of the language removes the distinction between data and meta-data, which enables all transformation operators to be used more than once per query. Second, the syntax used for data/meta-data transformations is more easily readable and comprehensible.

An earlier proposal, *MetaData SQL (MD-SQL)*, also extends SchemaSQL with a two-tiered meta-algebra. Compared to standard SQL operators, MD-SQL introduces two additional concepts: generalized join and generalized projection. The *generalized join* evaluates the relational join operator over the set of relations specified by a meta-data variable of a MD-SQL sub-query. In similar fashion, the *generalized projection* adds to the SELECT clause of a MD-SQL query the set of attributes specified by a meta-data variable [Rood et al., 1999].

Generally, MD-SQL shares most of the goals of FISQL. The main difference between these two approaches is that the sub-query capabilities of MD-SQL are more limited. Such as in SchemaSQL, the "nested variable declaration" over database meta-data must be unpacked before the algebraic query plan can be constructed. Moreover, the generalized join operator is more complex to evaluate than the appropriate FISQL operator [Wyss and Robertson, 2005].

Higher Order Logics-based languages.

Deductive databases combine logic programming with relational database technology [Bravo and Bertossi, 2005]. A deductive database specifies relations through a declarative language, i.e. as *rules* over *predicates* ("relations") with *arguments* ("attributes"). In order to compute instances of deductive databases, an inference engine interprets the rules over a set of given *facts*, deducing tuples of the target predicates [Navathe and Elmasri, 2004, pp. 784].

The notation of queries against deductive databases—i.e., of rule declaration—is inspired by the *Prolog* logic programming language. Rules consist of a *conclusion* (head) on the left-hand side, and *premise(s)* on the right-hand side (the body). Typically, the rule conclusion contains only a single predicate, representing the "target relation". The rule body, in turn, consists of one or more predicates, that are implicitly connected with the *logical and* operator [Navathe and Elmasri, 2004, pp. 784]. *Datalog* is a typical example of deductive query languages that use Prolog as their starting point.

The main advantage of deductive databases over SQL-based relational databases is the possibility to specify *recursive rules.* Furthermore, the rule based inference mechanism provides a framework for deducing new information that is not necessarily stored physically, i.e. as predicate facts in one of the data sources [Navathe and Elmasri, 2004, pp. 784]. Therefore, deductive databases are ideally suited for handling very large sets of data, or for integrating several data sources with rules [Bravo and Bertossi, 2005].

IDL (Interoperable Database Language) introduced higher order logical operators, i.e. expressions ranging over the meta-data of data sources. The data model underneath the IDL language defines the universe of discourse as *tuple of databases*, each database as *tuple of relations* (predicates), and each relation as *tuple of attributes* (arguments). Consequently, IDL expressions may contain names of relations or attributes, instead of ranging only over data (facts or tuples), as is the case in the relational algebra. Thus, the higher order operators enable the transformation of heterogeneous schemas across multiple data sources into an output schema [Krishnamurthy et al., 1991].

Additionally, IDL allows the definition of *higher order views* as target predicate. This means that the number and names of output relations (predicates) depends on some IDL expression, enumerating over the data or meta-data of some data source. The values contained in the IDL expression determine the number and names of output relations [Krishnamurthy et al., 1991]. In contrast to Datalog, however, IDL does not allow recursion within rules [Bravo and Bertossi, 2005].

HiLog extends traditional predicate logic with higher order logical operators [Chen et al., 1993]. In particular, HiLog's syntax permits highly flexible rules: it allows arbitrary terms to appear where the predicate calculus—the formal foundation of Prolog—expects only predicates, functions and atomic formulas. This feature is particularly powerful since it enables the transformation of data (facts) into meta-data (predicates or their attributes), or vice versa. HiLog's semantic, however, is first order which facilitates its theoretical proof and practical implementation. Similar to Datalog, HiLog allows recursive rules, as has been explained above. Thus, HiLog's functionality exceeds the expressive power of SQL and the relational algebra—which corresponds to first order logical functions [Codd, 1970]—while its implementation is possible in standard Prolog compilers [Chen et al., 1993].

SchemaLog is another Prolog-based logical programming language, extending rule predicates with higher-order operators. Analogously to HiLog, the higher-order logical expressions of SchemaLog allow to transform heterogeneous schemas of multiple data sources into homogeneous output predicates [Gingras et al., 1997]. Moreover, SchemaLog introduces powerful aggregation features. Besides vertical aggregation (i.e., column-wise over attributes) known from SQL and relational algebra, SchemaLog supports *horizontal aggregation* (i.e., tuple-wise over several attributes) and *global aggregation* (i.e., over tuple blocks across several relations). These additional aggregation features increase the user's flexibility when designing the "global" output relations [Lakshmanan et al., 1997].

Reflection-based languages.

Reflection is a functional extension to the object oriented programming paradigm. Reflection-oriented programs may analyze their own state and behavior, and modify themselves at runtime. The reflection concept includes self-examination, self-modification and self-replication of programs. In the context of database query languages, reflection is used for dynamically generating database sub-queries from an original database query [Van den Bussche et al., 1996].

Schema Independent SQL (SISQL) adopts the basic SQL syntax, but the user may replace concrete attribute names or relation names with variables [Masermann and Vossen, 2000]. At runtime, the SISQL query processor *reifies* the query. This means, it dynamically analyzes a data dictionary of known data sources for possible replacements of the variables with attributes or relations. For every data source identified during reification, the SISQL processor generates a query formulated in the *Reflective SQL* language [Dalkilic et al., 1996].

A procedural implementation of Reflective SQL, in turn, is the *Reflective Relational Algebra* introduced in [Van den Bussche et al., 1996].

SISQL represents a user friendly approach for ad-hoc integration of heterogeneous data sources. The advantage of the SISQL language is its robustness—schema changes of local data sources are completely transparent to the users. Thus, the reflection concept allows for querying data sources without exact knowledge about the schemas of the sources [Masermann and Vossen, 2000].

However, reflection-oriented query languages suffer from the high complexity of query processing and their non-intuitive syntax. In particular, the reification of reflective queries requires several iterations of reformulation on a query tree, based on the current state of databases referenced in the query. This approach is more expensive than traditional query evaluation.

3.4.2 Global-As-View Approaches

In the *global-as-view (GAV)* approach to data integration, heterogeneities among schemata of data sources are repaired by the definition of mappings. GAV mappings describe an integrated, homogeneous schema in terms of the data sources—analogously to view definition (hence the name "global-as-view"). To some degree, GAV mappings explicitly tell the data integration system *how* to overcome heterogeneity among the sources. Therefore, GAV data integration is often denoted procedural [Lenzerini, 2002].

GAV mappings are commonly used in *mediator* architectures to specify the solution to a particular heterogeneity problem among distributed data sources. Mediators provide a global schema—called the *mediated schema*—against which the users send queries[Wiederhold, 1992]. Instead of repairing heterogeneity among data sources "on-the-fly" at query time, the mappings are stored in a repository and reused by the mediator for rewriting the original query. Thus, mediator architectures are much more comfortable to use than multi-databases.

Prominent examples of mediator-based data integration systems TSIMMIS (The Stanford-IBM Manager of Integrated Information Systems), Hermes and Garlic. *TSIMMIS* uses a declarative mechanism—the *Mediator Specification Language* MSL—for the specification of mediators [Garcia-Molina et al., 1997]. Based on the *Object Exchange Model* [Papakonstantinou et al., 1995], the TSIMMIS framework compiles the MSL specification down to wrappers of individual data sources, and generates the mediator among the wrappers. *Hermes* (He̲terogeneous R̲easoning and Me̲diator S̲ystem) provides a similar mediator design toolkit with special focus on the integration of multimedia data sources. Internally, Hermes uses parameterized procedure calls to access data sources with restricted query functionality [Adali et al., 1996]. *Garlic* is a research prototype with similar aims to the Hermes system. Originally, the Garlic system has been developed to support the integration of both, relational and non-relational data sources, also focusing on multimedia data sources [Josifovski et al., 2002]. The Garlic query processing approach has also been successfully integrated into IBM's DB2 database system [Carey et al., 1995].

Query processing in GAV data integration systems is usually quite simple, because the mappings specify exactly which queries over data sources correspond to which part of the global schema. Thus, due to the procedural flavor of mappings, the straightforward *unfolding* of user queries is sufficient to answer the user query. Unfolding means the generation of a query plan—i.e., a sequence of queries against the local data sources—from (1) the original query against the global, integrated schema, plus (2) the mappings [Lenzerini, 2002].

The disadvantage of the GAV approach is its vulnerability to system changes. In particular, this includes two different aspects. First, extending GAV data integration systems with a new data source becomes problematic, if the new data source has an impact on several elements of the global schema (e.g., if a new attribute sub-category must be introduced in the global relation books for classifying book stock in a library management system). Second, the GAV paradigm is only effective if the schemas of data sources are stable [Lenzerini, 2002].

GAV data integration systems allow the materialization of globally defined, integrated views. Strictly speaking, Data Warehouses (see Section 3.1) are themselves data integration systems that follow the GAV paradigm with view materialization. While the materialized view approach considerably improves query response time, it also introduces the danger of reading *potentially outdated views*. In order to avoid this problem, numerous *view maintenance* techniques have been proposed. This work is clearly out of the scope of this thesis; we refer to [Lee et al., 2007, Zhou et al., 2007, Gupta and Mumick, 2006, Quass et al., 1996] for an overview of various maintenance techniques for materialized views.

3.4.3 Local-As-View Approaches

Data integration systems conforming to the *local-as-view (LAV)* approach also define mappings between heterogeneous data source schemas, but follow exactly the opposite paradigm than GAV. LAV mappings describe the local data sources as "global mediated" schema, with each data source specified in terms of global schema elements, i.e. as view over the global schema. In contrast to the GAV approach, LAV mappings specify *what* global schema elements the local data sources contain (rather than *how* the data sources map to the global schema, as is the case in the GAV approach). Therefore, LAV mappings are typically of declarative nature [Lenzerini, 2002].

It is important to point out, though, that in both approaches only the sources physically store the actual data, whereas the global schema consists of *virtual relations*. This means, that the data integration system instantiates the global relations only at query time in both, the GAV and LAV paradigm [Lenzerini, 2002]. Otherwise, globally stored data in a networked architecture with several other data sources is characteristic for distributed database systems [Navathe and Elmasri, 2004].

Data integration with LAV mappings is particularly appropriate if the global schema is based on an *enterprise model* or *ontology*. An enterprise model or ontology uses a formal mechanism for specifying a taxonomical descriptions of all conceptual entities and their interrelationships in the universe of discourse shared by all data sources. However, the LAV data integration paradigm is only effective if the enterprise model or ontology underneath the global schema is stable. Every change of global schema elements invalidates the existing mappings, which can entail considerable effort for redesign of mappings [Lenzerini, 2002].

Ontology-based data integration became a popular research field once the community discovered the positive effects of a formalized global enterprise model. The big advantage of the ontology-based data integration approach is that—besides relational, well-structured data—semi-structured and even unstructured data sources can be considered for integration to the global system. The main problem of the ontology-based approach is that the proper design of the ontology itself is of crucial importance to the outcome of the data integration process. Thus, many different formalisms and design tools for ontologies have been proposed. For comprehensive surveys of this field we refer to [Buitelaar et al., 2008, Xu and Embley, 2006, Tzitzikas et al., 2005].

Unfortunately, query processing in LAV data integration systems is inherently more complex than in GAV systems because the exact schema of the data sources is unknown. Basically, query plans must be *inferred* from incomplete information since the only knowledge about the data sources is represented in the mappings, i.e. the view specifications [Lenzerini, 2002]. It is known, though, that query processing in LAV data integration systems reduces to the problem of answering queries using views, a well-understood problem in database theory [Halevy, 2001].

The problem of *answering queries using views* is generally defined as finding an equivalent (and possibly minimal), rewritten query that accesses only database views, instead of executing the original query against a (set of) database relation(s) [Pottinger and Halevy, 2001, Abiteboul and Duschka, 1998, Srivastava et al., 1996, Yang and Larson, 1987]. It has been shown, however, that an equivalent rewriting for queries is often impossible to compute in data integration systems. If the extensions of a given set of views (i.e., LAV mappings to data sources) do not define a unique global database instance, the best possible solution is to find a *maximally contained* rewriting, which often involves several source queries with subsequent union [Halevy, 2001, Duschka et al., 2000].

The big advantage of the LAV paradigm is the *extensibility* of the data integration system. Whenever a new data source should be integrated, it is possible to add a mapping for the new source without invalidating the other mappings [Lenzerini, 2002]. There are many real-world settings in which such flexibility for incremental definition of mappings is advantageous (e.g., after mergers or acquisitions). Prominent examples of LAV data integration systems are the Information Manifold [Levy et al., 1996], InfoMaster [Genesereth et al., 1997] and SIMS [Arens et al., 1993], all of which also introduce efficient algorithms for finding maximally contained query rewritings.

3.4.4 Both-As-View Data Integration

The *both-as-view (BAV)* approach to data integration combines ideas from the previously presented techniques GAV and LAV. The *AutoMed* integration framework [Boyd et al., 2004] defines a generic *hypergraph-based data model (HDM)* [Poulovassilis and McBrien, 1998] for integrating the schemas of heterogeneous data sources. HDM provides low-level conversion constructs. This approach offers the advantage of a unifying semantics for several higher level modelling languages (e.g., relational, ER, UML, XML). In order to map heterogeneous schemas, the BAV approach defines *transformation pathways* between data sources and the global schema. Instead of specifying mappings as view definitions such as the GAV and LAV paradigms, the transformation pathways comprise primitive transformations on the model representation in HDM [McBrien and Poulovassilis, 2003].

Basically, BAV data integration combines the advantages of the previously proposed GAV and LAV approaches. It has been shown that both, GAV and LAV mappings can be derived from BAV transformation pathways [McBrien and Poulovassilis, 2003]. Thus, data integration systems conforming to the BAV approach are more secure against schema evolution of both, the global schema—e.g., as the consequence of changes to the enterprise model—and the local data sources, if these are autonomous [McBrien and Poulovassilis, 2002].

Similar ideas are proposed in the so-called *BGLaV* ("both global and local as view") approach. In BGLaV, the user specifies a relational "*target schema*"—i.e., the global schema—which is independent of any of the source schemas. Subsequently, *source-to-target mappings* are defined for each data source individually [Xu and Embley, 2004]. This approach—in similar fashion to the BAV approach [Boyd et al., 2004, McBrien and Poulovassilis, 2003]—combines the advantages of GAV and LAV data integration. While processing of queries against the global schema follows the simple unfolding of mapping rules such as in GAV, the independent global schema ensures extensibility of the data integration system such as under the LAV paradigm [Xu and Embley, 2004]. This means, adding a new data source never changes the global schema, which in turn eliminates the negative side effects on existing mappings that are otherwise observed in the GAV paradigm [Lenzerini, 2002].

3.4.5 Data Integration Approaches for Multi-dimensional Systems

Data Warehouse integration exceeds relational data integration in terms of complexity because the multi-dimensional model introduces the additional dimension entity [Golfarelli et al., 1998]. So-called "data cubes" of DW schemas store *facts* in measure attributes, categorized by the several *dimensions* that represent business perspectives. Dimensions, in turn, may be organized in hierarchies of aggregation levels (cf. Chapter 4, pp. 51).

Several authors have proposed loosely coupled integration of multi-dimensional Data Marts without a global schema, based on XML technologies [Niemi et al., 2002, Pedersen et al., 2002c, Pedersen et al., 2002b, Jensen et al., 2001]. Since XML technology allows to include Web data into the Data Warehouse infrastructure, XML has been highly popular for multi-dimensional integration [Kimball and Merz, 2000]. Moreover, Golfarelli et al. have proposed to exploit enterprise knowledge represented in XML documents for the design of Data Warehouse conceptual schemas [Golfarelli et al., 2001]. This is motivated by the observation that almost any data model can be represented efficiently using XML [Christophides et al., 2000].

XML technology can represent the data stock of a Data Warehouse to different degrees. An interesting intermediate approach is the logical federation of OLAP sources with XML data sources proposed in

[Pedersen et al., 2002c, Jensen et al., 2001]. In contrast, the XCube approach fully represents facts and meta-data of multi-dimensional cubes in XML documents [Hümmer et al., 2003].

The framework of Mangisengi et al. defines an XML-based query language which allows the ad-hoc integration of Data Marts [Mangisengi et al., 2003]. The XML layer of their approach exchanges meta-data between autonomous schemas and supports some simple transformation of the data. Thus, while query processing exploits XML technology, the underlying data model of the Data Marts is not restricted to a particular representation. In contrast, [Abelló et al., 2002] define relationships based on structural similarity between relational Data Marts that merely enable drill-across queries. Clearly, the disadvantage of all such systems is that the user is responsible for repairing heterogeneity among the elements of multi-dimensional schemas within the query.

In contrast, relational query languages such as SQL have been rarely used for multi-dimensional data integration. Only few approaches have addressed extensions of standard SQL for OLAP applications in distributed Data Warehouse environments. For example, SQL$_M$ formally defines an OLAP data model and introduces a set of transformation operators. The SQL$_M$ language supports irregular hierarchies in dimensions of stand-alone Data Warehouses—e.g., multiple aggregation paths. Although the approach allows the user to include additional XML data extending the query scope, SQL$_M$ does not support matches between multiple Data Mart schemas [Pedersen et al., 2002a]. Other languages that belong to this category are nD-SQL [Gingras and Lakshmanan, 1998] and FISQL [Wyss and Robertson, 2005]. These two approaches have been explained earlier in this Subsection (pp. 32).

Algorithms for query processing in Distributed DWs have brought interesting findings that are also relevant for Federated DWs. In particular, two approaches named Skalla [Akinde et al., 2003] and DWS-AQA [Bernardino et al., 2002] have independently introduced the idea of globally replicating consolidated dimensions. The experiments conducted in both these approaches indicate that replicated dimensions improve the overall response time of queries against the global schema. Query answering in such settings is reduced to the retrieval of facts because all dimension data is globally available.

3.5 Multi-dimensional Schema Integration

This Section summarizes previous effort within the Data Warehousing community towards the integration of multi-dimensional data sources, overcoming heterogeneity among the individual schemas. Multi-dimensional schema integration denotes the construction of an *integrated multi-dimensional schema* from several, heterogeneous multi-dimensional schemas. While the previous Subsection 3.4.5 surveyed approaches for loosely coupled integration of multi-dimensional data, the following discussion aims at investigating how Data Warehouses and Data Marts can be integrated under a consolidated, global schema (irrespective of physical vs. logical integration of the multi-dimensional *data*).

As pointed out before, the multi-dimensional data model is more complex than the relational model, due to the distinction between the fact entity and the dimension entity [Golfarelli et al., 1998]. Moreover, facts *logically depend* on dimensions in the multi-dimensional model (cf. Chapter 4, pp. 51). Thus, analyzing multi-dimensional schemas for heterogeneity with well-established methods [Lee et al., 1995, Kim and Seo, 1991] is insufficient. Instead, multi-dimensional schema mappings have to consider (1) the attributes in facts and dimensions, (2) the dependencies between facts and dimensions, and (3) the aggregation hierarchies.

In order to reduce complexity, multi-dimensional data sources should be integrated at the level of Data Marts, instead of complete Data Warehouses. As argued by [Kimball, 2002], even when merging only company internal, independently developed Data Marts across divisions, the necessary integration ("data migration") projects often cost several months or even years. The simplest is the migration of existing, heterogeneous Data Marts into a new Data Warehouse with an integrated global schema, translating all data to the integrated schema (cf. "schema integration" in Section 3.3, pp. 29). It is common, however, that privacy concerns restrict the necessary full access to the source Data Marts. Moreover, in the case of business cooperations, preserving the autonomy source Data Marts will likely be necessary. In such cases, the better solution is to integrate Data Marts at the logical schema level—i.e., to build a federation in the spirit of data integration (cf. the previous Section 3.4).

Integration of multi-dimensional data sources comprises two tasks. First, the dimensions have to be consolidated across the sources (*dimension integration*). Second, the facts stored in cubes of the multi-dimensional data sources are integrated and related to the integrated dimensions (*fact integration*). Thus, both the tasks of multi-dimensional schema integration and their order corresponds to the logical dependency of facts on dimensions that is inherent to the multi-dimensional model [Golfarelli et al., 1998].

Surprisingly, to the best of our knowledge, there are only few approaches that investigate multi-dimensional schema integration systematically. Dimension integration—the first sub-problem of multi-dimensional schema integration—has been addressed by some recent projects. In contrast, fact integration—the second sub-problem—is often oversimplified. Most authors treat facts as relational tables, for which the classification of possible conflicts is well defined and understood (see [Lee et al., 1995, Kim and Seo, 1991]). Indeed, facts in Data Warehousing can be integrated with techniques developed for relational databases—e.g., [Zhao and Ram, 2007]—and then connected to the consolidated dimensions. However, this simplifying assumption ignores the interplay of dimension integration and fact integration.

Due to the logical dependency between facts and dimensions in the multi-dimensional model, conversions of dimensions and facts influence each other. For example, changing the base level of a dimension implies a roll-up of all facts connected with this dimension. Moreover, *schema–instance conflicts* between multi-dimensional data sources (cf. pp. 62 in Chapter 5) are basically a variant of *concept mismatch* (see e.g. [Garcia-Molina et al., 1997]), i.e. different conceptualizations of real-world concepts in dimension and fact entities across several sources. These conflicts affect the *dependency* between a fact and dimension entity, which again indicates the interplay between dimension integration and fact integration.

Therefore, dimensions and facts must be integrated *conjointly*, using a systematic methodology that considers the interplay between the two sub-problems. If, however, dimension integration and fact integration are tackled in an isolated way, such conflicts cannot be repaired properly. Even worse, it is possible that schema–instance and other conflicts remain undetected.

Only [Tseng and Chen, 2005] have approached dimension integration and fact integration in a conjoint and systematic methodology—i.e., taking into account the logical dependency of facts to dimensions in the multi-dimensional model. Most previous work addresses either dimension integration or fact integration as isolated problems. As our analysis of heterogeneity in the multi-dimensional model points out, however, the isolated integration of dimensions and facts is insufficient to overcome all heterogeneity among multi-dimensional data sources comprehensively and systematically (cf. Chapter 5).

Tseng and Chen have proposed the use of XML technologies in a tightly coupled integration architecture for multi-dimensional data sources. Their basic idea is to generate two XML files per cube—one for the *cube meta-data*, and the other for *fact data*—and store these files in the native XML database system Tamino [Schöning, 2003, Schöning, 2001]. Subsequently, using XQuery statements [Chamberlin, 2002] they map the sources' XML fact data to a global, integrated multi-dimensional schema. Thus, the OLAP users simply query the global schema. All heterogeneity among the multi-dimensional data sources remains transparent from the OLAP applications [Tseng and Chen, 2005].

The outstanding contribution of Tseng and Chen's XML-based approach is the first definition of a *systematic methodology* for multi-dimensional schema integration. In particular, [Tseng and Chen, 2005] classify conflicts in the multi-dimensional data model, based on the relational conflict taxonomy given by [Lee et al., 1995]. The integration methodology addresses these conflicts systematically, taking into account the dependency between facts and dimensions in the multi-dimensional model [Tseng and Chen, 2005].

Torlone et al. have presented a systematic methodology especially for dimension integration, and provide the visual tool "DaWaII" that implements their methodology [Torlone, 2008]. The contributions of this approach are twofold. First, Cabibbo and Torlone introduce an *algebraic query language* that allows to transform the dimensions of multi-dimensional data sources [Cabibbo and Torlone, 1998]. Second, they formally define *desirable properties* of dimension mappings, i.e. *coherence*, *soundness* and *consistency* [Cabibbo and Torlone, 2005]. The DaWaII tool, in turn, allows the user to test dimensions of various multi-dimensional sources for compatibility, and specify mappings between compatible dimensions. Subsequently, the mappings can be checked whether they satisfy the desirable properties [Cabibbo et al., 2006, Torlone and Panella, 2005].

The DaWaII approach studies two different options for integrating facts of multi-dimensional data sources on the mapped dimensions, "loosely coupled" and "tightly coupled". The *"loosely coupled"* integration method refers to drill-across queries among cubes with at least one mapped dimension in common. Following the *"tightly coupled"* integration method, the DaWaII tool creates a materialized view over the complete, integrated multi-dimensional sources

[Torlone, 2008]. For both cases, however, the authors leave open the question of how to repair possible conflicts among the facts.

In order to facilitate the multi-dimensional schema integration process, Banek et al. apply schema matching techniques (see [Rahm and Bernstein, 2001] and Section 3.3) on the aggregation levels of dimensions in data cubes [Banek et al., 2007]. Their approach assumes a scenario with only restricted access to the sources (e.g., medical databases, or business cooperations). Hence, the matching algorithm exploits exclusively structural information (i.e., the meta-data of multi-dimensional schemas). To calculate *similarity values* between aggregation levels, the algorithm compares attribute names among dimensions linguistically, based on WordNet [Miller, 1995]. In a second phase, Banek's matching algorithm produces mappings between aggregation levels based on their similarity, while preserving the functional ordering of dimension data [Banek et al., 2007].

3.6 Model Driven Architecture (MDA)

The *Model Driven Architecture (MDA)* is an official standard published by the Object Management Group (OMG), encouraging formal software models at different levels of abstraction. In particular, MDA provides four model layers that address the complete life cycle of software development (i.e., specification, design, implementation, deployment, version management, and so forth). MDA supports both, *forward engineering* and *reverse engineering* of models with automated transformations between the abstraction layers [(OMG), 2003b].

MDA's model layers are designed to separate the specification of system functionality from its implementation. The *functionality* of software systems is normally stable and valid across various platforms, whereas the *implementation* typically changes repeatedly over time (e.g., to correct bugs, or to port the system into a new programming language or paradigm). Moreover, system implementation is platform-specific—i.e., only plausible for one particular target platform or language. Following this consideration, the relatively stable, functional specifications reside on different model layers of MDA than the unsteady specification of implementation for a certain platform [Gruhn et al., 2005, Kleppe et al., 2003].

Thus, the most abstract layer of MDA is the *Computation Independent Model (CIM)*, generically describing the system's functionality as black box within its environment. Next, the *Platform Independent Model (PIM)* specifies the precise functional requirements for the system, but without any specific information on the technology used to realize it. On the next layer, the *Platform Specific Model (PSM)* translates the PIM into an implementation model for some particular target platform, including the exact technological details on how to implement the system's functionality within the target platform. Finally, the implementation itself—i.e., the actual source code—is generated for each PSM in the chosen platform or language [(OMG), 2003b, Kleppe et al., 2003].

The core idea behind MDA is the automated *translation* of models. Ultimately, this means that even the source code of software systems can be generated automatically. Automated translation between the MDA model layers is motivated by the possible better re-use of the platform independent software specification [Kleppe et al., 2003]. OMG provides another standard, the transformation language *Query/View/Transformation (QVT)*, for that purpose [Mazón et al., 2006, Mazón et al., 2005].

MDA has been applied successfully in the Data Warehousing domain, particularly for the conceptual design of Data Warehouses and Data Marts [Mazón and Trujillo, 2008]. For example, Mazón and Trujillo demonstrate the usefulness of MDA concepts for the conceptual design of multi-dimensional schemas in two different scenarios, *data-oriented* development [Mazón et al., 2006, Mazón et al., 2007] and *goal-oriented* development [Mazón and Trujillo, 2007, Glorio and Trujillo, 2008]. While the data-oriented approach reconciles several operational data sources [Mazón et al., 2006], the goal-oriented approach allows the modernization of existing DWs through reverse engineering and subsequent evolution of their conceptual schemas [Mazón and Trujillo, 2007]. Another application of the goal-oriented approach, model-driven design of Data Marts on top of the corporate DW, is studied in [Pardillo and Trujillo, 2008]. Furthermore, [Pardillo et al., 2008] have generated cube meta-data for the use in OLAP applications from the conceptual multi-dimensional model.

Generally, MDA does not restrict the modelling language to be used for specifying PIMs and PSMs. However, it is advantageous to use MOF-compliant languages such as the *Unified Modelling Language (UML)* [(OMG), 2009] for that purpose—for two reasons. First, UML is a widely used, general purpose and standardized modelling language with rich tool support. Second, UML is extensible for arbitrary application domains. It provides several extension mechanism (e.g., profiles) that allow defining highly specialized languages. Therefore, the UML is typically used to represent PIMs and PSMs in model-driven software engineering [Glorio and Trujillo, 2008]. In the next Section, we will survey the use of UML for the Data Warehousing field.

3.7 DW Modelling and Design with UML

UML 2.x has become the *de facto* standard for information systems modelling [Larsen et al., 2009]. It is ideally suited for representing Data Warehouse schemas because UML's packaged structure corresponds with the nature of DW models that are also organized hierarchically. Moreover, the UML notation is easy to understand and well-known, while its extension mechanisms allow to customize the UML semantics for arbitrary application domains. Thus, several authors have extended the UML with profiles for various facets of DW modelling and design, for example conceptual schemas [Luján-Mora et al., 2006, Prat et al., 2006, Zubcoff and Trujillo, 2007], physical schemas [Luján-Mora and Trujillo, 2006, Luján-Mora and Trujillo, 2004], or security constraints on Data Marts [Fernández-Medina et al., 2007].

Utilizing the UML for standardization of multi-dimensional concepts has been an important prerequisite for the implementation of powerful DW design tools. The *Common Warehouse Meta-model (CWM)* is an official standard for DW modelling with broad industry backup [(OMG), 2003a]. CWM is MOF-compliant so that it supports a model-driven approach for the development and interchange of DW models, based on the UML and XMI standards [Poole, 2003].

Prat et al. [2006] propose a UML-based design method for multi-dimensional DW schemas spanning the three abstraction levels—conceptual, logical and physical—recommended by ANSI/SPARC [Navathe and Elmasri, 2004]. Their *Unified Multi-dimensional Meta-model* integrates common concepts of the many multi-dimensional data models proposed in literature [Prat et al., 2006]. The Unified Multi-dimensional Meta-model is the most comprehensive attempt of identifying the commonly accepted fundamental concepts behind conceptual DW models. While the Meta-model of [Prat et al., 2006] provides a comprehensive palette of DW properties, it also has two shortcomings. First, it does not consider the CWM standard [(OMG), 2003a]. Second, diagrams of multi-dimensional schemas based on the Unified Meta-Model tend to be too complex to provide for good readability.

In contrast, Luján-Mora et al. define conceptual DW schemas in package diagrams with several abstraction layers [Luján-Mora et al., 2006]. Level 1 and 2 of their approach represent cubes within Data Warehouses, respectively the facts and dimensions of each DW. Every dimension, in turn, is also contained in its own package. Finally, level 3 specifies the details for every dimension package. At each of the three modelling levels, OCL constraints [Warmer and Kleppe, 1999] are used to ensure the correct use of the stereotypes and tagged values defined by the proposed UML profile [Luján-Mora et al., 2006].

Data mapping diagrams with UML for conceptual modelling of ETL processes is an interesting approach closely related to our own work. Luján-Mora et al. use stereotyped UML class and package diagrams to represent data flows from heterogeneous, operational data sources to the DW at different levels of abstraction [Luján-Mora et al., 2004]. While their framework specifically repairs attribute and table conflicts between relations [Kim and Seo, 1991], it does not address multi-dimensional heterogeneity, identified in [Berger and Schrefl, 2006].

3.8 Summary and Requirements for the FedDW Approach

In the previous sections of this chapter, we surveyed the enabling technologies for Federated Data Warehouses, and analyzed previous work in related research areas. Based on the current state of knowledge in the field of Federated DW systems we now formulate the requirements on our approach named "FedDW". According to [Tseng and Chen, 2005], the problems that must be addressed by multi-dimensional data integration are the following:

- *Formal definitions* for the "building blocks" of Data Warehouses and Data Marts—i.e., a formal model for multi-dimensional data sources.
- Recognition and classification of *semantic heterogeneity* in the multi-dimensional data model.

As indicated by the current state of the art, the integration of autonomous Data Marts must also address the following challenges:

- Decide whether to physically migrate multi-dimensional data, or logically integrate the Data Mart schemas.
- Provide an appropriate query language across the autonomous Data Marts that supports multi-dimensional data.
- Choose the optimal paradigm for the integration of multi-dimensional schemas and data (i.e., global-as-view versus local-as-view versus both-as-view).
- Repair heterogeneity among multi-dimensional sources conjointly, addressing the interplay between dimension integration and fact integration.

From previous work in the Data Warehousing and related research fields, we draw the following four, most significant conclusions for Data Mart integration:

Logical integration is clearly preferable for the huge data sets in Data Marts and Data Warehouses. As indicated by [Kimball, 2002], the physical migration of Data Mart data is too elaborate in most practical settings. Even if developed in one and the same company, the migration of Data Marts into an integrated Data Warehouse costs several months to years.

Current multi-database query languages are not sufficiently powerful for multi-dimensional integration. Multi-system query languages have brought interesting results concerning the transformation of data into meta-data, or vice versa. However, all of the existing approaches are optimized for relational data. Multi-dimensional data is only supported by nD-SQL (p. 32) and FISQL (p. 32), although merely for relatively simple OLAP operations.[1]

FISQL introduces powerful data and schema transformations—e.g., variables ranging over attribute names and other meta-data—that are also needed when integrating multiple heterogeneous Data Marts. The nD-SQL language even supports multi-dimensional data [Gingras and Lakshmanan, 1998], but its transformation features are less powerful than in several other of the approaches summarized above. Moreover, aggregation hierarchies in dimensions cannot be represented in nD-SQL's underlying data model. Unfortunately, this behavior seems too restrictive in most practical scenarios.

Both-as-view mappings combine the advantages of two previous techniques: global-as-view and local-as-view. In particular, the BGLaV approach has introduced so-called "source-to-target" mappings [Xu and Embley, 2004]. Such mappings allow to combine the efficient generation of query plans for answering user queries over the global Data Mart schema (GAV) with the extensibility of the federated system (typical for LAV). It is worth noting, though, that this approach still does not eliminate the need to propagate changes of the global schema. That means, if the global schema evolves, all existing mappings must be updated accordingly.

Finally, the integration of Data Marts into a Federated system clearly calls for more powerful methodological frameworks than traditional data integration. Most closely related to our work are the XML-based integration framework of [Tseng and Chen, 2005], and the "DaWaII" approach of [Torlone, 2008]. Based

[1]For comprehensive reviews of the query languages analyzed in Subsection 3.4.1 we refer to [Brunneder, 2008, pp. 66–95] and [Wyss and Robertson, 2005].

on the review given in Section 3.5 (see pp. 39) we identify the following shortcomings of these approaches:

1. *Incomplete classification of heterogeneity*:

   Tseng and Chen's framework presents the most comprehensive and systematic methodology for conjoint integration of dimensions and facts among autonomous cubes so far. However, the authors do not address *overlapping facts*. Moreover, they leave aside the possibility of *cube-to-dimension conflicts* (denoted *schema–instance conflicts* in our classification; see Chapter 5, pp. 59).

   Torlone's DaWaII approach considers cube-to-dimension conflicts within multi-dimensional schemas, supporting the conversion of measure attributes to dimensional attributes, and vice versa [Cabibbo and Torlone, 1998]. However, these operations are merely *schema–schema transformations*. Thus, the schema–instance conflicts we mentioned above are not addressed in the DaWaII approach either.

2. *Isolated integration of multi-dimensional sources*:

   Torlone's "DaWaII" approach addresses only dimension integration as *isolated problem* while ignoring fact integration. However, the logical connection between dimensions and facts is fundamental to the multi-dimensional model (see Section 3.5 and [Golfarelli et al., 1998]). Hence, heterogeneity among multi-dimensional data sources potentially affects both their dimensions and facts. In turn, this observation calls for the conjoint integration of dimensions and facts among multi-dimensional data sources with a methodology that considers their logical interdependency.

3. *Materialization of factual data (large data sets)*:

   Tseng and Chen's framework is based on complete *materialization* of multi-dimensional cubes in native XML databases. While such an approach is clearly effective for reasonably small amounts of data, it also raises two problems, a fundamental one and an architectural one:

   (a) View materialization in Data Warehousing always introduces the view refreshment problem [Lee et al., 2007, Zhou et al., 2007, Gupta and Mumick, 2006]. Therefore, the usefulness of Tseng and Chen's XML-based integration framework much depends on an efficient *view maintenance* mechanism. The authors, however, leave this question aside in [Tseng and Chen, 2005].

   (b) In the case of very large Data Marts, the materialization of fact data in an XML database is questionable for two reasons. First, this approach wastes storage space—clearly more efficient would be the materialization of aggregated facts. Second, current native XML query processors suffer from high memory consumption, which prevents them from querying large XML documents efficiently [Härder et al., 2007, Bonifati and Cuzzocrea, 2007]. Indeed, current research indicates a trend towards combining relational with XML-native techniques to improve performance of data querying and management in very large XML documents [Gou and Chirkova, 2007].

Analogously, Torlone et al. propose a “tightly coupled” variant of their integration approach, i.e. the complete materialization of multi-dimensional sources—according to dimension mappings—in relational Data Marts [Torlone, 2008, Cabibbo and Torlone, 2005]. Again, the same objections as above hold against this approach: (i) view maintenance problem, (ii) waste of storage space. The authors do not consider these points at all, although the view maintenance mechanism would be particularly important to evaluate the viability of the approach.

4. *Inter-source 1:1-mapping paradigm*:

   DaWaII defines 1:1-mappings, i.e. it integrates dimensions across several multi-dimensional data sources in pairwise manner, similarly to global-as-view data integration (cf. 3.4.2). Using these inter-source mappings, the DaWaII tool generates merged dimensions “that can be either attached to an imported Data Mart or exported to an external system” [Torlone, 2008, page 93]. As far as the tool is concerned, this approach enables a clear user interface design. From the methodological viewpoint, however, it raises the question of whether DaWaII is able to integrate more than two Data Marts properly. The authors explain exclusively the rather simplistic case of only two Data Marts being integrated [Torlone, 2008].

   Generally, pairwise mappings defined directly between data sources—i.e., in data integration architectures without a stable, integrated global schema—are disadvantageous [Lenzerini, 2002]. In particular, integrating n Data Marts with DaWaII requires $\frac{n(n-1)}{2}$ mappings (since they are unidirectional). This exponential number of necessary mappings applies to the worst case, i.e. a strict pairwise mapping strategy between multi-dimensional data sources.

   From the information given by the authors, however, it seems reasonable to assume that the DaWaII tool supports integration of three or more multi-dimensional sources according to the *“ladder strategy”*—i.e., step-wise 1:1 integration of sources into an evolving, intermediate result [Batini et al., 1986, page 343]. In this case, the number of mappings necessary to integrate n Data Marts with DaWaII would reduce to only $n-1$. This number corresponds to the necessary iterations, each producing an intermediate result.

   Nevertheless, the ladder strategy also has two disadvantages: *ambiguity* and *waste of storage space* (albeit one excludes the other). On the one hand, if following the “loosely coupled” integration approach, one must suspect that the incremental definition of intermediate results possibly produces ambiguity, i.e. contradictory mappings. The authors, however, leave aside whether such contradictions between incrementally defined mappings must be considered in their approach, and the possible consequences. On the other hand, the “tightly coupled” integration method of DaWaII generates fully materialized views of the facts and dimensions according to the mapping result. Considering the intermediate results of every mapping step, the full materialization approach wastes substantial amounts of storage space since the intermediate materialized views are worthless for the users. Deleting the intermediate views, however, would mean losing the mappings.

> Under these considerations, extending a system of integrated multi-dimensional sources defined with DaWaII seems also difficult. In the worst case, the integration of an additional Data Mart renders all pre-existing mappings obsolete. In the best case, existing mappings remain valid, but the new mapping step produces another complete materialized view of the integrated multi-dimensional data sources.

The above considerations give strong evidence that *federated* architectures with *source-to-target* mappings, using a "virtual" instance of the global schema with distributed query processing, are the ideal solution currently available for the integration of multi-dimensional data sources. Furthermore, Dina Bitton gives a strong statement in favor of federated architectures among autonomous Data Warehouses: "*Virtualize data across multiple Warehouse boundaries and virtualize new Data Marts.* [...] Another scenario where a virtual Data Mart is a great solution is where a new external data source [...] needs to be integrated with an existing Data Warehouse." [Halevy et al., 2005, page 4].

Indeed, a federated architecture is especially well suited for Data Warehousing environments, for which *huge data sets* and *high selectivity* of analytical queries are characteristic. The huge amounts of data are the main reason why materialization has not proven particularly effective for multi-dimensional source integration, as we have discussed above. In turn, the high selectivity of typical OLAP queries reduces the amount of data shipped between various sources if virtual cubes with distributed query processing are realized. True federations of multi-dimensional data sources cannot be established using the approaches proposed to date, though.

Consequently, we define the following requirements for Federated Data Warehouse systems to ameliorate previous approaches to multi-dimensional source integration, addressing the open research challenges indicated above:

R 1. Extend the definition of multi-dimensional heterogeneity, based on a system independent, conceptual model of multi-dimensional data.

R 2. Define an architecture that provides a stable, global schema over multi-dimensional data sources, and enables the sources to remain autonomous.

R 3. Develop a methodology for conjoint integration of dimensions and facts among autonomous, multi-dimensional data sources, systematically considering the interdependencies between dimensions and facts inherent to the multi-dimensional model.

R 4. Define a declarative conversion language addressing all classes of heterogeneity in the multi-dimensional model, that suits as mapping mechanism between sources and the global schema of the integration architecture.

R 5. Make these mechanisms easy to use by providing tools with graphical user interfaces for the business analysts. Provide high-level operators upon the data model, so that the actual source code of mappings can be generated by the system (model-driven approach).

In the following parts of the thesis, we present how the FedDW approach to Federated Data Warehouse systems addresses the above requirements.

Part II

Architecture and Concepts of Federated Data Warehouses

Chapter 4

Conceptual Multi-Dimensional Data Model

Contents

This chapter introduces a conceptual model for multi-dimensional data that is rich enough to represent all properties of dimensions and facts commonly modelled in the cube schemas of Data Marts. Although numerous multi-dimensional data models have been proposed in previous literature, none of them is semantically rich enough to represent all properties and peculiarities of cubes that have been identified in Prat et al.'s Unified Multi-dimensional Model. The definitions in this chapter both give a formal foundation for the building blocks of multi-dimensional data structures, and clarify essential terminology used throughout the thesis.

The chapter starts with the basic elements of the multi-dimensional data model. If defines Data Marts, Dimensions with members and Cubes with cells. These concepts are complemented with functions and additional properties of dimensions and their members. Thus, the conceptual data model is the prerequisite for identifying possible sources of heterogeneity among the schemas and instances of autonomous Data Marts.

This Chapter discusses the conceptual model of multi-dimensional data in autonomous Data Marts. It represents the object oriented, *Unified Multi-dimensional Meta-model* proposed in [Prat et al., 2006], which is a logical meta-model, at the conceptual level. The Unified Multi-dimensional Meta-model is the most comprehensive definition of typical and fundamental properties for multi-dimensional data sources.

Among the many multi-dimensional models proposed in literature (see [Vassiliadis and Sellis, 1999, Blaschka et al., 1998] for a summary), none supports all of the properties identified by [Prat et al., 2006]. Moreover, the *Common Warehouse Metamodel (CWM)*—another candidate for representing multi-dimensional Data Marts—has been criticized for being too generic to accommodate all peculiarities of conceptual multi-dimensional schemas [Medina and Trujillo, 2002]. Hence, the FedDW approach employs the conceptual multi-dimensional model introduced in in [Berger and Schrefl, 2008].

In FedDW's architecture (Chapter 6; depicted in Figure 6.1, page 83), the conceptual multi-dimensional data model is used as the "canonical model" of federated systems [Sheth and Larson, 1990] to represent the schemas of local and autonomous Data Marts. The FedDW conceptual data model supports the essential concepts *fact* and *dimension*, refining a previous proposal [Cabibbo and Torlone, 2005] with extended properties of dimensions. As any model at the conceptual level, it allows the definition of multi-dimensional schemas independent of any implementation aspects.

Intuitively, a Data Mart defines one or more measure variables within facts, categorized by some dimensions that in turn are organized in hierarchies of aggregation levels. The "cube" is the central data structure of the multi-dimensional model. It acts as container that links facts to one or more dimensions, thus comprising a multi-dimensional space for storing factual data. Facts and dimensions consist of both their schema *and* the corresponding instances. For dimension instances, we use the commonly accepted term *dimension members* (or *members* for short) [Vassiliadis and Sellis, 1999] throughout the thesis. In contrast, the term *fact* may refer either to the schema or the instances of multi-dimensional facts. Whenever necessary, we will speak of the "fact schema" respectively "fact instances" to distinguish the schema from the instance level.

Basically, the model concepts *data mart*, *cube* and *dimension* (Definitions 4.1 and 4.9) define a local Data Mart as the *universe of discourse* for the declaration of dimensions, i.e. the constructs *cube* and *dimension* are first-class citizens of the model. The FedDW conceptual multi-dimensional data model extends the so-called $\mathcal{MD}$ model [Cabibbo and Torlone, 2005], introducing additional properties of the *dimension* construct. These model extensions allow to represent all properties contained in the Unified Multi-dimensional Meta-model [Prat et al., 2006], and to cope with *all* multi-dimensional heterogeneities identified and analyzed in [Berger and Schrefl, 2006] (see Chapter 5). In particular, the additional features of FedDW's data model are the following:

(1) The level schema of dimensions supports not only level attributes, but also non-dimensional attributes (Definition 4.2).

(2) The functions *level* and *members* specify the relationship between the schema and instances of dimensions, i.e. between the hierarchy and roll-up functions of levels (Definitions 4.4 and 4.5).

For the definitions that follow in this Chapter, let $\{\tau_1, ..., \tau_m\}$ be a finite set of data types (e.g., integers) with their domain defined by function $dom(\tau)$.

4.1 Data Marts

From an object oriented viewpoint, the *Data Mart* is the top-level container for the other multi-dimensional constructs, i.e. dimensions and cubes. Intuitively, a Data Mart provides a "view" on some well-defined subset of the dimensions and cubes defined within the scope of the underlying Data Warehouse. We formalize the model constructs dimension and cube in Sections 4.2 and 4.4, respectively.

The corporate Data Warehouse can itself be regarded conceptually as Data Mart. As explained in Chapter 3 (see 3.1.2, p. 26), the definition of Data Marts on top of the corporate DW allows to address the information needs of some particular department, or to ensure the privacy of sensitive fact data (e.g., anonymous patient data in health care; or sales figures on weekly basis of retail trade companies). For the integration of autonomous, multi-dimensional data, though, the distinction between Data Mart or Data Warehouse as the source of data is irrelevant. Therefore, the terminology and formal notation of the FedDW approach always assume that Data Marts are to be integrated.

Definition 4.1 (Data Mart):
A *Data Mart* $DM = \{C_1, ..., C_n; D_1, ..., D_m\}$ $(n, m > 0)$ consists of a non-empty set of Cubes C and a non-empty set of dimensions D. ■

Example 4.1: As depicted in Figure 2.1 on page 16, the health care organization of our case study defines two Data Marts: dwh1 and dwh2. In turn, each of the two DMs contains two Cubes, named treatment and medication. Moreover, both dwh1 and dwh2 define several dimensions. Using the notation given above, we obtain the following descriptions of the case study's DMs: dwh1 = $\{$*treatment, medication; physician, method, date_time, drug, patient*$\}$, dwh2 = $\{$*treatment, medication; cost_cat, method, date_time2, drug, patient, date_time*$\}$.

Assume that dwh1 and dwh2 are defined on top of corporate Data Warehouses $DW_1 \supseteq dwh1$ and $DW_2 \supseteq dwh2$, hiding some details so to meet legal obligations for the privacy of personal patient data. The complete schemas of DW_1 and DW_2 remain unknown, but are irrelevant for the integration of the dwh1 and dwh2 Data Marts. Thus, the Data Mart schemas of dwh1 and dwh2 represent the health insurances' *export schema*, i.e. their interface that defines which part of the corporate DW is accessible for business partners. ◇

4.2 Dimensions

In the FedDW conceptual data model, dimensions are first-class citizens. This means that dimensions and their properties are defined in the context of the underlying Data Mart, and that several Data Marts—or even several cubes within one Data Mart—can share dimensions. In order to use dimension D in some Data Mart DM, it is necessary to "import" D—i.e., referencing D in the declaration of DM (cf. Definition 4.1). Every dimension consists of both its schema (Definition 4.2) and its instance or members (Definition 4.3).

Definition 4.2 (Dimension Schema):
Every *Dimension* $D \in \{D_1, ..., D_m\}$ of DM has the *dimension schema* $S_D = (L_D, S(L_D), H_D)$ containing (I) the finite, non-empty set of Levels $L_D = \{l_1, ..., l_j, ..., l_m, l_{all}\}$ with level schema $S(L_D) = \{S_{l_1}, ..., S_{l_m}\}$ and (II) the roll-up hierarchy $H_D \subseteq L_D \times L_D$, where H_D forms a lattice. If $l, k \in H_D$, we write $l \mapsto k$ and we say l "rolls-up to" k.

The *level schema* $S_l \in S(L_D)$ of some level $l \in L_D$ is an attribute schema $l(K, N_1, ..., N_k)$ with name l, key K (the dimensional or "roll-up attribute") and optional non-dimensional attributes (or "N-attributes") $N_1, ..., N_k$, denoted alternatively as $l.K, l.N_1, ..., l.N_k$. Every attribute $K, N \in S_l$ has an associated data type τ as domain, denoted $dom(l.K) = dom(\tau_{l.K})$ and $dom(l.N) = dom(\tau_{l.N})$. ■

In the remainder of the thesis, we will often use square brackets to denote level names of dimension schemas wherever this allows for an easier presentation (e.g., [month] instead of simply month). The same notation of level names in square brackets is also used within cube schemas to connect facts with aggregation levels (Definition 4.9), but in that case the brackets are mandatory.

Example 4.2: The schema of dimension drug in Data Mart dwh1 is given by (1) L_{drug} = {*[l_drug], [l_manufacturer]*}, (2) $S(L_{drug})$ = {*l_drug (drug, pkg_size), l_manufacturer (manufacturer, country)*}[1] and (3) $H_{drug} = \{l\_drug \mapsto manufacturer\}$. Figure 2.1 depicts the dimensions of the example DMs and all their attributes—dimensional and non-dimensional—but not the level names. Thus, level name *l_drug* and all other level names are not shown there. ◇

Definition 4.3 (Dimension Instance):
A *dimension instance* $d(S_D)$ over schema S_D consists of (I) its name d, identifying the instance, (II) its set of *members* V_d with each $v \in V_d$ being a tuple over a level schema $S_l \in S(L_D)$, and (III) the family of "roll-up" functions ρ_d between the member subsets according to H_D (see Definition 4.5). ■

Example 4.3: The instance of dimension drug in Data Mart dwh1 is named dwh1::drug. Its member set V_{drug} consists of the tuples {('A', ...), ('B', ...), ('C', ...)} shown in Figure 2.3 on page 18. This corresponds to the union of all tuples over *l_drug (drug, pkg_size)* (i.e. the drug-members) with all tuples over *l_manufacturer (manufacturer, country)* (i.e. the manufacturer-members). ◇

4.3 Functions and Properties of Dimensions

Before defining the concept of a data cube, we need to specify additional functions and properties of dimensions. In particular, we introduce *dimension functions* and *roll-up functions* ρ_d in the following definitions. Later in this thesis, we will use these functions to formally define heterogeneities among dimension schemas and members. Moreover, we require the dimension functions to define cubes precisely. To improve clarity within the forthcoming examples, we explicitly designate the Data Mart in all symbol subscripts, i.e. we use, for example, "$H_{dwh1::drug}$" instead of "H_{drug}".

[1]Notice that Example 4.2 uses prefix 'l_', denoting level names, in order to avoid confusion of identical dimension *level* names and level *attribute* names.

Definition 4.4 (dimension functions):
Let D be a dimension with schema S_D, level schema $S(L_D)$ and instance $d(S_D)$. We define the following functions over D:

- $level : V_d \rightarrow L_D$ returns the level l corresponding to a given $v \in V_d$.
- $members : L_D \rightarrow 2^{V_d}$ returns the set $T = \{v \in V_d \mid level(v) = l\}$ containing all members $v \in V_d$ that belong to some level l. ■

Example 4.4: The function expression $members(dwh1::l\_drug)$ returns all $v \in V_{drug}$ that belong to level *l_drug*, i.e. all tuples over level schema *l_drug(drug, pkg_size)*. Evaluating this function over dimension dwh1::drug_dim as shown in Figure 2.3 (page 18), we obtain three members: {('A', '25 pcs.'), ('B', '25 pcs.'), ('C', '250 ml.')}. The *level* function is the inverse of the *members* function. Accordingly, function expression *level*(('A', '25 pcs.')) results in level *l_drug*. ◇

Aggregation levels are one of the fundamental concepts in the multi-dimensional model [Bauer and Günzel, 2006, Golfarelli et al., 1998]. Hierarchies of aggregation levels in dimensions allow to view fact data with varying precision. The following Definitions 4.5, 4.6 and 4.7 detail the properties of aggregation hierarchies formally. Finally, Definition 4.8 establishes an "ordering" relationship among aggregation levels in hierarchies.

Definition 4.5 (roll-up functions):
Let $l, k \in L_D$ be two levels, $l \neq k$, of a dimension D. The roll-up function $\rho_D^{l \mapsto k}$ is defined for each pair $l, k \in H_D$. The family of roll-up functions ρ_d contains all $\rho_D^{l \mapsto k}$ defined in this way. ■

Example 4.5: Dimension dwh1::drug defines only two levels, *l_drug* and *l_manufacturer* (see Example 4.2), so that the hierarchy H_{drug} is relatively simple ($\{drug \mapsto manufacturer\}$). Thus, $\rho_{dwh1::drug}$ contains only a single roll-up function: $\rho_{dwh1::drug}^{drug \mapsto manufacturer}$. ◇

Definition 4.6 (base level of hierarchy):
The *base level* of hierarchy H_D, denoted l_0^D, representing the finest grain of the dimension's members, is the bottom element of the lattice H_D. The *all-level*, denoted l_{all}^D or l_{all} for short, is the top element of the lattice H_D. ■

Example 4.6: The base level of hierarchy $H_{dwh1::drug}$ (see the previous Example 4.2) is *[l_drug]*. Accordingly, the base level of hierarchy $H_{dwh1::date\_time}$ is *[l_day]* and so forth. Every hierarchy of the case study's dimensions in dwh1 and dwh2 contain the all-level l_{all}, that is implicit to every dimension and therefore not shown for any dimension in Figure 2.1. ◇

Definition 4.7 (roll-up consistency):
Let $l, k \in L_D$ be two levels, $l \neq k$, of a dimension D, and $T_l = members(l)$, $T_k = members(k)$. Roll-up function $\rho_D^{l \mapsto k}$ is consistent with the hierarchy H_D iff all members of level l roll-up to a member of level k, such that $\forall v \in T_l : \rho_D^{l \mapsto k}(v) = w \Rightarrow w \in T_k$. ■

Example 4.7: Roll-up function $\rho_{dwh1::drug}^{drug \mapsto manufacturer}$ of dimension dwh1::drug is consistent because it maps every member of level [l_drug] unambiguously to one member of level [manufacturer]. Assumed that dimensions dwh1::drug and dwh2::drug are merged, however, the roll-up function would be inconsistent because it is defined ambiguously for member 'B' among dwh1 and dwh2: $\rho_{dwh1::drug}^{drug \mapsto manufacturer}$('B') = 'Novartis' versus $\rho_{dwh2::drug}^{drug \mapsto manufacturer}$('B') = 'Bayer'.$\diamond$

Definition 4.8 (Fineness and coarseness of levels):
Let $l, \hat{l} \in H_D$ be two levels in some dimension D and operator $\mapsto^+$ denote the transitive closure of operator $\mapsto$ (rolls-up to, see Definition 4.2). If $l \mapsto^+ \hat{l}$, we say that l is "finer than" or "more fine-grained than" $\hat{l}$, and that $\hat{l}$ is "coarser than" or "more coarse-grained than" l. ■

Example 4.8: In dimension dwh1::date_time (see Figure 2.1 on page 16), level [day] is finer than [month] and [year]. Accordingly, level [year] is coarser than [month] and [day], and so on. $\diamond$

4.4 Cubes

Analogously to dimensions (Section 4.2), cubes are first-class citizens in the FedDW conceptual data model. That is, cubes and their properties are defined in the underlying Data Mart, and they can be used in several Data Marts. Every cube consists of both its schema (Definition 4.9) and its instance or cells (Definition 4.10).

Definition 4.9 (Cube Schema):
Every *Cube* C has the *cube schema* $S_C = \{A_C, M_C\}$ that is composed of (I) a set of dimension attributes $A_C = \{A_1, ..., A_n\}$, (II) a set of measure attributes $M_C = \{M_1, ..., M_k\}$. Each $A_i \in A_C$ $(1 \le i \le n)$ is linked to a level $l \in L_D$ of some dimension $D \in \{D_1, ..., D_m\}$ of the Data Mart, denoted $A_i[l]$. Each $M_j \in M_C$ $(1 \le j \le k)$ is linked to a data type τ_j.

The domain of each attribute $A_i[l] \in \{A_1, ..., A_n\}, M_j \in \{M_1, ..., M_k\}$ of S_C is defined as $dom(A_i) = members(l)$ and $dom(M_j) = dom(\tau_j)$. The number n of dimensional attributes in A_C is referred to as the *dimensionality of* C, whereby $n \le m$ (value m denotes the number of dimensions defined in the Data Mart). ■

Example 4.9: Cube schema medication in Data Mart dwh1 is given as (1) $A_{medication}$ = {patient [l_patient], drug [l_drug], date_time [day]} (dimension attributes) and (2) $M_{medication}$ = {qty, cost} (measure attributes). Thus, the dimensionality of dwh1.medication equals 3 (i.e. the cube is 3-dimensional).

The domains $dom(A[l])$ of the dimensional attributes $A \in A_{medication}$ match the domains $dom(l.K)$ of their respective linked level l (e.g., dom(drug[l_drug]) = dom(l_drug.K) = dom(drug), and so on). In turn, the domains of the measure attributes $M_{medication}$ are $integer$ and $float$ for qty and cost, respectively. As indicated by the resemblance of names, the dimension attributes associate cube dwh1::medication to the levels given in square brackets of the dimensions dwh1::patient, dwh1::drug and dwh1::date_time, respectively. $\diamond$

As stated in the above Definition 4.9, the level associated with some dimensional attribute must be given in square brackets (e.g., date_time[day]). All definitions of this thesis adhere to this notation. Within the examples in the forthcoming Chapters, however, we often write simply A instead of $A[l]$—when it is clear from the context that $A[l]$ refers to level l—to improve legibility.

Definition 4.10 (Cube instance):
A *Cube instance* $c(S_C)$ over schema S_C consists of (I) its name c, identifying the instance, and (II) a set of tuples over $\{[dom(A_1) \times ... \times dom(A_n)], [dom(M_1) \times ... \times dom(M_k)]\}$. A tuple $f \in c(S_C)$ is called a *"cell"* or *"fact"*. Moreover, with the *"coordinate"* of a cell we denote the values $[f(A_1), ..., f(A_n)]$ modelling the multi-dimensional context for the measures $[f(M_1), ..., f(M_k)]$. ■

Example 4.10: The cube instance of cube medication in Data Mart dwh1 is named dwh1::medication and consists of all the cells that are defined as tuples over the fact schema, as depicted in Figure 2.3. ◇

4.5 Names of Dimension and Cube Schemas

To denote elements in dimension schemas (see Definition 4.2) and cube schemas (see Definition 4.9), and to formalize naming conflicts the following definition introduces the name function of dimension and cube schemas.

Definition 4.11 (Name function of dimension and cube schemas):
Function $name : \{D \cup S_D.l \cup S_l.K \cup S_l.N \cup C \cup S_C.A \cup S_C.M\} \rightarrow string$ allocates some name, which is a character string, to each dimension D or cube C, or elements of the dimension schema S_D respectively cube schema S_C. In particular, the *name* function is applicable to level names $S_D.l$, roll-up attributes $S_l.K$, non-dimensional attributes $S_l.N$ of dimension schemas S_D, as well as to dimensional attributes $S_C.A$ and measures $S_C.M$ of cube schemas S_C. ■

Thus, intuitively, dimension levels $S_D.l$ and attributes $S_l.K, S_l.N$ of dimension schemas S_D, as well as $S_C.A, S_C.M$ of cube schemas S_C denote ontological concepts, whereas their respective names give "*labels*" to these concepts. In the forthcoming examples of the thesis, however, we omit explicit calls to the *name* function for better legibility whenever it is unambiguous. In most cases, we refer to the names of schemas or schema elements by giving the Data Mart name as prefix (e.g., dwh1::method refers to $name(dwh1.method)$).

Example 4.11 (Name functions): Let time denote the ontological concept represented by dimensions "dwh1::date_time" and "dwh2::date_time2" to illustrate the difference between the concept D and label $name(D)$ of a dimension schema. The *name*-function allocates the following names to the date dimensions of the treatment cubes (cf. Figure 2.1): $name(dwh1.time) =$ dwh1::date_time, $name(dwh2.time) =$ dwh2::date_time2. ◇

Chapter 5

Taxonomy of Conflicts in Multi-dimensional Data

Contents

The following Chapter introduces a taxonomy of heterogeneities in multi-dimensional Data Warehouses and Data Marts, extending the overview given in Table 2.1 (page 22). Conflicts among independent and autonomous Data Marts may occur either at the schema or at the instance level and concern both dimensions and facts. Additionally, schema–instance conflicts may occur when the universe of discourse is modelled differently, once using schema and once instance constructs. The Chapter classifies the possible conflicts along the two dimensions *schema level—instance level* and *dimension—fact*.

The following Sections detail the overview of conflicts given in Table 2.1 (page 22), introducing a taxonomy of heterogeneities in the multi-dimensional data model. We will show that conflicts among independent and autonomous Data Marts may occur either at the schema or at the instance level and concern both dimensions and facts. Additionally, *schema–instance conflicts* happen when the universe of discourse is modelled differently, once using schema and once instance constructs at different sites.

A basic classification of possible conflicts at the schema and instance level of relational data was introduced by Kim and Seo. Their work surveys *structural heterogeneity* of database models, defining several conflict classes that affect the main conceptual entity behind relational data—the *relation*. For example, "table versus table" conflicts occur when different table definitions represent the same relation(s) in different databases [Kim and Seo, 1991]. Later on, the basic classification has been extended [Lee et al., 1995], as explained in Chapter 3.

In recent years, the data integration community has been focusing on *semantic heterogeneity* of data. The semantic integration problem is far more complex than structural integration since semantic heterogeneity refers to how users interpret the meaning of given schema elements according to their understanding of the real world. In order to repair semantic heterogeneity, most approaches use ontologies [Kedad and Métais, 1999, Hakimpour and Geppert, 2001] or similar knowledge representation techniques [Singh et al., 1997].

Generally, heterogeneities among Data Marts covering the same application domain (e.g., sales figures of grocery stores) result from the use of either (1) different modelling patterns and methodologies, or (2) ambiguous domain vocabularies, or (3) a combination of both these factors. In other words, there are neither "best practices" nor universal patterns for modelling a given application domain as multi-dimensional schema; usually, every data designer conceptualizes the universe of discourse in slightly different ways. Moreover, a comprehensive and generally accepted vocabulary for the Data Warehousing domain—e.g., an ontology—has not been proposed yet.

Remarkably, even if all Data Mart designers would use such standard modelling patterns and consistent vocabularies of their application domains, the resulting models and instances of Data Marts still are not guaranteed to be free of conflicts. When representing the universe of discourse—e.g., business processes—in conceptual schemas, every organization includes some peculiarities to the model. Thus the final model depends on the organizational background, according to how its human members interpret the application domain and the organization itself. According to Navathe and Elmasri, conceptual schemas do not guarantee that consensus on the ontological definitions of the application domain exists [Navathe and Elmasri, 2004].

Multi-dimensional data is semantically richer than relational data, since it distinguishes two different conceptual entities, namely dimensions and facts. Consequently, the possible conflict classes at the schema and instance level of multi-dimensional data are not only more numerous, but may also be more complex due to dependencies between dimensions and facts. In particular, the semantics of aggregation hierarchies in dimensions gives birth to several conflicts specific to the multi-dimensional data model. Thus, data cubes have to be checked for heterogeneities separately among both the levels and non-dimensional attributes in their dimensions as well as among the "multi-dimensional space" (the set of dimension attributes) of the cube cells.

Accordingly, we will classify the possible heterogeneities in multi-dimensional information systems along two dimensions: *modelling scope* (schema level—instance level), and *model entity* (dimension—fact). Thus we obtain four basic categories of heterogeneities, plus the afore-mentioned schema–instance conflicts, as Figure 5.1 illustrates:

- Schema versus Instance .... (Section 5.1)
- Schemas of dimensions .... (Section 5.2)
- Schemas of cubes .... (Section 5.2)
- Instances of dimensions ("members") .... (Section 5.3)
- Instances of cubes ("cells") .... (Section 5.3)

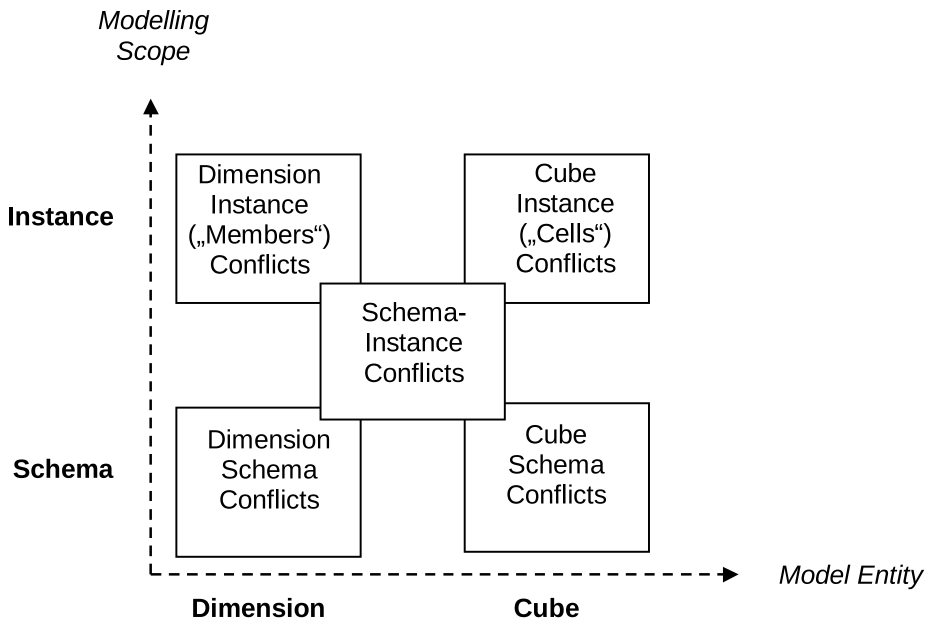

Modelling Scope
Instance
Schema
Dimension Instance („Members") Conflicts
Cube Instance („Cells") Conflicts
Schema-Instance Conflicts
Dimension Schema Conflicts
Cube Schema Conflicts
Model Entity
Dimension
Cube

Figure 5.1: Overview of Categories for Multi-dimensional Heterogeneities

Based on the multi-dimensional data model specified in Chapter 4, which formally defined schemas and instances of dimensions and cubes of Data Marts, we classify multi-dimensional heterogeneity at the following levels: (i) schema–instance conflicts (Section 5.1), (ii) schema conflicts for dimensions and cubes (Section 5.2), and (iii) instance conflicts for dimensions and cubes (Section 5.3). Detecting and eliminating all heterogeneities according to this classification is the prerequisite for obtaining *homogeneous* dimensions and cubes, as explained and formalized in the next Sections.

5.1 Schema versus Instance Conflicts

Schema–instance conflicts among Data Marts result from using different modelling elements to represent the same application domain as multi-dimensional data structures. Generally, a data cube allows the human business analyst to interpret numerical benchmarks—modelled as measure variables—in the particular *context* or business perspective given by the cube's dimensions. Thus, the combination of dimension variable values specifies the context of the facts in each cell, which is necessary for the analysts to correctly interpret the measures. Only with the help of context information the analyst is able to gain previously unknown information about the enterprise from the data cube. Measures not connected with any dimension variable are merely numbers without any semantics whatsoever.

Sometimes, part of the fact context is modelled using once elements at the schema and once elements at the instance level across distinct autonomous cubes. In particular, the multi-dimensional model provides *measure attributes* (Definition 4.9) and *dimension members* (Definition 4.3) for representing context information within the schema or instance of cubes, respectively. If dimension members are used, the context information is directly and conveniently visible from the cube—as an additional dimension variable in the cube schema. Otherwise, special names for measures in the cells encode the context information implicitly. In the latter case, the fact context is only obvious from the presence of several measure variables, the semantics of which are closely related.

These conflicts are very difficult to detect. As argued in previous work, even simpler occurrences of pure structural heterogeneity—i.e., schema–schema conflicts among models—or data heterogeneity—i.e., instance–instance conflicts among tuples—cannot be matched in fully automatic manner. The interaction of the human designer is always needed, at least to check whether automatically generated matchings are valid and plausible [Schwarz et al., 1999, Halevy et al., 2006]. Thus, the designer of the Federated DW schema has to inspect the real-world entities modelled by the autonomous Data Marts carefully in order to reveal possible schema-instance conflicts.

In most cases it is adequate to represent context of facts exclusively within the members of dimensions. The reason for this is twofold. First, the names of measure variable subsets representing some part of the fact context can be viewed as enumeration of the values of an implicit, invisible *context dimension.* Second, the dimension schema provides additional properties—e.g., hierarchies, non-dimensional attributes—that allow to model the members more precisely. Thus, using dimension members for expressing the fact context leads to more concise and more easily comprehensible cube schemas.

Example 5.1: The treatment data cubes of the case study (see Figure 2.1 on page 16 and Figure 2.4 on page 19) both represent cost figures of medical treatments, distinguishing personnel costs from material costs. In dwh1, the costs categories are contained in the schema of the treatment cube as measure variables cost_m and cost_p, whereas in dwh2 the costs categories are modelled as instances of the cost_cat dimension. Notice that the cost_cat dimension need not necessarily contain material and personnel costs, but could also provide for additional costs categories not modelled in the schema of dwh1::treatment. ◇

5.2 Schema Level Conflicts

This Section systematically categorizes and explains the conflicts that may occur among the schemas of autonomous data cubes. According to the overview depicted in Figure 5.1, the taxonomy we give further distinguishes heterogeneities among the *intensions* of dimensions and facts, respectively. Before discussing these schema conflicts in two separate Subsections, the next Subsection starts with some remarks on naming conflicts.

5.2.1 Naming Conflicts among Dimension/Cube Schemas

Inconsistent naming for attributes and other elements of autonomously and locally managed schemas is one of the basic and most obvious reasons for schema heterogeneities. Traditionally, the problem of detecting naming conflicts has been a major concern of the database and data integration research communities [Doan and Halevy, 2005]. Most of the findings achieved in these fields can be applied to Data Warehousing as well, because multi-dimensional schemas of Data Marts can be translated easily to relational schemas (see for example [Golfarelli et al., 1998]).

We denote as *naming conflicts* the occurrence of different words as labels of semantically equivalent entities in multi-dimensional Data Marts across schema boundaries. Due to the lack of standardized ontological vocabularies for the Data Warehousing domain, every organization usually maintains its own "private" vocabulary, that is well-known to the members of the organization but not disclosed to the public. Thus, it is common that heterogeneous vocabularies underlie the models of semantically "same" ontological concepts in the schema of a Data Mart. The naming conflicts across dimensions or cubes become obvious as soon as independently developed multi-dimensional schemas have to be integrated across the original organizational boundaries.

In general, naming conflicts in the context of multi-dimensional schemas are usually either homonyms or synonyms to attribute names in dimension schemas and/or cube schemas. *Homonyms* to attribute names occur if an identical attribute name maps to different ontological concepts among autonomous schemas. For example, an attribute named "good" could designate articles, as well as being a boolean or categorical attribute concerning customer feedback. In contrast, *synonyms* occur if several attribute names map to the same ontological concept. For instance, an attribute containing the price of sold products can be denoted as "price", or "unit_price", etc.

It is difficult to detect naming conflicts automatically [Embley et al., 2004, Schwarz et al., 1999]. Many approaches in the field of data integration have addressed naming conflicts—usually in combination with other, more complex problems involved [Dhamankar et al., 2004, Calvanese et al., 2001, Aslan and McLeod, 1999, Singh et al., 1997, Hammer and McLeod, 1993]. Lately, due to the popularity of information integration approaches in both industry and the research community, the focus of classical data integration has broadened to new domains, such as context oriented information retrieval, event-based or stream-based information integration systems, and so forth [Mohania and Bhide, 2008, Xu and Embley, 2006].

More recently, research in the Semantic Web community has focused on how to use ontologies for the *semantic integration* of both, schemas and information. The large body of work done in this area indicates the hardness of the problems involved. For example, the work of [Buitelaar et al., 2008] surveys recent literature and presents SOBA, an ontology-based approach for information extraction and integration. SOBA extends the earlier proposal of [Biskup and Embley, 2003] with linguistic analysis during the generation phase of an integrated ontology. In both these systems, the integrated data is materialized within views of a global database. In contrast, virtualization approaches such as InfoMosaic [Telang et al., 2008a, Telang et al., 2008b] define global schemas together with mediators (cf. Subsection 3.4.5, pp. 38).

In the multi-dimensional model, heterogeneous attribute names affect dimension schemas and cube schemas alike. Naming conflicts among autonomous dimensions may occur within: (1) the name of the dimension, (2) level names within hierarchies, (3) level attribute names in level schemas, and (4) non-dimensional attributes in level schemas (Definition 5.1). In turn, naming conflicts among autonomous cubes are possible within: (1) the name of the cube, (2) dimension attribute names, (3) measure attribute names (Definition 5.2).

Definition 5.1 (Naming conflicts among dimension schemas):
Let D and $\bar{D}$ denote two dimensions of autonomous cubes. Moreover, let $l \in L_D$, $\bar{l} \in L_{\bar{D}}$ be some level, $l.K \in S_l$, $\bar{l}.\bar{K} \in S_{\bar{l}}$ be the roll-up attributes and $l.N \in S_l$, $\bar{l}.\bar{N} \in S_{\bar{l}}$ be some non-dimensional attributes of these levels in dimension schemas S_D and $S_{\bar{D}}$, respectively. Naming conflicts among autonomous dimensions D and $\bar{D}$ affect the following elements of dimension schemas:

- Dimension: $name(D) \neq name(\bar{D})$
- Level names within hierarchies: $name(l) \neq name(\bar{l})$
- Roll-up attributes in level schemas: $name(l.K) \neq name(\bar{l}.\bar{K})$
- Non-dimensional attributes in level schemas: $name(l.N) \neq name(\bar{l}.\bar{N})$ ■

Example 5.2 (naming conflicts among dimension schemas): The dimensions of both treatment data cubes contain two naming conflicts among Data Marts dwh1 and dwh2. First, the names of the time dimensions are different: dwh1::date_time versus dwh2::date_time2. Second, the non-dimensional attributes of the method dimensions have heterogeneous names: description in dwh1::method, versus descrpt in dwh2::method. ◇

Definition 5.2 (Naming conflicts among cube schemas):
Let C and $\bar{C}$ denote two autonomous cubes. Moreover, let $A \in A_C$, $\bar{A} \in A_{\bar{C}}$ be some dimensional attribute, and $M \in M_C$, $\bar{M} \in M_{\bar{C}}$ be some measure in cube schemas S_C and $S_{\bar{C}}$, respectively. Naming conflicts among autonomous cubes C and $\bar{C}$ affect the following elements of cube schemas:

- Cube: $name(C) \neq name(\bar{C})$
- Dimensional attributes: $name(A) \neq name(\bar{A})$
- Measures: $name(M) \neq name(\bar{M})$ ■

Example 5.3 (naming conflicts among cube schemas): The treatment cubes illustrate a naming conflict among their dimensional attributes (see Figure 2.4). While dwh1::treatment contains an attribute named date, cube dwh2::treatment defines an attribute date/hr. ◇

For the discussion in the following two Subsections we assume, respectively, dimension and cube schemas in which all naming conflicts have already been repaired. Thus, our taxonomy concentrates on heterogeneities that are specific to the multi-dimensional model, irrespective of possible naming conflicts.

5.2.2 Conflicts among Dimension Schemas

The basic concept underlying the classification of heterogeneities among dimension schemas is the *equivalence* of aggregation levels across autonomous schemas. Intuitively, equivalent level intensions represent some particular real-world entity uniformly within their level schemas. Using the notion of level equivalence, we will also define various degrees of *correspondence* among sets of levels.

Our model determines equivalence among aggregation levels of autonomous dimension schemas only by the keys of level schemas, i.e. by their roll-up attributes (Definition 5.4 below). First and foremost, the hierarchy among the roll-up attributes of aggregation levels defines the semantics of dimensions—i.e., the classification of some business entity in varying detail. The non-dimensional attributes $l.N_1, ..., l.N_k$ ("N-attributes") complement the roll-up attribute $l.K$ of some level schema S_l, representing additional properties of the l-members (cf. Definition 4.2), but they have no influence on the notion of correspondence.

For the following definitions, let D and $\bar{D}$ denote two dimensions in autonomous Data Marts.

Definition 5.3 (Conceptual correspondence of levels):
Two levels $l \in S_D$ and $\bar{l} \in S_{\bar{D}}$ are called *conceptually corresponding*, denoted $l \simeq \bar{l}$, iff $l.K = \bar{l}.K$, i.e. the roll-up attributes of level schemas S_l and $S_{\bar{l}}$ model the same ontological concept. ■

Definition 5.4 (Equivalence of levels):
Two levels $l \in S_D$ and $\bar{l} \in S_{\bar{D}}$ are called equivalent, denoted $l \equiv \bar{l}$, iff $l \simeq \bar{l}$ (see above Definition 5.3), and iff $dom(l.K) = dom(\bar{l}.K)$, i.e. their associated domains are equal. If $l \equiv \bar{l}$ we also say that l *matches* $\bar{l}$ and vice versa. ■

Example 5.4: The levels [l_drug] of the two dimensions dwh1::drug and dwh2::drug are equivalent because their roll-up attribute drug corresponds conceptually between the two level schemas l_drug—i.e., both represent drugs for medications. Moreover, the domains of the drug attributes match (see Figure 2.3, page 18). Analogously, the levels [manufacturer] are also equivalent among dwh1::drug and dwh2::drug due to roll-up attribute manufacturer.

Notice how the N-attributes dwh1::drug.pkg_size and dwh2::drug.pkg_size are irrelevant for determining equivalence among level schemas dwh1::drug and dwh2::drug. Assuming heterogeneous domains for the pkg_size attributes—e.g., pieces versus grams—these domain conflicts would have to be repaired before merging the two dimensions, as explained in Chapter 7. Importantly, however, the aggregation hierarchies ($\{drug \mapsto manufacturer\}$) remain unaffected by such conflicts among N-attributes. ◇

Building upon the concept of level equivalence, heterogeneities among the schemas of autonomous dimensions may affect all elements of the dimension schema given in Definition 4.2 (page 54). Accordingly, the following subsections classify conflicts among (1) the set of levels, (2) the hierarchy among the levels, and (3) the attributes defined within level schemas.

Correspondence of Level Sets

The semantic similarity—*correspondence*—among sets of aggregation levels is based upon level equivalence (Definition 5.4). Intuitively, the more levels match across autonomous dimension schemas, the more homogeneously modelled are these schemas. Perfectly homogeneous dimension schemas—among which all levels match—are likely to represent the same real-world entities in uniform manner. Conversely, if not even a single level matches, the according dimension schemas are heterogeneous and probably represent different real-world entities.

Correspondence among the sets of aggregation levels is the formal concept for measuring the similarity of dimension schemas, as defined below. Let D and $\bar{D}$ denote two dimensions in autonomous Data Marts. Correspondence among L_D and $L_{\bar{D}}$ refers to all levels of the schemas S_D and $S_{\bar{D}}$, and whether the levels of L_D are equivalent with the levels of $L_{\bar{D}}$. Thus, correspondence among dimension schemas quantifies “set-wise equivalence” of their aggregation levels:

Definition 5.5 (Corresponding level sets):
Two level sets $L_D \in S_D$ and $L_{\bar{D}} \in S_{\bar{D}}$ of the dimension schemas S_D and $S_{\bar{D}}$ are called *corresponding*, iff every level $l \in L_D$ is equivalent with some level $\bar{l} \in L_{\bar{D}}$ and vice versa, i.e. the subsets of non-equivalent levels among L_D and $L_{\bar{D}}$ are empty: $\forall l \in L_D \exists \bar{l} \in L_{\bar{D}} : l \equiv \bar{l} \wedge \forall \bar{l} \in L_{\bar{D}} \exists l \in L_D : \bar{l} \equiv l$. We also say that L_D *matches* $L_{\bar{D}}$ and write $L_D \equiv L_{\bar{D}}$. ■

Autonomous dimension schemas S_D and $S_{\bar{D}}$ are heterogeneous if one of their levels violates the above Definition 5.5 (i.e., if their level sets are not fully corresponding). For the successful integration of dimension schemas it is necessary to exactly determine their semantic correspondence—i.e. the extent to which the level schemas match. We distinguish the following degrees of intensional heterogeneity among dimension schemas:

- Non-corresponding dimension schemas—disjoint sets of aggregation levels: the dimension schemas S_D and $S_{\bar{D}}$ are called *non-corresponding* if their level sets L_D and $L_{\bar{D}}$ do not have any equivalent level in common, i.e. $L_D \cap L_{\bar{D}} = \emptyset$.

  Example 5.5 (non-correspondence): The dimension schemas dwh1::drug and dwh2::date_time2 are non-corresponding because $L_{dwh1::drug} \cap L_{dwh2::date\_time2} = \emptyset$, i.e. there is no pair of equivalent aggregation levels among the two level sets (see Figure 2.1, page 16). ◇

- Partial-corresponding dimension schemas—overlapping sets of aggregation levels: the dimension schemas S_D and $S_{\bar{D}}$ are called *partial-corresponding* if their level sets L_D and $L_{\bar{D}}$ have at least one, but not all equivalent levels in common, i.e. $L_D \cap L_{\bar{D}} \neq \emptyset$. We further classify according to the equivalence of base levels in the hierarchies H_D and $H_{\bar{D}}$:

  a. If $l_0^D \not\equiv l_0^{\bar{D}}$, i.e. the two base levels are different, the dimension schemas S_D and $S_{\bar{D}}$ are called *inner-level corresponding.*

  b. If $l_0^D \equiv l_0^{\bar{D}} \wedge L_D \not\equiv L_{\bar{D}}$, i.e. the two base levels match, the dimension schemas S_D and $S_{\bar{D}}$ are called *base-level corresponding.*

Example 5.6 (inner-level correspondence): The dimension schemas dwh1::date_time and dwh2::date_time2 are inner-level corresponding because the levels [day] and [year] match between the two level sets, but the two base levels are different: $l_0^{dwh1::date\_time}$ = [day] versus $l_0^{dwh2::date\_time2}$ = [day/hr] (see Figure 2.4, page 19). ◇

Example 5.7 (base-level correspondence): The dimension schemas dwh1::date_time and dwh2::date_time2′ (= dwh2::date_time2 $\setminus$ [l_day/hr]; i.e., $H_{dwh2::date\_time2'}$ = { [day] $\mapsto$ [week] $\mapsto$ [year]}) are base-level corresponding. The two base levels are equivalent and level [year] also matches between the two level sets: $l_0^{dwh1::date\_time} = l_0^{dwh2::date\_time2'}$ = [day] (see Figure 2.4). However, the levels [month] of dwh1::date_time and [week] of dwh2::date_time2′ do not match. ◇

- Flat-corresponding dimension schemas—corresponding aggregation levels with different hierarchies: the dimension schemas S_D and $S_{\bar{D}}$ are called *flat-corresponding* if level set L_D matches level set $L_{\bar{D}}$, and the two base levels are the same, but the two hierarchies differ, i.e. $L_D \equiv L_{\bar{D}} \wedge l_0^D \equiv l_0^{\bar{D}} \wedge H_D \neq H_{\bar{D}}$.

  Example 5.8 (flat correspondence): Let calendar be a dimension schema with equivalent level sets to dwh1::date_time. Assuming that the calendar-hierarchy be defined as $H_{calendar} = \{day \mapsto year \mapsto month\}$, the dimension schemas dwh1::date_time and calendar are only flat-corresponding although their level sets match exactly, because $H_{calendar} \neq H_{dwh1::date\_time}$ (recall that $H_{dwh1::date\_time} = \{day \mapsto month \mapsto year\}$—see Figure 2.4, page 19). ◇

Conflicts among Attribute Domains

Domain conflicts among dimension schemas occur if two attributes in level schemas model the same concept, but assign different sets of allowed values. The FedDW conceptual model associates the domains of both the roll-up attribute $l.K$ and the non-dimensional attributes $l.N_1, ..., l.N_k$ in some level schema S_l with a data type τ (cf. Definitions 4.2 and 4.9). Thus, domain conflicts may affect roll-up attributes and N-attributes of dimensions alike.

Domain conflicts can only be discovered reliably after identifying structural (intensional) heterogeneity among dimensions, as explained above. This means, even if conceptually corresponding levels (Definition 5.3) suggest homogeneous level schemas across autonomous dimensions, the attributes of the level schemas may nevertheless be subject to domain conflicts. In such cases, the affected dimensions represent the same real-world entities, although either the precision or the measuring unit of an attribute domain differs.

According to the elements of dimension schemas we distinguish the following classes of domain conflicts. Let $l \in L_D$ and $\bar{l} \in L_{\bar{D}}$ be two levels of the dimensions D respectively $\bar{D}$ with $l \simeq \bar{l}$ and $l \in H_D$, $\bar{l} \in H_{\bar{D}}$.

- Roll-up domain conflicts (roll-up attributes in hierarchies): the domains of the two levels l and $\bar{l}$ (with $l \simeq \bar{l}$) are heterogeneous, if $dom(l.K) \neq dom(\bar{l}.K)$. We further classify inner level and base level domain conflicts, according to the position of l and $\bar{l}$ within the hierarchies H_D and $H_{\bar{D}}$:

a. *Inner level domain conflicts* occur if neither l nor $\bar{l}$ are the base levels of the hierarchies H_D and $H_{\bar{D}}$, respectively—i.e. $\exists l' \in H_D : l' \mapsto^+ l \wedge \exists \bar{l}' \in H_{\bar{D}} : \bar{l}' \mapsto^+ \bar{l}$.

b. *Base level domain conflicts* occur if either l or $\bar{l}$ is the base level of the hierarchies H_D and $H_{\bar{D}}$, respectively—i.e. $\nexists l' \in H_D : l' \mapsto^+ l \vee \nexists \bar{l}' \in H_{\bar{D}} : \bar{l}' \mapsto^+ \bar{l}$.

Notice that two conceptually corresponding levels $l \simeq \bar{l}$ become equivalent ($l \equiv \bar{l}$) by repairing the heterogeneous domains (cf. Definitions 5.3, 5.4).

Example 5.9 (inner level domain conflict): Assuming that levels [manufacturer] of the two dimensions dwh1::drug_dim and dwh2::drug_dim represent only Swiss and German manufacturers, respectively, the domains of the both inner levels [manufacturer] are different. In this case, the domain mismatch among dwh1::drug_dim and dwh2::drug_dim occurs since the schemas logically constrain that manufacturer.country be set only to the values 'CH' respectively 'DE'. ◇

Example 5.10 (base level domain conflict): The date_time dimension schemas associate the base levels with different domains in both treatment cubes (see Figure 2.4, page 19). While base level [day] in dwh1::date_time models days of years, base level [day/hr] in dwh2::date_time2 additionally contains the time of day. ◇

- Non-dimensional domain conflicts (N-attributes): the domain of two non-dimensional attributes, say $l.N$ and $\bar{l}.N$, is heterogeneous if $dom(l.N) \neq dom(\bar{l}.N)$.

  Example 5.11 (non-dimensional domain conflict): Assuming that non-dimensional attribute hourly_costs of the [method] level in the two dimensions dwh1::method and dwh2::method records costs figures once in US-$ and once in Euros, the hourly_costs domains are in conflict. ◇

It is important to note that base-level domain conflicts require special attention. Cube schemas associate measure variables with base levels in dimension schemas (see Definition 4.9, page 56). Therefore, base-level domain conflicts affect all measure attributes in cube schemas that are linked to these dimensions. In contrast, inner level domain conflicts or non-dimensional domain conflicts only affect the dimension, not the associated cubes.

5.2.3 Conflicts among Cube Schemas

The basic concept underlying our classification of heterogeneities among cube schemas is the pairwise *equivalence* of dimension attributes across autonomous schemas. Intuitively, equivalent dimensional attributes represent context of measures—i.e., some business perspective—uniformly across autonomous cube schemas, associating the cube schemas with equivalent aggregation levels across dimensions. Based on the notion of equivalent dimensional attributes, we will identify various levels of correspondence among cube schemas, and classify the possible heterogeneities.

For the following definitions, let C and $\bar{C}$ denote two cubes in autonomous Data Marts.

Definition 5.6 (Conceptual correspondence of dimension attributes): Two dimensional attributes $A[l] \in A_C$ and $\bar{A}[\bar{l}] \in A_{\bar{C}}$ of cube schemas S_C respectively $S_{\bar{C}}$ are called *conceptually corresponding*, denoted $A \simeq \bar{A}$, iff the two dimension attributes represent the same ontological concepts. ■

Definition 5.7 (Equivalence of dimension attributes): Two dimensional attributes $A[l] \in A_C$ and $\bar{A}[\bar{l}] \in A_{\bar{C}}$ of cube schemas S_C respectively $S_{\bar{C}}$ are called *equivalent*, denoted $A \equiv \bar{A}$, if their concepts match (see above Definition 5.6), and iff the levels associated with the dimensional attributes are equivalent (Definition 5.4), i.e. if $A \simeq \bar{A} \wedge l \equiv \bar{l}$. ■

Correspondence of Cube Schemas

In our model, we determine semantic similarity of cube schemas—denoted as *correspondence*—by the extent to which their equivalent dimensional attributes "overlap", based on Definition 5.7. Intuitively, corresponding cube schemas model the same context for the measure values in their cells by using the same sets of dimension attributes. Perfectly corresponding cube schemas model exactly the same multi-dimensional space, since all of their dimensional attributes are pairwise equivalent.

In order to correctly recognize the conflicts among autonomous cube schemas, it is important to assume the following two prerequisites. First, no schema–instance conflicts may exist among the cubes (cf. Section 5.1). If necessary, all schema–instance conflicts have to be repaired. Second, the dimensions associated with the cube schemas must be free of schema heterogeneities (cf. the previous Subsection 5.2.2). If necessary, the existing conflicts among all dimension schemas must be overcome.

Again, let C and $\bar{C}$ denote two cubes in autonomous Data Marts.

Definition 5.8 (Corresponding cube schemas): Two cube schemas S_C and $S_{\bar{C}}$ are called *corresponding* iff all of their dimensional attributes $A_C = \{A_1[l_1], ..., A_i[l_i], ..., A_n[l_n]\}$ and $A_{\bar{C}} = \{\bar{A}_1[\bar{l_1}], ..., \bar{A}_i[\bar{l_i}], ..., \bar{A}_n[\bar{l_n}]\}$ are pairwise equivalent, i.e.: $A_C = A_{\bar{C}} \wedge \forall A_i[l_i] \in A_C, \bar{A}_i[\bar{l_i}] \in A_{\bar{C}} : l_i \equiv \bar{l}_i$. We also say that S_C and $S_{\bar{C}}$ *match exactly* and write $S_C \equiv S_{\bar{C}}$, respectively $A_C \equiv A_{\bar{C}}$. ■

Autonomous cube schemas S_C and $S_{\bar{C}}$ are heterogeneous if one of their dimensional attributes violates the above Definition 5.8 (i.e., if their dimensional attributes are not fully corresponding). Although cube correspondence is defined upon the dimensional attributes alone, heterogeneities among autonomous cube schemas may affect both parts of the schema (cf. Definition 4.9). Accordingly, we will classify heterogeneity (1) among the sets of dimensional attributes, and also (2) among the sets of measure attributes.

Based on the notion of cube equivalence, we distinguish the following heterogeneities among the dimensional attributes of cube schemas:

- Non-corresponding cube schemas—disjoint dimensional attributes: the cube schemas S_C and $S_{\bar{C}}$ are denoted *non-corresponding* if they do not have any dimensional attribute in common, i.e. $A_C \cap A_{\bar{C}} = \emptyset$. Notice that equivalence of dimensional attributes is impossible for non-corresponding cube schemas since there is not even a single pair of same dimensional attributes.

Example 5.12 (non-corresponding cube schemas): Let dwh3::sales be a cube with dimension attributes $A_{dwh3::sales} = \{store, product\}$ and arbitrary measure attributes $M_{dwh3::sales}$. The cube schemas dwh3::sales and dwh1::treatment are non-corresponding because $A_{dwh3::sales} \cap A_{dwh1::treatment} = \emptyset$, i.e. there is no common dimension attribute among both cube schemas. ◇

- Partial-corresponding cube schemas—dimensionality conflicts: the cube schemas S_C and $S_{\bar{C}}$ are denoted *partial-corresponding* if they have at least one equivalent, but not all dimension attributes in common, i.e. $A_C \cap A_{\bar{C}} \neq \emptyset$. Since the number of dimensional attributes in case of partial correspondence differs, we also denote such heterogeneity as *dimensionality conflicts*. We further classify according to the non-equivalent subsets of dimensional attributes in S_C and $S_{\bar{C}}$:

  a. Dimension intersecting cube schemas: the cube schemas S_C and $S_{\bar{C}}$ are *dimension intersecting*, denoted $S_C \underset{\sim}{\cap} S_{\bar{C}}$, if each of the cube schemas defines at least one dimensional attribute which cannot be matched with an equivalent dimensional attribute of the other schema, i.e. $A_C \setminus A_{\bar{C}} \neq \emptyset \wedge A_{\bar{C}} \setminus A_C \neq \emptyset$.

  b. Cube schema containment: cube schema S_C is *contained in* cube schema $S_{\bar{C}}$, denoted $S_C \underset{\sim}{\subseteq} S_{\bar{C}}$, if the dimensionality of cube schema $S_{\bar{C}}$ exceeds the dimensionality of S_C, but all of the dimensional attributes in S_C match with an equivalent attribute of the other cube schema $S_{\bar{C}}$, i.e. $A_C \setminus A_{\bar{C}} = \emptyset \wedge A_{\bar{C}} \setminus A_C \neq \emptyset$.

Example 5.13 (partial-corresponding cube schemas): The cube schemas dwh1::treatment and dwh2::treatment are partial-corresponding because $A_{dwh1::treatment} \cap A_{dwh2::treatment} = \{method\}$, i.e. both cube schemas commonly define a dimension attribute method. In particular, both cube schemas are dimension-intersecting, because the dimension attributes date, date/hr, phys and cost_cat are only defined in one of these cube schemas (see Figure 2.4, page 19). ◇

Notice that the measure attributes are irrelevant for determining correspondence between autonomous cube schemas. Of course, domain conflicts that occur among measures must be repaired before merging the extensions of the autonomous cubes, as explained in Chapter 7. Importantly, however, if the dimensional attributes match exactly (Definition 5.8), an integrated cube can be computed—even if the measures only match partially (albeit this would lead to null values in the merged cube).

Conflicts among Attribute Domains

Domain conflicts among cube schemas occur if two attributes model the same concept, but assign different sets of allowed values. The FedDW conceptual model presented in Chapter 4 (pp. 51) associates the domain of each of the dimensional attributes with the domain of some aggregation level of one of the dimensions in the Data Mart. Each of the measure attributes is associated with some data type τ (cf. Definition 4.9, page 56). Therefore, domain conflicts may affect both the dimensional and the measure attributes of cubes.

Accordingly, we classify heterogeneity among the domains of attributes in cube schemas as follows. Let C and $\bar{C}$ denote two cubes in autonomous Data Marts, and $A[l] \in A_C$, $\bar{A}[\bar{l}] \in A_{\bar{C}}$ denote a pair of conceptually corresponding dimensional attributes among the two cube schemas, i.e. $A[l] \simeq \bar{A}[\bar{l}]$.

- Domain conflicts among dimensional attributes: the dimensional attributes $A[l] \in A_C$ and $\bar{A}[\bar{l}] \in A_{\bar{C}}$ cause a domain conflict if the levels associated with the attributes are not intensionally equivalent, i.e. $l \not\equiv \bar{l}$. If level l is finer than level $\bar{l}$, though (i.e. $l \mapsto^{+} \bar{l}$, see Definition 4.8), the cube schemas are called *roll-up compatible*, denoted $S_C \overset{\mapsto}{\sim} S_{\bar{C}}$.

  Example 5.14 (dimensional attributes domain conflict): Between cube schema dwh1::medication (see Figure 2.3) and another cube schema dwh3::med (patient [l_patient], drug [l_drug], date_time [month]; amount) we can observe a domain conflict: cube schema dwh3::med associates dimension attribute date_time with level [month], compared to level [day] in dwh1::medication. However, level [day] rolls-up to [month] in dwh1::medication ([day] $\mapsto$ [month], thus [day] $\mapsto^{+}$ [month]). Consequently, the cube schemas dwh1::medication and dwh3::med are roll-up compatible ($S_{dwh1::medication} \overset{\mapsto}{\sim} S_{dwh3::med}$), which means that the roll-up operation in cube dwh1::medication is sufficient to repair the conflict. $\diamond$

- Domain conflicts among measure attributes: two measure attributes, say $M \in S_C$ and $\bar{M} \in S_{\bar{C}}$ with data types τ resp. $\bar{\tau}$, cause a domain conflict if $dom(\tau) \neq dom(\bar{\tau})$.

  Example 5.15 (measure attributes domain conflict): Figure 2.4 shows that dwh2::treatment records costs figures in US-$ (attribute cost-$). Assuming that costs are given in Euros at dwh1 (attributes cost_p and cost_m), the domains of the costs attributes in both Data Marts are heterogeneous. $\diamond$

Notice that two conceptually corresponding dimensional attributes $A[l] \simeq \bar{A}[\bar{l}]$ become equivalent (i.e., $A[l] \equiv \bar{A}[\bar{l}]$) by repairing the heterogeneous domains among the associated levels l and $\bar{l}$ (cf. Definitions 5.6 and 5.7).

5.3 Instance Level Conflicts

This Section systematically categorizes and explains the conflicts that may occur among the instances—i.e., the "tuples" or "objects"—of autonomous data cubes. According to the overview depicted in Figure 5.1, the taxonomy we give further distinguishes heterogeneities among the *extensions* of dimensions and facts, i.e., among their sets of members and cells, respectively (according to the terminology discussed in Chapter 4, pp. 51).

Heterogeneous instances can only be recognized correctly between homogeneous schemas. Alternatively, if heterogeneities exist at the schema level, the intensions of the autonomous schemas must be matched first before attempting to detect heterogeneity among their extensions [Doan and Halevy, 2005]. Therefore, for our discussion about instance level conflicts we have to assume that all existing conflicts among the schemas of both dimensions and facts—see the previous Section 5.2—have been repaired.

At the instance level, heterogeneity is caused by *conflicting values* assigned to attributes when instantiating a dimension or cube schema. Analogously to naming conflicts (see Subsection 5.2.1, pp. 63), the assignment of inconsistent attribute values for "identical" real-world entities is one of the most obvious reasons for heterogeneity among autonomously managed models of the real world. Like at the schema level, value conflicts occur if heterogeneous vocabularies are used to instantiate semantically identical ontological concepts.

The detection of value conflicts among instances of homogeneous schemas—commonly denoted as *merge/purge problem* or *entity matching*—has always been a crucial challenge in data integration. If not detected properly, value conflicts cause duplicates of tuples in an integrated database. Thus, approaches for finding value conflicts among tuples of autonomous databases have received much attention in literature, especially the database, artificial intelligence (AI), knowledge discovery in databases (KDD) and WWW research communities [Doan and Halevy, 2005].

In the FedDW conceptual multi-dimensional model, value conflicts may affect either dimension members or cube cells, as analyzed and discussed in the following two Subsections.

5.3.1 Conflicts among Dimension Instances

Assuming that the schemas of dimensions are homogeneous, further heterogeneity may occur among the members (i.e., the dimension instances). Conflicts among dimension members may be caused either by single values of member attributes, or by the member extensions as a set (see Definition 4.3, page 54). Therefore, in this Subsection we further categorize (1) heterogeneities across single members in the dimension extensions, and (2) heterogeneities among the extensions of dimensions, i.e. among the member sets as a whole.

The basic concept for determining and classifying heterogeneity among dimension instances is the *equivalence* of members. For the following definition, let $l \in L_D$ and $\bar{l} \in L_{\bar{D}}$ denote two equivalent dimension levels (i.e., $l \equiv \bar{l}$, see Definition 5.4, page 65), with common parent level $\hat{l}$ (i.e., $l \mapsto \hat{l}$ and $\bar{l} \mapsto \hat{l}$) in two dimensions D and $\bar{D}$ with homogeneous dimension schemas. Moreover, let $m \in members(l)$ and $\bar{m} \in members(\bar{l})$ denote sample members of the two levels l and $\bar{l}$, respectively.

Definition 5.9 (Equivalent members):
The two members $m, \bar{m}$ (of levels l respectively $\bar{l}$, with $l \equiv \bar{l}$) are called *equivalent*, denoted $m \equiv \bar{m}$, if their keys are the same, i.e. $m(l.K) = \bar{m}(\bar{l}.K)$. ■

The notion of equivalence among members allows to decide whether two members of autonomous dimensions refer to the same real-world entity or not. In case of equivalence, two members independently represent the same real-world entity across autonomous dimensions, with potentially conflicting descriptions or properties (i.e., values of roll-up or non-dimensional attributes). Equivalent members must be merged to a single member in an integrated dimension to avoid duplication. Conversely, two non-equivalent members represent different entities. In the latter case, value conflicts are not possible because both members have to be kept separately in an integrated dimension.

Conflicts among Single Members

From the viewpoint of single instances, pairs of dimension members cause heterogeneity if either their roll-up functions differ or if values of non-dimensional attributes are ambiguous, as defined below. On the one hand, conflicting roll-up functions would corrupt the value hierarchies in dimensions, so that aggregations along the hierarchy performed by an analyst's OLAP tool would deliver false results. On the other hand, value conflicts are a problem because the heterogeneous values give ambiguous descriptions of dimension members.

According to the elements of level schemas (cf. Definition 4.2, page 54), we distinguish between heterogeneity among the roll-up attributes respectively the non-dimensional attributes as follows:

- Heterogeneous roll-up functions: the equivalent members $m \equiv \bar{m}$ are *roll-up heterogeneous* if the roll-up functions $\rho_D^{l \mapsto \hat{l}}$ and $\rho_{\bar{D}}^{\bar{l} \mapsto \hat{l}}$ deliver ambiguous parent members for the keys $m(l.K)$ resp. $\bar{m}(\bar{l}.K)$, i.e. $\rho_D^{l \mapsto \hat{l}}(m) \neq \rho_{\bar{D}}^{\bar{l} \mapsto \hat{l}}(\bar{m})$.

  Example 5.16 (heterogeneous roll-up functions): The drug_dim dimensions in Figure 2.3 illustrate heterogeneous roll-up functions among their members. In dwh1::drug_dim, the member with drug-value = 'B' rolls-up to manufacturer 'Novartis' ($\rho_{dwh1::drug\_dim}^{drug \mapsto manufacturer}(B) = Novartis$), whereas member 'B' in dwh2::drug_dim rolls-up to manufacturer 'Bayer' ($\rho_{dwh2::drug\_dim}^{drug \mapsto manufacturer}(B) = Bayer$). $\diamond$

- Non-dimensional value conflicts: the equivalent members $m \equiv \bar{m}$ cause a non-dimensional value conflict, if the N-attributes of S_l and $S_{\bar{l}}$ overlap (i.e. $N_{l \cap \bar{l}} = \{l.N_1, ..., l.N_k\} \cap \{\bar{l}.\bar{N}_1, ..., \bar{l}.\bar{N}_k\} \neq \emptyset$) and the values of some common N-attribute(s) N' differ among the two members, i.e. $\exists N' \in N_{l \cap \bar{l}} : m(N') \neq \bar{m}(N')$.

  Example 5.17 (Non-dimensional value conflict): Among the two drug_dim-dimensions, the values of N-attribute pkg_size are inconsistent for the members with drug-value 'A' (see Figure 2.3): '25 pieces' in dwh1::drug_dim versus '30 pieces' in dwh2::drug_dim. $\diamond$

Heterogeneous Member Extensions

From the set-oriented point of view, pairs of dimension extensions cause problems if subsets of their members are equivalent, and, additionally, non-dimensional attributes of overlapping equivalent members contain conflicting values. In contrast, overlapping sets of members from autonomous dimension can be merged easily if (1) the schemas of the autonomous dimension are homogeneous, and (2) the member extensions are free of heterogeneous roll-up functions and non-dimensional value conflicts, as defined above. If both these prerequisites hold, duplicate members are straightforward to detect and repair. Moreover, disjoint member extensions can simply be unified.

- Overlapping member extensions: given a pair of levels $l \equiv \bar{l}$, the member sets $members(l)$ and $members(\bar{l})$ (see Definition 4.4) *overlap* if the extensions of key attributes $l.K$ and $\bar{l}.K$ are non-disjoint, i.e. $\exists m \in members(l), \bar{m} \in members(\bar{l}) : m \equiv \bar{m} \wedge \exists m' \in members(l), \bar{m}' \in members(\bar{l}) : m' \not\equiv \bar{m}'$.

Example 5.18 (overlapping member extensions): The extensions of the [l_manufacturer] levels among dimension instances dwh1::drug and dwh2::drug overlap: as stated in Example 5.4 (page 65), the [l_manufacturer] schemas of dwh1 and dwh2 are equivalent, and the non-dimensional attributes of the two level schemas are identical (see Figure 2.3, page 18). The l_manufacturer-members ('Roche', 'CH') exist in both extensions. ◇

Notice that the extensions of autonomous dimensions have to be unified in order to compute an integrated extension for the global dimension. While the set union of disjoint member extensions is easy to determine, overlapping subsets of member extensions have to be checked carefully for value conflicts among the roll-up attributes (see heterogeneous roll-up functions) and the non-dimensional attributes (see non-dimensional value conflicts) in order to avoid duplicate members. In literature, this problem is often referred to as the "merge/purge problem" [Doan and Halevy, 2005].

5.3.2 Conflicts among Cube Instances

Assuming that the schemas of autonomous cubes are homogeneous, further heterogeneities may occur among the cells (i.e., the cube instances). Importantly, the assumption of homogeneous cube schemas guarantees a degree of *integrity* among the cube cells. In particular, the dimensional attributes of fact schemas (i.e., the "coordinates" of the cells, cf. Definition 4.9 on page 56) must be equivalent, and the domains of the dimensional attributes match (see Subsection 5.2.3), given that the cube schemas are homogeneous.

Therefore, the coordinates of cells merely determine whether two cells of autonomous cubes *"collide"* (i.e., they refer to the same point in the multi-dimensional space of an integrated cube, and thus have to be merged), or *"co-exist"* (i.e., the two cells represent different facts, and thus, both of them must be kept). We denote this concept as equivalence of cube cells, as defined below.

For the following definition, let $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ denote two cube instances (i.e., sets of cells) in autonomous Data Marts, while $f \in c$ and $\bar{f} \in \bar{c}$ denote two single cells of the cube instances $c(S_C)$ and $\bar{c}(S_{\bar{C}})$, respectively.

Definition 5.10 (Equivalent cells):
The two cells f and $\bar{f}$ (of $c(S_C)$ respectively $\bar{c}(S_{\bar{C}})$, with $S_C \equiv S_{\bar{C}}$, $A_C = A_{\bar{C}}$) are called *equivalent*, denoted $f \equiv \bar{f}$, if the coordinates of both cells match exactly, i.e. $\forall A_i \in A_C, \bar{A}_i \in A_{\bar{C}} : f(A_i) = \bar{f}(\bar{A}_i)$. ■

Based upon the notion of *equivalent cells*, we define precisely the semantic extensional correspondence between cell sets across autonomous cubes. Two autonomous cube extensions *overlap* iff they contain at least one pair of equivalent cells. Conversely, disjoint cube extensions contain only non-equivalent cells. Accordingly, we distinguish overlapping and disjoint cube extensions as follows:

- Overlapping cube instances: the cell extensions $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ *"potentially overlap"* if the cube schemas S_C and $S_{\bar{C}}$ define overlapping or identical measure attributes, i.e. $M_C \cap M_{\bar{C}} \neq \emptyset$ or $M_C = M_{\bar{C}}$. If additionally at least one pair of cells is equivalent—i.e., collides—among the two cell sets (i.e., $\exists f \in c, \bar{f} \in \bar{c} : f \equiv \bar{f}$), $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ *"overlap"*.

Example 5.19 (overlapping cube extensions): The cell sets of cubes dwh1::medication and dwh2::medication potentially overlap because the two cube schemas are homogeneous, and define identical measure attributes. Therefore, it may happen that the coordinates of cells in the both cube instances are the same, as applies to our case study (see the cells with coordinates (‘p2’, ‘A’, 03-02-06) in Figure 2.3, page 18). ◇

- Disjoint cube instances: the cell sets $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ are *disjoint* if the cube schemas S_C and $S_{\bar{C}}$ define disjoint measure attributes, i.e. $M_C \cap M_{\bar{C}} = \emptyset$. In this case, the cubes $C, \bar{C}$ are called *merge-compatible*.

  Example 5.20 (disjoint cube instances): Assumed that the measure attributes of the medication cubes were named “qty1” and “cost1” in dwh1 instead of qty and cost, the cube instances dwh1::medication and dwh2::medication would be disjoint (see Figure 2.3, page 18). Although there are two cells with coordinates (‘p2’, ‘A’, 03-02-06), in this case their measure attribute values were distinct. The two cube instances could be merged into a cube with measure attributes qty, cost, qty1 and cost1. ◇

Both overlapping cube instances and disjoint cube instances are defined on homogeneous cube schemas, i.e. their dimension attributes are exactly the same (cf. Subsection 5.2.3). Consequently, subsets of cells with identical coordinates may occur in overlapping cube instances and disjoint cube instances alike. The difference is, however, that conflicting *values* of measure attributes among these cells can only occur if the measure attributes of the cube schemas overlap. By definition, the measure attributes of disjoint cube instances do not overlap. Therefore, conflicting measure values can never occur among the cells of disjoint cube instances, so that these cells can easily be merged into a global cube with measure attributes $M_C \cup M_{\bar{C}}$.

5.4 Summary

In the previous Sections we classified heterogeneity among multi-dimensional models at the schema-instance level (5.1), the schema level (5.2) and the instance level (5.3). The following tables summarize the conflict categories introduced and refer to the definitions and examples discussed in this Chapter.

- Table 5.1 outlines schema versus instance conflicts studied in Section 5.1.

Table 5.1: Schema versus instance heterogeneity in multi-dimensional models

Conflict	Description	Example
Fact context vs. members	Part of the fact context is modelled differently; in one Data Mart as measure variables (cube schema), whilst in another DM as dimension members (cube instance).	treatment cubes (Figure 2.4)—see Example 5.1.

- Table 5.2 outlines conflicts among the schemas of dimensions, as studied in the first Subsection of 5.2.

Table 5.2: Classes of heterogeneity among autonomous dimension schemas

Conflict	**Description**	**Example**
Naming conflicts	Different name labels for ontologically same schema elements: – $name(D) \neq name(\bar{D})$, or – $name(l) \neq name(\bar{l})$, or – $name(l.K) \neq name(\bar{l}.\bar{K})$, or – $name(l.N) \neq name(\bar{l}.\bar{N})$	Dimensions of the treatment cubes (Figure 2.4)—see Example 5.2.
Non-corresponding dimension schemas	Two dimension schemas S_D and $S_{\bar{D}}$ do not have any common equivalent aggregation level: $L_D \cap L_{\bar{D}} = \emptyset$.	Dimension schemas dwh1::drug and dwh2::date_time2 (Figure 2.1)—see Example 5.5.
Partially corresponding dimension schemas (overlapping aggregation levels)	The level sets of two dimension schemas S_D and $S_{\bar{D}}$ have at least one, but not all equivalent level(s) in common: $L_D \cap L_{\bar{D}} \neq \emptyset$. a. Inner-level corresponding: $l_0^D \not\equiv l_0^{\bar{D}}$. b. Base-level corresponding: $l_0^D \equiv l_0^{\bar{D}}$.	Time dimensions of the medication and treatment cubes (Figure 2.1): a. Inner-level corresp.: dimension schemas dwh1::date_time and dwh2::date_time2 (see Example 5.6); b. Base-level corresp.: dimension schemas dwh1::date_time and dwh2::date_time2′—new base level [day] instead of [l_day/hr] (see Example 5.7).
Flat corresponding dimension schemas (same levels, different hierarchies)	Two dimension schemas S_D and $S_{\bar{D}}$ have equivalent sets of aggregation levels with identical base levels, but the hierarchies do not match: $L_D \equiv L_{\bar{D}} \wedge l_0^D \equiv l_0^{\bar{D}} \wedge H_D \neq H_{\bar{D}}$.	Dimension schema dwh1::date_time (Figure 2.4) and dimension schema calendar with $H_{calendar} = \{day \mapsto year \mapsto month\}$—see Example 5.8.

Table 5.2: Classes of heterogeneity among autonomous dimension schemas

Conflict	Description	Example
Domain conflicts	Two attributes in level schemas of autonomous dimensions assign different sets of allowed values (i.e., different domains). These domain conflicts affect either roll-up attributes of conceptually corresponding levels (Definition 5.3), or non-dimensional attributes of conceptually corresponding or equivalent levels (Definitions 5.3, 5.4) in dimension schemas.	Roll-up attributes: a. Inner level domains: [manufacturer] levels of dwh1::drug_dim and dwh2::drug_dim in the medication cubes (Figure 2.3), assumed restriction to Swiss resp. German manufacturers only—see Example 5.9. b. Base level domains: base levels of dwh1::date_time and dwh2::date_time2 in the treatment cubes (Figure 2.4)—see Example 5.10.
		Non-dimensional attributes: [method] levels of dimension schemas dwh1::method and dwh2::method in the treatment cubes (Figure 2.4)—see Example 5.17.

• Table 5.3 outlines conflicts among the schemas of cubes, as studied in the second Subsection of 5.2.

Table 5.3: Classes of heterogeneity among autonomous fact schemas

Conflict	Description	Example
Naming conflicts	Different name labels for ontologically same attributes: – $name(C) \neq name(\bar{C})$, or – $name(A) \neq name(\bar{A})$, or – $name(M) \neq name(\bar{M})$	Measures of the treatment cubes (Figure 2.4)—see Example 5.3.
Non-corresponding cube schemas (disjoint dimensional attributes)	Two cube schemas S_C and $S_{\bar{C}}$ do not have any equivalent dimensional attribute in common: $A_C \cap A_{\bar{C}} = \emptyset$.	Cube schema dwh1::treatment (Fig. 2.4) and fictitious cube schema dwh3::sales with $A_{dwh3::sales}$ =\{*store*, *product*\}—see Example 5.12.

Table 5.3: Classes of heterogeneity among autonomous fact schemas

Conflict	**Description**	**Example**
Partially corresponding cube schemas (dimensionality conflicts)	Two cube schemas S_C and $S_{\bar{C}}$ have at least one, but not all equivalent dimensional attribute(s) in common: $A_C \cap A_{\bar{C}} \neq \emptyset$. a. Dimension intersecting ($S_C \overset{\cap}{\sim} S_{\bar{C}}$): subsets of non-equivalent dimensional attributes in both S_C and $S_{\bar{C}}$. b. Cube schema containment ($S_C \overset{\subseteq}{\sim} S_{\bar{C}}$): dimensionality of $S_{\bar{C}}$ is higher than of S_C.	The cube schemas dwh1::treatment and dwh2::treatment (Figure 2.4) are dimension intersecting—see Example 5.13.
Domain conflicts	Two attributes in autonomous cube schemas assign different sets of allowed values (i.e., different domains). These domain conflicts affect either conceptually corresponding dimensional attributes (Definition 5.6), or measure attributes of cube schemas.	Conceptually corresponding dimensional attributes: cube schema dwh1::medication (Fig. 2.3) and fictitious cube schema dwh3::med (patient [l_patient], drug [l_drug], date_time [month]; amount); the domains of dimensional attributes date_time are [day] (dwh1::medication) versus [month] (dwh3::med)—see Example 5.14.
		Measure attributes: costs attributes of cube schemas dwh1::treatment and dwh2::treatment (Fig. 2.4)—see Example 5.15.

- Table 5.4 outlines conflicts among the instances of dimensions (members), as studied in the first Subsection of 5.3.
- Finally, Table 5.5 outlines conflicts among the instances of cubes (cells), as studied in the second Subsection of 5.3.

Table 5.4: Classes of heterogeneity among members of autonomous dimensions

Conflict	**Description**	**Example**
Heterogeneous roll-up functions	The roll-up functions of two equivalent members $m \equiv \bar{m}$ define ambiguous parent members: $\rho_D^{l \mapsto \hat{l}}(m) \neq \rho_{\bar{D}}^{\bar{l} \mapsto \hat{l}}(\bar{m})$.	Members with drug-value = 'B' of the drug_dim dimensions in the medication cubes (Figure 2.3)—see Example 5.16.
Non-dimensional value conflict	Some N-attribute contains ambiguous values among two equivalent members $m \equiv \bar{m}$ that belong to levels with equivalent schemas: $\exists N' \in \{l.N_1, ..., l.N_k\} \cap \{\bar{l}.\bar{N}_1, ..., \bar{l}.\bar{N}_k\} : m(N') \neq \bar{m}(N')$.	N-attribute pkg_size among members with drug-value = 'A' of the drug_dim dimensions in the medication cubes (Figure 2.3)—see Example 5.17.
Overlapping member extensions	The sets of members among two equivalent levels $l \equiv \bar{l}$ contain at least one pair of equivalent members: define ambiguous parent members: $\exists m \in members(l), \bar{m} \in members(\bar{l}) : m \equiv \bar{m} \wedge \exists m' \in members(l), \bar{m}' \in members(\bar{l}) : m' \not\equiv \bar{m}'$.	Member extensions of the l_drug-levels among dimension instances dwh1::drug and dwh2::drug in the medication cubes (Figure 2.3)—see Example 5.18.

Table 5.5: Classes of heterogeneity among cells of autonomous cubes

Conflict	**Description**	**Example**
Overlapping cube extensions (cells)	Two cubes (i.e., sets of cells) $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ define identical dimensional attributes and overlapping measure attributes: $(M_C \cap M_{\bar{C}} \neq \emptyset \vee M_C = M_{\bar{C}}) \wedge \exists f \in c, \bar{f} \in \bar{c} : f \equiv \bar{f}$. Therefore, all cells with identical coordinates among c and $\bar{c}$ "collide".	Cell sets of the medication cubes (Figure 2.3)—see Example 5.19.
Disjoint cube extensions (cells)	Two cubes (i.e., sets of cells) $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ define identical dimensional attributes and disjoint measure attributes: $M_C \cap M_{\bar{C}} = \emptyset$. Therefore, all cells with identical coordinates among c and $\bar{c}$ "co-exist"; $c(S_C)$ and $\bar{c}(S_{\bar{C}})$ are merge-compatible.	Cell sets of cubes dwh2::medication (see Figure 2.3) and dwh1::medication′ with renamed measure attributes ($M_{dwh1::medication'} = \{qty1, cost1\}$)—see Example 5.20.

Chapter 6

Architecture of Federated Data Warehouses

This chapter presents the architectural foundation of the FedDW approach to the integration of autonomous multi-dimensional Data Marts. The proposed reference architecture describes the schema layers and components that are needed to realize the basic paradigm of *source-to-target mappings* from the autonomous Data Marts to the stable, global schema. Using the schema layers and the auxiliary components *Dimension Repository* and *Data Dictionary* of the reference architecture, the DM integration process consists of the following phases: (1) global schema modelling, (2) dimension integration, and (3) fact integration.

The reference architecture described in this Chapter—depicted in Figure 6.1—gives an outline of the concepts and components needed for Federated DW systems. The architecture is conceptual, i.e. it does not restrict the technology for implementing the Federated DW system. Part IV of this thesis will discuss prototypes of both, a visual schema integration tool and a querying tool, that conform to the introduced reference architecture and implement its concepts (see pp. 143). Along with the reference architecture we propose a comprehensive methodological framework for dimension and fact integration, as explained below.

The global, mediated multi-dimensional schema of the Federated DW reference architecture hides the heterogeneities among autonomous Data Marts from users and the application layer. *Semantic mappings* specify the necessary transformations from each of the autonomous DMs to the global schema. This approach allows the users to query the global schema, while the *mediator* of the federation layer reformulates the user query to an equivalent set of queries against the autonomous Data Marts.

Thus, the reference architecture describes a Federated DW system as *tightly coupled* with the autonomous and possibly distributed component DMs. It is important to point out that "tightly coupled"—in the sense used in this Chapter—is the commonly accepted term for data integration systems that provide a stable, global schema (e.g., see [Sheth and Larson, 1990]). In contrast, the "tightly coupled" approach to Data Mart integration of DaWaII [Cabibbo and Torlone, 2005] refers to a fundamentally different concept—i.e., the complete materialization of autonomous cubes, according to the consolidated multi-dimensional schema (cf. Section 3.5 in Chapter 3, pp. 39).

Based on the general five-level architecture of federated databases [Sheth and Larson, 1990], the Federated DW reference architecture defines *source-to-target mappings* for the integration of autonomous DMs against the stable, global multi-dimensional schema. This mapping paradigm follows the idea proposed in similar form by the so-called BGLaV ("both global and local as view") approach [Xu and Embley, 2004]. It combines the advantages of global-as-view (GAV) and local-as-view (LAV) data integration (as discussed in Subsection 3.4.4 of Chapter 3, pp. 37). On the one hand, the direction of mappings from sources to the global schema (as in the GAV strategy) facilitates query processing since the query plan can be computed from the mapping in straightforward manner. On the other hand, the stable global schema (as in the LAV strategy) secures the federation layer against local schema changes [Lenzerini, 2002, Halevy, 2001].

The Federated DW reference architecture defines an additional component, the so-called *Dimension Repository*, extending "classical" and well-established Federated Database systems. The dimension repository stores the consolidated dimension schemas and mirrors their data, such that copies of all dimension members are present at the federated layer (see Figure 6.1). When answering queries, the dimensional data and meta-data (members, hierarchies, etc.) are available directly from the dimension repository.

This approach reduces the complexity of distributed query processing in the Federated DW system, and improves overall response time. By reading dimensions from the repository, the mediator component can eliminate

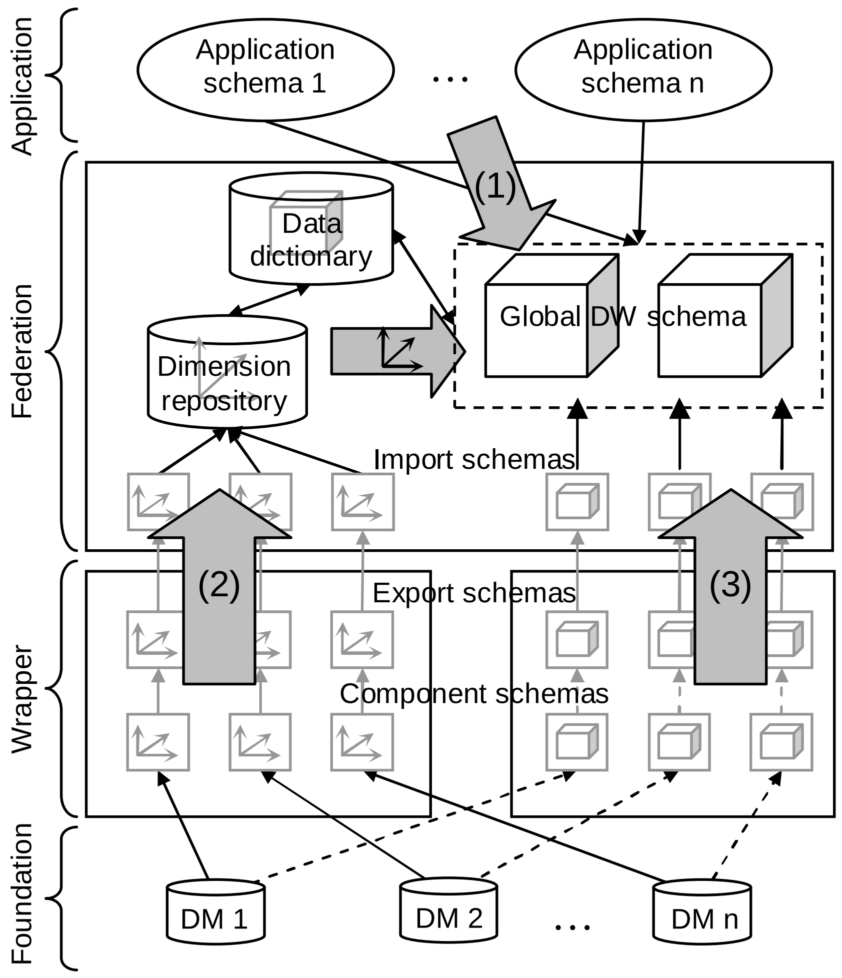


Figure 6.1: Federated Data Warehouse conceptual architecture.

the sub-queries necessary to retrieve the dimensions from local Data Marts. The dimension repository is motivated by the basic idea behind the "Skalla" [Akinde et al., 2003] and "DWS-AQA" approaches [Bernardino et al., 2002]. Experiments conducted in these two approaches indicate that local dimension replicates improve query performance. Moreover, dimensions—similar to domain ontologies—typically evolve slowly, and are relatively small in size [Kimball, 2002, Akinde et al., 2003].

Based on the concepts (1) mediated, multi-dimensional global schema, (2) both-as-view data integration, and (3) replication of dimension members and meta-data into the dimension repository, the Federated DW reference architecture allows for systematic integration of Data Marts. We propose the following methodological framework for Data Mart integration, addressing the interrelated sub-problems dimension integration and fact integration among the logical schemas of autonomous Data Marts in a systematic way [Berger and Schrefl, 2008]:

- *Global schema modelling* (label (1) in Figure 6.1) and *user requirements engineering*: in the first place, the Federated DW administrator designs the cubes and dimensions of the global multi-dimensional schema. The cube definitions of the global schema should correspond to the information needs of the applications and users that work on top of the federated layer. Object oriented modelling techniques, e.g. the Unified Modelling Language UML [(OMG), 2009], are commonly used by software engineers to model the user requirements [Luján-Mora et al., 2006]. In our architecture, for example, UML use case diagrams can be used to express the "desired" cubes of the global schema. In general, requirements engineering is well researched, but it lies out of the scope of this thesis. The interested reader can refer, for example, to the survey [Nuseibeh and Easterbrook, 2000] for an overview of the requirements engineering field.
- *Dimension integration* (label (2) in Figure 6.1): the federation layer of the Federated DW system needs to correctly interpret the common dimensional context of the fact data in the component Data Marts. Therefore, it is necessary to represent the export schemas of local, heterogeneous dimensions in the canonical model—i.e., to define the import schemas. Moreover, the Federated DW administrator must design *semantic mappings* from the local to the global dimensions.

  Each mapping associates a series of transformation operators with the import schema of one local dimension, specifying how to repair the heterogeneities at the schema and at the instance level in order to represent the members of the local dimension in the global multi-dimensional schema. The result of the dimension integration process is a consolidated, global dimension schema, which constitutes the "common multi-dimensional space" of the federated system. The *dimension repository* is the "cache" of consolidated dimension schemas, hosting the integrated dimension schemas, the mappings and copies of the dimension members.
- *Fact integration* (label (3) in Figure 6.1): in order to unify the fact data of autonomous Data Marts, the multi-dimensional entities modelled in the facts of component Data Marts must be mapped to the global schema. For that purpose, the fact integration process aims at representing the export schemas of autonomous facts within the canonical model—i.e., creating the import schemas. Moreover, the Federated DW administrator must design *semantic mappings* from the local DMs to the global schema.

  Each mapping associates a sequence of transformation operators with the import schema of one local fact, specifying how to overcome the heterogeneities at the schema and at the instance level in order to represent the instances of the local fact in the global multi-dimensional schema. As the result of the fact integration process, the fact data of local cubes gets connected to the global multi-dimensional space, that is defined by the consolidated dimensions. Both the meta-data of the global schema and the mappings are stored in the *data dictionary.* When processing a user query over the global schema, the federated system uses the mappings both to generate a query plan—decomposing the query to a set of queries to the local systems—and to transform all result data to the global schema.

The main benefits of the Federated DW approach presented herein are the following: (1) it allows the integration of local, autonomous Data Marts to a global schema, whilst the DMs retain schema and data autonomy; (2) at the same time, the global schema is stable, so that the federated layer is secure against schema changes in the local DMs; (3) the tightly coupled federation of DMs allows the users and applications to operate on the global schema, providing a level of abstraction from the heterogeneities among the DMs; and (4) the proposed architecture supports various physical models (e.g., relational or multi-dimensional implementation platforms).

Part III

Enabling the Federated Data Warehouse—the FedDW Approach

Chapter 7

Integration Methodology for Autonomous Data Marts

Contents

This chapter discusses a general methodology for the integration of autonomous Data Marts. It introduces the notion of *homogeneity* as the desirable property of autonomous facts and dimensions based on the conceptual multi-dimensional data model discussed in Chapter 4. First, Section 7.1 introduces the Dimension Algebra and Fact Algebra. The algebras provide a rich set of operators that formalize the transformation of multi-dimensional models. Sequences of the Dimension/Fact Algebra operators express the semantic mappings from autonomous Data Marts to the global multi-dimensional schema. Thus, the conversion operators are used as the "building blocks" of the FedDW integration methodology. Second, Section 7.2 details the general methodology to follow for Data Mart integration, when using the Dimension/Fact Algebra to formulate the semantic mappings. The methodology precisely describes the steps necessary to obtain homogeneous facts and dimension among autonomous Data Marts. Starting with the design of a global multi-dimensional schema and the import schemas of the autonomous Data Marts, the outcome of the integration methodology is the semantic mappings—i.e, for each Data Mart the sequence of conversions at the conceptual schema level.

7.1 Import Schema Transformation with Dimension/Fact Algebra Expressions

Federated DW systems conforming to the reference architecture proposed in the previous Chapter 6 provide an integrated view over heterogeneous Data Marts without physically building a global Data Mart. Instead, semantic mappings specify the exact sequence of operations that are necessary to convert the heterogeneous data into the global multi-dimensional schema. The reference architecture of Federated DW systems requires a mechanism to define source-to-target mappings in order to integrate autonomous Data Marts with a global multi-dimensional schema.

This Section addresses the need for a source-to-target mapping definition formalism by introducing the *Dimension Algebra (DA)* and *Fact Algebra (FA)*. These algebras formally define a rich set of conversion operators for dimensions respectively facts defined in the FedDW conceptual multi-dimensional model (see Chapter 4). The conversion operators address all classes of heterogeneity in the multi-dimensional model that have been analyzed in Chapter 5. In turn, using the DA and FA operators as "building blocks", the next Section 7.2 will demonstrate how to employ such conversion operators in a systematic methodology that addresses the conjoint integration of dimensions and facts of autonomous Data Marts.

In order to illustrate the operators of both, the Dimension Algebra and Fact Algebra, we slightly simplify the treatment cube and date_time dimension of Data Mart dwh1 in the case study (see Chapter 2, Figure 2.4 on page 19). Figure 7.1 shows the two fact tables treatmt and treats, as well as dimension table date of Data Mart "foo". Compared to dimension dwh1::date_time, dimension foo::date introduces the additional non-dimensional attribute name to the [month] level. Moreover, assume another fact table foo2::treatmt, which we will use to demonstrate the effect of the n-ary operator μ (merge dimensions).

foo::treatmt (date [day]; cost_p, cost_m)

date	cost_p	cost_m
23-02-06	356.0	425.0
24-02-06	125.2	1742.0
25-02-06	473.0	903.8

foo::treats (date[day], ccat[ccat]; costs)

date	*ccat*	costs
23-02-06	cost_p	480.0
24-02-06	cost_m	613.0
25-02-06	cost_m	624.5

foo::date (day $\mapsto$ month $\mapsto$ year)

[day]	[month]	name	[year]
23-02-06	02-06	'Feb. 06'	2006
24-02-06	02-06	'Feb. 06'	2006
25-02-06	02-06	'Feb. 06'	2006

foo2::treatmt

date	cost_p	cost_m
23-02-06	298.0	607.0
24-02-06	1872.4	941.0

Figure 7.1: Data Mart "foo" with facts treatmt, treats, and dimension date. The [all]-level of dimension foo::date has been omitted. Moreover, fact treatmt of Data Mart "foo2" is specified.

7.1.1 Dimension Algebra

To support schema transformation with a platform independent language of conversion operators in the Federated DW reference architecture (Chapter 6, see pp. 81), our approach employs the so-called *Dimension Algebra* (DA), introduced in [Berger and Schrefl, 2008]. The DA allows to transform the dimension import schemas of autonomous Data Marts during the dimension integration process (cf. 7.2.2 and 7.2.3). Its operators manipulate dimension schema elements or instances, thus expressing the semantic mappings from autonomous dimensions to the global dimension (see Definition 7.1).

The Dimension Algebra comprises the three operators σ, π and ψ (select, project, aggregate; see Examples 7.1, 7.2, 7.3, respectively) of an earlier proposal [Cabibbo and Torlone, 2005], plus the five additional operators defined in [Berger and Schrefl, 2008]. The DA operators are applied to an input dimension D, defined in the conceptual data model introduced in Chapter 4 (see pp. 51). Each DA operator results in an output dimension D', of which either the original schema S_D or its instance $d(S_D)$ is modified compared to D. This closed semantics—inspired by the approach followed in the relational algebra [Codd, 1970]—allows to use the output of DA operators as input for another DA operator. Thus, several operators can be combined easily to form more complex expressions:

Definition 7.1 (DA expression):
A Dimension Algebra *expression* is a sequence of the DA operators σ (select), π (project), ψ (aggregate)—cf. Examples 7.1, 7.2 and 7.3—as well as ζ (rename), δ (change), γ (convert), Ω (override roll-up)—see Definitions 7.2 – 7.5—that is applied to some input dimension D, deriving the output dimension D'. Alternatively, the n-ary operator μ (merge dimensions—see Definition 7.6) is applicable on a set $\mathbb{D} = \{D_1, D_2, ..., D_n\}$ of input dimensions with identical names, producing an output dimension D^G. ■

The following Examples 7.1, 7.2 and 7.3 illustrate the basic DA operators select, project and aggregate, respectively, as defined in [Cabibbo and Torlone, 2005]. For the following examples, let *date* denote the dimension 'foo::date' with instance $date(S_{foo.date})$, as shown in Figure 7.1.

Example 7.1 (σ – select): Applying $\sigma_{\{day \geq 24-02-06\}}(date)$, we obtain the following dimension $date'$:

foo::date'

[day] $\mapsto$	[month] name $\mapsto$	[year] $\mapsto$	[all]
24-02-06	02-06 'Feb. 06'	2006	all
25-02-06	02-06 'Feb. 06'	2006	all

◇

Example 7.2 (π – project): Operator $\pi_{\{day\}}(date)$ obtains dimension $date'$ with $L_{date'} = \{day\}$, $S(L_{date'}) = \{l\_date(day)\}$ and $H_{date'} = \{day \mapsto all\}$:

foo::date'

[day] $\mapsto$	[all]
23-02-06	all
24-02-06	all
25-02-06	all

◇

Notice from the above Example 7.2 that the [all]-level can never be eliminated with the π-operator.

Example 7.3 (ψ – aggregate): Operator $\psi_{month}(date)$ obtains dimension $date'$ with $L_{date'} = \{month, year, all\}$, $S(L_{date'}) = \{$*l_month (month, name), l_year (year)*$\}$, and $H_{date'} = \{month \mapsto year \mapsto all\}$:

foo::date'		
[month] name $\mapsto$	[year] $\mapsto$	[all]
02-06 'Feb. 06'	2006	all

◇

In what follows, we define the DA operators *rename* (ζ), *change* (δ), *convert* (γ) and *override rollup* (Ω) formally. For Definitions 7.2–7.5, let D be a dimension with schema $S_D = \{L_D, S(L_D), H_D\}$ (Definition 4.2) and instance $d(S_D) = \{V_d, \rho_d\}$ (Definition 4.3). With D' we denote the output dimension with schema S'_D and instance $d'(S'_D)$ obtained by applying a DA expression on D.

Definition 7.2 (ζ – rename):
Operator ζ changes the name of some attribute in S_D. In particular, ζ can be applied to the following objects:

- Rename the dimension to 'newD': $\zeta_{newD \leftarrow D}(D)$.
- Rename the dimension instance to 'new-d': $\zeta_{new-d \leftarrow d}(d(S_D))$.
- Rename a dimension level $l \in L_D$ to 'l′': $\zeta_{l' \leftarrow l}(D)$.
- Rename the level attribute $l.K$ of level schema $S_l \in S(L_D)$ to '$l.K'$': $\zeta_{K' \leftarrow K}(S_l)$.
- Rename some non-dimensional attribute $l.N$ of level schema $S_l \in S(L_D)$ to '$l.N'$': $\zeta_{N' \leftarrow N}(S_l)$. ■

Example 7.4: Applying $\zeta_{[time] \leftarrow [day]}(date)$, we obtain the following dimension $date'$:

foo::date'			
[time] $\mapsto$	[month] name $\mapsto$	[year] $\mapsto$	[all]
23-02-06	02-06 'Feb. 06'	2006	all
24-02-06	02-06 'Feb. 06'	2006	all
25-02-06	02-06 'Feb. 06'	2006	all

◇

Definition 7.3 (δ – change):
Let $T_l = members(l)$ be a subset of V_d with $l \in L_D$. Further, let $\bar{N} \in S_l = \{l.K, l.N_1, ..., l.N_k\}$ be some non-dimensional attribute of level l, and $v, w \in dom(l.\bar{N})$ be values for N-attribute $l.\bar{N}$, with $v \neq w$. Operator $\delta_{w \leftarrow v}(d.l.\bar{N})$ computes a new member-subset T'_l in d' such that $T'_l = \{t'(S_l) \mid \exists t \in T_l : t'(S_l \setminus \bar{N}) = t(S_l \setminus \bar{N}),\ t(\bar{N}) = v,\ t'(\bar{N}) = w\}$. ■

Example 7.5: Operator $\delta_{\text{'}Febr.\%\text{'} \leftarrow \text{'}Feb.\%\text{'}}(date.month.name)$ obtains dimension $date'$ with the following instance $d'(S_{date'})$:

foo::date'			
[day] $\mapsto$	[month] name $\mapsto$	[year] $\mapsto$	[all]
23-02-06	02-06 'Febr. 06'	2006	all
24-02-06	02-06 'Febr. 06'	2006	all
25-02-06	02-06 'Febr. 06'	2006	all

◇

Definition 7.4 (γ – convert):
Let $l.\bar{N} \in S_l$ be some non-dimensional attribute, $T_l = members(l)$ and θ be an operator over $dom(\bar{N})$. The result of $\gamma_{\theta\bar{N}}(d)$ is d' with member set $T'_l = \{t(S_l) \mid \exists t' \in T_l : t(S_l \setminus \bar{N}) = t'(S_l \setminus \bar{N}),\, t(\bar{N}) = \theta t'(\bar{N})\}$. ■

Example 7.6: Assume that *trunc(n)(str)* is an operator over $dom(month.name)$, truncating a character string *str* after the first n characters. Operator $\gamma_{trunc(2)(month.name)}(date)$ results in the dimension $date'$ with the following instance $d'(S_{date'})$:

foo::date'

[day] ↦	[month] name ↦	[year] ↦	[all]
23-02-06	02-06 ‘Fe’	2006	all
24-02-06	02-06 ‘Fe’	2006	all
25-02-06	02-06 ‘Fe’	2006	all

◇

Definition 7.5 (Ω – override roll-up):
Let $m, v \in V_d$ be members of some dimension with $l = level(m)$, $k = level(v)$, such that $l \mapsto k \in H_D$ and $v \neq \rho^{l \mapsto k}(m)$. The result of $\Omega_{m \mapsto v}(d.l)$ is d' in which the result of $\rho^{l \mapsto k}(m)$ is changed to v. ■

Example 7.7: Assume that V_{date} in Figure 7.1 contains an additional *year*-value of *‘06*. Then, operator $\Omega_{02-06 \to ‘06}(date.month)$ obtains dimension $date'$ with the following instance $d'(S_{date'})$:

foo::date'

[day] ↦	[month] name ↦	[year] ↦	[all]
23-02-06	02-06 ‘Feb. 06’	‘06	all
24-02-06	02-06 ‘Feb. 06’	‘06	all
25-02-06	02-06 ‘Feb. 06’	‘06	all

◇

In order to merge several import dimension schemas, defined by DA expressions, the operator μ (merge) is applied. From an input set of n dimensions $D_1, D_2, ..., D_n$ having identical names, with each D_i stored in a different Data Mart, the μ operator returns a merged output dimension D^G, determining its set of levels $D^G.L_D$, its hierarchy $D^G.H_D$, and member set $D^G.V_D$ from the given dimensions. Finally, the family of roll-up functions $D^G.\rho_d$ is computed. The result of the algorithm—i.e., the output dimension D^G—is *consistent* if Definition 7.7 holds.

Definition 7.6 (μ – merge dimensions):
Let $\mathbb{D} = \{D_1, D_2, ..., D_n\}$ be a set of dimensions with identical names. Operator $\mu(\mathbb{D})$ merges the dimensions $D_1, D_2, ..., D_n$ according to Algorithm 7.1, computing an output dimension D^G, the “global” dimension.

To obtain a meaningful result, all heterogeneities among the dimensions in $\mathbb{D}$ should be repaired with the appropriate other DA operators, as explained in the previous Section 7.2. ■

Definition 7.7 (Merge consistency):
The dimension D^G computed by Algorithm 7.1 is *consistent*, iff (1) $D^G.H_D$ forms a lattice, and (2) $\rho^{l \mapsto k}$ is consistent in the sense of Definition 4.5 for all pairs $l, k \in D^G.H_D$. ■

Algorithm 7.1 mergeDim

Input: Dimensions $\{D_1, ..., D_n\}$
Output: Merged dimension D^G

```
Require: $D_1 = D_2 = ... = D_n$                              ▷ Name equality
1: $D^G.L_D = \bigcup_{i=1...n}\{D_i.L_D\}$;
2: $D^G.H_D = \bigcup_{i=1...n}\{D_i.H_D\}$;
3: $D^G.V_d = \bigcup_{i=1...n}\{\forall l \in D^G.L_D \mid members(D_i.l)\}$;
4: $D^G.\rho_d = \bigcup_{l,k} \rho^{l \mapsto k}\colon\ l, k \in D_G.H_D$;
```

7.1.2 Fact Algebra

Fact integration in relation with dimension integration has not been addressed as a problem in the literature. It can be argued that well-known, relational data integration techniques [Zhao and Ram, 2007, Lenzerini, 2002] suffice—but only for very simple fact sets (e.g., with degenerate dimensions). These techniques, however, cannot handle adequately the semantics of dimensions and hierarchies specific to the multi-dimensional data model. Instead, several heterogeneities among autonomous Data Marts must be resolved in an interrelated way, as discussed in Chapter 5, pp. 59 [Berger and Schrefl, 2006].

To tackle the fact integration problem in the Federated DW reference architecture (Chapter 6, see pp. 81), our approach employs a platform independent fact transformation language called *Fact Algebra* (FA). The FA allows to transform the cube import schemas of autonomous Data Marts during the fact integration process (cf. 7.2.1, 7.2.2, and 7.2.3). Its operators manipulate fact schema elements or instances, thus expressing the semantic mappings from the facts of autonomous cubes to the global cube (see Definition 7.8).

The FA defines an overall set of ten operators that address the heterogeneities among facts analyzed in Chapter 5. Applying the FA operators to some input cube C—defined in the conceptual data model introduced in Chapter 4—results in an output cube C', of which either the schema S'_C or its instance $c'(S'_C)$ is modified compared to C. Analogously to the DA, this closed semantics—inspired by the approach followed in the relational algebra [Codd, 1970]—allows to "chain" several FA operators, using the output of FA operators as the input for another FA operator. Thus, several operators can be combined easily to form more complex FA expressions:

Definition 7.8 (FA expression):
A Fact Algebra expression is a sequence of the unary FA operators σ (select), π (project), λ (delete measure), ζ (rename), γ (convert), ε (enrich dimensions), ϱ (roll-up), χ (merge measures) and ξ (split measure)—see Definitions 7.9 – 7.14—that is applied to some input cube C, deriving an output cube C'. Alternatively, n-ary operator μ (merge facts—Definition 7.15) is applicable on two overlapping input cubes C_1 and C_2, producing an output cube C^G. ■

Now we will formally define the FA operators *select* (σ), *project* (π), *delete measure* (λ), *rename* (ζ), *convert* (γ), *enrich dimensions* (ε), *roll-up* (ϱ), *merge measures* (χ) and *split measure* (ξ) that are applied in the local context of fact integration. Let C be a cube with schema $S_C = \{A_C, M_C\}$, instance $c(S_C)$, and C' be the result of the FA expression with schema S'_C and instance $c'(S'_C)$.

Definition 7.9 (σ–select, π–project, λ–delete measure, ζ–rename): The four basic operators σ, π, λ and ζ are defined like in the relational algebra [Codd, 1970] and are to be used as follows:

- Select: $\sigma_{[P]}(c)$ computes the cube instance $c'(S_C)$ containing all tuples $t \in c(S_C)$ satisfying the predicate(s) P.
- Project: $\pi_{(L \subset A_C)}(C)$ reduces the dimensionality of C' by projecting on one or more dimensional attributes L.
- Delete measure(s): $\lambda_{(N \subset M_C)}(C)$ computes a fact set C' reducing the measure attributes to the projection N.
- Rename: $\zeta_{A' \leftarrow A}(C)$ and $\zeta_{M' \leftarrow M}(C)$ rename dimensional attribute $A \in A_C$ to A' and measure attribute $M \in M_C$ to M' in C', respectively. ■

Definition 7.10 (γ – convert): Let $\bar{M} \in M_C$ be a measure attribute of C, and θ be an operator defined over $dom(\bar{M})$, and $v \in dom(\bar{M})$ be some legal $\bar{M}$-value. The result of operator $\gamma_{\bar{M}\theta v}(c)$ is C' with schema $S'_C = S_C$ and $c'(S'_C) = \{t' \mid \exists t \in c : t(S_C \setminus \bar{M}) = t'(S_C \setminus \bar{M}),\, t'(\bar{M}) = t(\bar{M})\theta v\}$. ■

For the examples in the remainder of this subsection, let *treatmt* denote the fact table foo::treatmt depicted in Figure 7.1 with schema $S_{treatmt}$ and instance $treatmt(S_{treatmt})$.

Example 7.8: Applying $\gamma_{cost\_m+25}(treatmt)$, we obtain cube $treatmt'$ with the following instance (or cells):

foo::treatmt'

date	cost_p	cost_m
23-02-06	356.0	450.0
24-02-06	125.2	1767.0
25-02-06	473.0	928.8

◇

Definition 7.11 (ε – enrich dimensions): Let $\hat{A}[l] \notin A_C$ be the name of some dimension attribute, and C be some cube of Data Mart DM. Moreover, $l \in L_D$ denotes an aggregation level of some dimension $D \in DM$. Operator $\varepsilon_{\hat{A}[l]=v}(c)$ results in cube C' with schema $S'_C = \{[A_C \cup \hat{A}], M_C\}$ and instance $c'(S'_C) = \{t' \mid \exists t \in c : t'(S'_C) = t(S_C),\, t'(\hat{A}) = v\}$. Thus, $\varepsilon_{\hat{A}[l]=v}(c)$ allows to increase the dimensionality of C'.

Alternatively, if level $\hat{l}$ is not available in the dimensions $\{D_1, ..., D_m\} \in DM$, operator $\varepsilon_{\hat{A}[\hat{l}]=v}(c)$ creates the new, degenerate dimension $\hat{D}$ in Data Mart DM, whereby $L_{\hat{D}} = \{\hat{l}\}$, $S_{\hat{l}} = (\hat{l})$, and $H_{\hat{D}} = \{\hat{l} \mapsto l_{all}\}$. ■

Example 7.9: Operator $\varepsilon_{source[src]=\text{‘}dm1\text{’}}(treatmt)$ obtains cube $treatmt'$ with the following cells:

foo::treatmt'

date	*source*	cost_p	cost_m
23-02-06	‘dm1’	356.0	450.0
24-02-06	‘dm1’	125.2	1767.0
25-02-06	‘dm1’	473.0	928.8

Notice that $\varepsilon_{source[src]=\text{‘}dm1\text{'}}(treatmt)$ creates the degenerate context dimension source with $L_{source} = \{src\}$, $S_{src} = (src)$, and $H_{source} = \{src \mapsto all\}$. $\diamond$

Definition 7.12 (ϱ – roll-up):
Let $A[l] \in A_C$ denote some dimensional attribute of S_C, and $\hat{l}$ be an aggregation level coarser than l in the hierarchy of the same dimension D, i.e. $l, \hat{l} \in S_D \wedge l \mapsto^+ \hat{l} \in H_D$. Operator $\varrho_{A[l \Rightarrow \hat{l}]}(c)$ is the convenience notation for the following sequence of other FA operators:

1. $c_1 = \zeta_{A' \leftarrow A}(c)$ – rename A to temporary name A'.
2. $c_2 = \varepsilon_{A[\hat{l}]=\rho_D^{l \mapsto^+ \hat{l}}(A')}(c_1)$ – enrich the dimensional attributes of S_C with attribute A. The new A-value $f(A)$ for every fact $f \in c_2(S'_C)$ is the (possibly transitive) parent of the l-member $f'(A')$ in $\hat{l}$, with $f' \in c_1(S'_C)$. This means, $\forall f \in c_2(S'_C), f' \in c_1(S'_C) : f(A) = \rho_D^{l \mapsto^+ \hat{l}}(f'(A'))$.
3. $c' = \pi_{A_C \setminus A'}(c_2)$ – finally, reduce the dimensionality of $c(S_C)$ to the original number of dimensional attributes by projection. $\blacksquare$

Thus, the roll-up operator $\varrho_{A[l \Rightarrow \hat{l}]}(c)$ decreases the detail level of cube instance $c(S_C)$ by rolling-up the dimensional attribute $A[l]$ to coarser aggregation level $\hat{l}$. The operator results in a *bag of facts* for every member $m \in members(\hat{l})$. An aggregation function such as sum would again group the bag of facts within a cell into a single fact per cell. Thus, intuitively, the roll-up operator ϱ merges several original cube cells into a single cell.

Example 7.10: $\varrho_{date[day \Rightarrow month]}(treatmt)$ obtains the following cube $treatmt'$:

foo::treatmt'

date	cost_p	cost_m
02-06	356.0	450.0
02-06	125.2	1767.0
02-06	473.0	928.8

Notice how the date column contains a bag of [month]-values ‘02-06’. The original treatmt-cube identified each fact uniquely per day, which is no longer possible in the coarser treatmt' cube (except with an aggregation function). $\diamond$

Finally, the *pivot* operators χ (merge measures) and ξ (split measure) can be used to repair schema–instance heterogeneity among cubes (cf. Section 5.1 of the conflict taxonomy, pp. 62, and 7.2.1). Two different pivot-options are available: *merge measure attributes* (χ) or *split measure attribute* (ξ)—see the following Definitions 7.13 and 7.14. Intuitively, χ merges several measures into a single one, preserving their original context by extracting an additional dimension attribute. In contrast, ξ transforms part of the fact context (i.e., one of the dimension attributes) into one or several new measure attributes.

Definition 7.13 (χ – merge measures):
Let $L = \{M_1, ..., M_k\}$ (i.e, $L \subseteq M_C = \{M_1, ..., M_m\}$, $k \leq m$) denote the set of measure attributes to be merged, $\bar{M} \notin M_C$ and $\bar{A} \notin A_C$ be new attribute names. Operator $\chi_{L \Rightarrow \bar{M}, \bar{A}}(c)$ creates cube C' with schema $S'_C = \{[A_C \cup \bar{A}], [(M_C \setminus L) \cup \bar{M}]\}$ and instance $c'(S'_C) = \bigcup_{i=1...k}\{t \mid \exists r \in c : t(\bar{A}) = M_i,\ t(\bar{M}) = r(M_i),\ t(A_C) = r(A_C),\ t(M_C \setminus L) = r(M_C \setminus L)\}$. $\blacksquare$

Example 7.11: Operator $\chi_{\{cost\_p,cost\_m\}\Rightarrow costs,ccat}(treatmt)$ results in cube $treatmt'$ with the following cells:

foo::treatmt'

date	*ccat*	costs
23-02-06	cost_p	356.0
23-02-06	cost_m	425.0
24-02-06	cost_p	125.2
24-02-06	cost_m	1742.0
25-02-06	cost_p	473.0
25-02-06	cost_m	903.8

◇

For the following Definition 7.14, let $\bar{A} \in A_C$ be some dimension attribute and $\bar{M} \in M_C$ be some measure of cube C.

Definition 7.14 ($\boldsymbol{\xi}$ – split measure):
Let $V = \{t(A') \mid t \in c(S_C)\}$ denote the set of all A'-values in c such that $V = \{V_1, ..., V_m\}$, and $B = A_C \setminus A'$ be the "remaining" dimension attributes. Operator $\xi_{\bar{M}\Rightarrow A'}(c)$ creates cube C' with schema $S'_C = \{[A_C \setminus A'], [(M_C \setminus \bar{M}) \cup V]\}$ and instance $c'(S'_C) = \{t(S'_C) \mid \exists t_1, ..., t_m \in c : \forall i, j = 1...m : t_i(B) = t_j(B) \wedge t(B) = t_1(B) \wedge t(V_1) = t_1(A') \wedge t(V_2) = t_2(A') \wedge ... \wedge t(V_m) = t_m(A')\}$.■

Example 7.12: Applying $\xi_{costs\Rightarrow ccat}(treatmt')$ to the $treatmt'$ cube of previous Example 7.11 results in cube $treatmt''$ with the following cells. Notice that $treatmt''$ is equivalent to the original cube treatmt given in Figure 7.1:

foo::treatmt"

date	cost_p	cost_m
23-02-06	356.0	425.0
24-02-06	125.2	1742.0
25-02-06	473.0	903.8

◇

As explained in 7.2.3, a higher number of measures results in higher expressivity of the fact schema, thus fewer cells, and vice versa. Therefore, *split measure* reduces the number of cells, whereas *merge measures* leads to an increased number of cells.

During the last step of the fact integration process the import fact schemas, defined by FA expressions, are mapped to the global DW. If one or more subsets of the facts have overlapping coordinates, the semantic relationship between the fact extensions has to be specified by the merge facts (μ) operator (see below).

Definition 7.15 ($\boldsymbol{\mu}$ – merge facts):
Let $\mathbb{C} = \{C_1, ..., C_n\}$ be a set of cubes, whereby $G = \bigcap_{i=1...n}\{M_{C_i}\}$, and *Op* = { *prefer, sum, avg, min, max* } be a predefined set of operators. Furthermore, let $N = \{\hat{M}_1, ..., \hat{M}_m\} \subseteq G$ be the subset of measures in all cubes for which an operator $\theta \in Op$ should be applied. Operator $\mu(\mathbb{C}[N, \theta])$ computes the result cube C^G with schema $S_{\hat{C}} = \bigcap_{i=1...n}\{A_{C_i}\}, \bigcup_{i=1...n}\{M_{C_i}\}$ and instance $\hat{c}(S_{\hat{C}}) = \{t(S_{\hat{C}}) \mid \exists t_1 \in c_1, ..., t_i \in c_i, ..., t_n \in c_n : t(S_{C_1} \setminus N) = t_1(S_{C_1} \setminus N), ..., t(S_{C_i} \setminus N) = t_i(S_{C_i} \setminus N), ..., t(S_{C_n} \setminus N) = t_n(S_{C_n} \setminus N), t(N) = \theta(t_1(N), ..., t_i(N), ..., t_n(N))\}$.

Example 7.13: Applying operator μ to merge the foo::treatmt and foo2::treatmt into a global cube named F'—i.e., $F' = \mu(\{foo::treatmt, foo2::treatmt\}\ [\{cost\_p, cost\_m\}, avg])$—we obtain as result:

F' (μ(*foo::treatmt* [{*cost_p*, *cost_m*}, *avg*] *foo2::treatmt*))

date	cost_p	cost_m
23-02-06	418.0	212.5
24-02-06	62.6	1177.5
25-02-06	236.5	764.15

◇

The FA operator *merge facts* (μ) computes semantically meaningful measures for fact subsets with overlapping coordinates (cf. 7.2.1). The *prefer*-operator is appropriate for facts on identical identities (with "stock" semantics [Lenz and Shoshani, 1997]), whereas the other set operators can be used for "context-related" entities [Berger and Schrefl, 2006] (with "flow" semantics [Lenz and Shoshani, 1997]). In most cases the sum-operator is the best choice. Only a human expert is able to choose an appropriate set operator θ according to the semantic relationship between the fact sets.

7.2 Defining Semantic Mappings

The integration of autonomous Data Marts pursues the definition of adequate semantic mappings from the import schemas of dimensions and facts to the global multi-dimensional schema. This Section defines the systematic process that is necessary for addressing the heterogeneities among multi-dimensional schemas and instances, classified in Chapter 5, in conjoint manner. During the integration process, the Federated DW administrator must (1) recognize existing conflicts and define the adequate conversions to obtain *homogeneous import schemas* of both dimensions and facts, and (2) specify how to handle *overlapping extensions* (sets of instances) among both dimensions and facts.

In order to recognize and resolve the heterogeneities among autonomous Data Marts correctly, it is important to identify the *real-world entities* in the universe of discourse that the dimensions and facts represent. As mentioned in Chapter 5 (see pp. 59), heterogeneities among Data Marts that cover the same application domain result from either different modelling approaches or ambiguous domain vocabularies, or even a combination of both these factors. Thus, for successful integration of autonomous Data Marts, the administrators of Federated DW systems must be aware of possibly ambiguous entities and vocabularies underneath the models of autonomous Data Marts.

Usually, ambiguity among Data Mart schemas results in naming conflicts (i.e., homonyms and synonyms among attribute names), but more subtle consequences of ambiguity are possible. In particular, homonymous dimension names can mask other conflicts at the schema level, e.g. a dimensionality conflict. For example, an attribute named "good" can either refer to article names, or mean a boolean attribute measuring whether customer were satisfied with the quality of services or not. In similar fashion, synonymous attribute names in either dimensions or facts often hide other classes of heterogeneity—e.g., domain conflicts—that are only revealed after renaming the synonyms.

In this Section we propose a systematic integration methodology for heterogeneous and autonomous Data Marts, based on the taxonomy of conflicts introduced in Chapter 5 (see pp. 59). The general paradigm behind this methodology views the federated layer as the *"privileged" hub* on top of all local Data

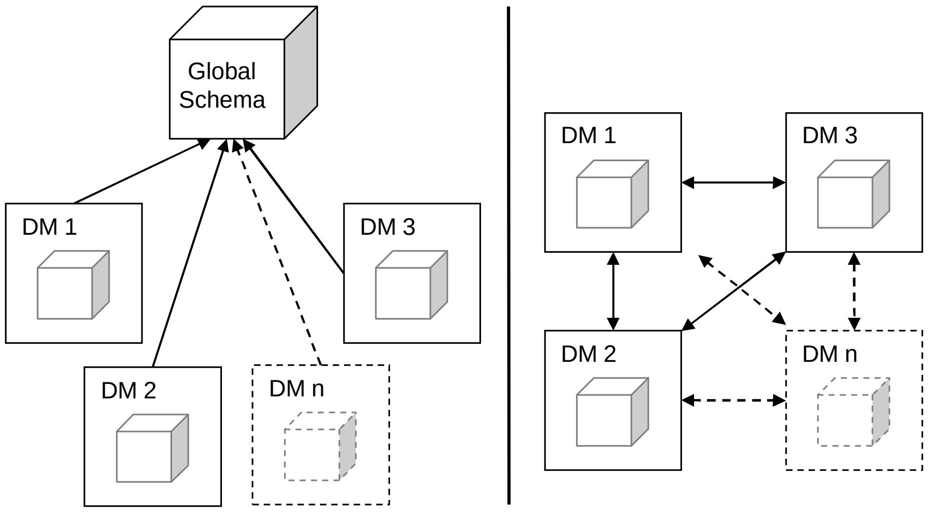


Figure 7.2: Paradigms for semantic mappings definition between Data Marts—“hub” (left) vs. “point-to-point” (right).

Marts. In this hub architecture, the Federated DW system administrates the global schema and collects mappings for each local and autonomous DM that participates in the federation. These mappings define how to convert local dimension and fact data to the global schema. The global “hub” is the only system component knowing the import schemas of all local DMs.

The hub integration paradigm allows for easier maintenance of the federated system than point-to-point Data Mart integration. As illustrated in Figure 7.2, the hub paradigm requires semantic mappings for each local Data Mart to the global schema, resulting in n necessary mappings for n Data Marts. In contrast, the point-to-point paradigm demands pairwise mappings among all Data Marts that participate in the federation. Thus, in an architecture with n Data Marts, point-to-point integration needs $\frac{n(n-1)}{2}$ mappings.

Moreover, hub federations of Data Marts are better extensible than point-to-point federations. If an additional Data Mart enters the federation, the necessary new mapping is easy to integrate with the existing hub of mappings. However, in point-to-point federations the additional mapping would at least require that the other mappings be checked for side-effects. In the worst case, all point-to-point mappings would have to be redesigned [Lenzerini, 2002].

In the FedDW approach, the design of autonomous Data Mart federations follows the Federated DW reference architecture (see the previous Chapter 6, pp. 81) and the hub mapping paradigm. In order to design the Federated DW system, it is necessary to define the global schema first. As mentioned in the previous Chapter, the business analysts’ information requirements determine the global schema design. Subsequently, the Federated DW administrator applies the proposed integration methodology individually on each of the local DMs, resulting in semantic mappings from the DMs to the global schema. This source-to-target mapping approach both ensures that the local Data Marts retain their schema and data management autonomy, and secures the federated system against local schema changes. If the component schema of a Data Mart changes, only the mapping must be updated accordingly while the global schema remains valid. Moreover, each existing mapping is isolated from changes happening in the other mappings, assuring the extensibility of the federation.

Designing the global multi-dimensional schema is an *iterative process*, driven by both, the information requirements of business analysts and the existing multi-dimensional Data Mart schemas. Of course, establishing a Federated DW system is only useful if the given, autonomous Data Marts sufficiently overlap (i.e. with respect to their universe of discourse). In practice, the most sensible option is to evolve the global schema from one of the preexisting schemas. For example, the global schema of the health insurance organization in the case study (see Figure 2.2 on page 17) has been designed from the preexisting schema of Data Mart dwh1 (Figure 2.1), but with the date_time dimension slightly modified (level [quarter] in g::date_time has replaced level [month] in dwh1::date_time).

The FedDW integration methodology for autonomous Data Marts consists of three consecutive phases. These phases systematically follow the taxonomy of heterogeneities among both dimensions and facts presented in Chapter 5, and repair all conflicts. Starting with the removal of schema-instance conflicts in the first phase (Subsection 7.2.1), the DM import schemas are harmonized step by step. During the second phase, the analyst overcomes heterogeneity at the schema level in order to achieve homogeneous dimension and fact schemas (Subsection 7.2.2). The third and final phase considers all remaining conflicts at the instance level (Subsection 7.2.3). In general, heterogeneities among dimensions are repaired before conflicts among facts, because cube schemas logically depend on the dimension schemas in the multi-dimensional data model (see Chapter 4, pp. 51).

For the following subsections, let $DM = \{C_1, ..., C_n; D_1, ..., D_m\}$ denote some local, autonomous Data Mart that is to be mapped to global Data Mart $\hat{DM} = \{\hat{C}_1, ..., \hat{C}_{\hat{n}}; \hat{D}_1, ..., \hat{D}_{\hat{m}}\}$. Notice that both, the numbers $n, \hat{n}$ of cubes and the numbers $m, \hat{m}$ of dimensions are not necessarily the same across DM and $\hat{DM}$. In practice, at least the number of dimensions probably mismatches among Data Marts, as illustrated in the case study (see Chapter 2).

As formalized in Definitions 7.16 and 7.17, the integration methodology defines mappings $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ for all dimensions $\{D \in \{D_1, ..., D_m\} \mid \exists \hat{D} \in \{\hat{D}_1, ..., \hat{D}_{\hat{m}}\} : S'_D \equiv S_{\hat{D}} \wedge d'(S'_D) \equiv d(S_{\hat{D}})\}$—i.e., all dimensions D in the local Data Mart of which the transformed dimension schema S'_D and instance $d'(S'_D)$ are equivalent to one of the dimensions given in the global schema. Moreover, the integration methodology defines mappings $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ for all cubes $\{C \in \{C_1, ..., C_n\} \mid \exists \hat{C} \in \{\hat{C}_1, ..., \hat{C}_{\hat{n}}\} : S'_C \equiv S_{\hat{C}}\}$, whereby $S'_C = \mathcal{M}_{C \rightsquigarrow \hat{C}}(S_C)$—i.e., all cubes C in the local Data Mart of which the *transformed cube schema* S'_C is equivalent to one of the cubes in the global schema. Finally, the methodology merges the extensions of each global dimension $\hat{D}$ and each global cube $\hat{C}$ from the instances of transformed, local dimensions and cubes with set-wise mappings $\mathcal{I}^{\hat{D}}$ respectively $\mathcal{I}^{\hat{C}}$. The set-wise instance mappings consider every dimension and cube of the local Data Marts that map to the respective global dimension $\hat{D}$ and cube $\hat{C}$. We denote with D^{DM} some dimension D of DM that is mapped to the global schema this way. Analogously, C^{DM} denotes one of the cubes C of DM mapped to the global schema.

The following Definition 7.16 formalizes the semantic mappings $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ and $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ for dimension and cube schemas, respectively:

Definition 7.16 (Semantic mappings – dimension and cube schemas): Given dimension D and cube C of some autonomous Data Mart DM, the FedDW integration methodology creates two schema mappings: $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ for dimension schema S_D, respectively $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ for cube schema S_C. Both these mappings are defined from DM to the global multi-dimensional schema $\hat{DM}$. The semantics of these schema mappings is given by the following functions:

- $\mathcal{M}_{D \rightsquigarrow \hat{D}} : S_D \rightarrow S'_D$ computes the *dimension import schema* S'_D from the given dimension export schema S_D. Each mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ comprises a Dimension Algebra expression (cf. Definition 7.1) that is restricted to operators ζ (rename), ψ (aggregate), π (project), and γ (convert).
- $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ computes the *cube import schema* S'_C from the given cube export schema S_C. Each mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}} : S_C \rightarrow S'_C$ comprises a Fact Algebra expression (cf. Definition 7.8) that is restricted to operators ζ (rename), ϱ (roll-up), π (project), and γ (convert). ■

The following Definition 7.17 formalizes the semantic mappings $\mathcal{I}^{\hat{D}}$ and $\mathcal{I}^{\hat{C}}$ for dimension and cube extensions, respectively:

Definition 7.17 (Semantic mappings – dimension and cube extensions): Given dimension $\hat{D}$ and cube $\hat{C}$ of the global Data Mart $\hat{DM}$, the FedDW integration methodology creates one instance mapping $\mathcal{I}^{\hat{D}}$ for all dimensions of local Data Marts $DM_1, ..., DM_k$ that map to dimension $\hat{D}$, respectively $\mathcal{I}^{\hat{C}}$ for all cubes of local Data Marts $DM_1, ..., DM_k$ that map to cube $\hat{C}$. The semantics of these instance mappings is given by the following functions:

- $\mathcal{I}^{\hat{D}} : \{DM_1.d(S'_D) \times ... \times DM_k.d(S'_D)\} \rightarrow \hat{d}(S_{\hat{D}})$ merges dimension instances $DM_1.d, ..., DM_k.d$ to compute the member extension $\hat{d}(S_{\hat{D}})$ of global dimension $\hat{D}$. Each mapping $\mathcal{I}^{\hat{D}}$ comprises a Dimension Algebra expression (cf. Definition 7.1) that is restricted to operators Ω (override roll-up), δ (change), and μ (merge dimensions).
- $\mathcal{I}^{\hat{C}} : \{DM_1.c(S'_C) \times ... \times DM_k.c(S'_C)\} \rightarrow \hat{c}(S_{\hat{C}})$ merges cube instances $DM_1.c, ..., DM_k.c$ to compute the cells extension $\hat{c}(S_{\hat{C}})$ of global cube $\hat{C}$. Each mapping $\mathcal{I}^{\hat{C}}$ comprises a Fact Algebra expression (cf. Definition 7.8) that is restricted to operators χ (pivot merge measures), ξ (pivot split measure), and μ (merge facts). ■

The following Algorithms 7.2 and 7.3 detail the steps that are necessary to create the semantic mappings for the schemas respectively instances of dimensions and cubes according to the FedDW integration methodology. In Algorithm 7.2, let the $\equiv$-operator denote the equivalence of ontological concepts representing dimensional context once in dimension members, and once in measure attributes of cube schemas. Thus, the $\equiv$-operator is used to indicate schema–instance conflicts (cf. Section 5.1, pp. 62).

Algorithm 7.2 Overview of the Data Mart schema integration process

Input: Local Data Mart $DM = \{C_1^{DM}, ..., C_n^{DM}; D_1^{DM}, ..., D_m^{DM}\}$; global Data Mart $\hat{DM} = \{\hat{C}_1, ..., \hat{C}_{\hat{n}}; \hat{D}_1, ..., \hat{D}_{\hat{m}}\}$

Output: Dimension mappings $\bigcup_{j=1...m}\{\mathcal{M}_{D_j \rightsquigarrow \hat{D}}\}$, and cube mappings $\bigcup_{i=1...n}\{\mathcal{M}_{C_i \rightsquigarrow \hat{C}}\}$.

```
1: for all $C \in \{C_1^{DM}, ..., C_n^{DM}\}$ do
   ▷ schema–instance conflicts: merge measures to context dimension
2:     if $\exists L \subseteq M_C, \hat{A}[\hat{l}] : L \equiv members(\hat{l})$ then
3:         Add $\chi_{L \Rightarrow \bar{M}, \bar{A}}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$
   ▷ schema–instance conflicts: split measure to contextualized facts
4:     else if $\exists A'[l'] \in A_C, \bar{B} \subseteq A_{\hat{C}} : members(l') \equiv \bar{B}$ then
5:         Add $\xi_{\bar{M} \Rightarrow A'}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$
6:     end if
7: end for                                    ▷ Schema–instance conflicts repaired
8: for all $D \in \{D_1^{DM}, ..., D_m^{DM}\}$ do
9:     Compute $\mathcal{M}_{D \rightsquigarrow \hat{D}}$                          ▷ See Algorithm 7.4
10: end for                          ▷ Dimension schema heterogeneities repaired
11: for all $C \in \{C_1^{DM}, ..., C_n^{DM}\}$ do
12:     Compute $\mathcal{M}_{C \rightsquigarrow \hat{C}}$                          ▷ See Algorithm 7.5
13: end for                               ▷ Cube schema heterogeneities repaired
```

As soon as all dimension and cube schemas have been integrated according to the above Algorithm 7.2, the extensions of the global dimensions and cubes are computed from the local Data Marts as specified below in Algorithm 7.3:

Algorithm 7.3 Overview of the Data Mart instance integration process

Input: Local Data Marts $DM_1, ..., DM_k$; global Data Mart $\hat{DM} = \{\hat{C}_1, ..., \hat{C}_{\hat{n}}; \hat{D}_1, ..., \hat{D}_{\hat{m}}\}$

Output: Dimension mappings $\bigcup_{j=1...\hat{m}}\{\mathcal{I}^{\hat{D}_j}\}$, cube mappings $\bigcup_{i=1...\hat{n}}\{\mathcal{I}^{\hat{C}_i}\}$.

```
1: for all $\hat{D}_j \in \{\hat{D}_1, ..., \hat{D}_m\}$ do
2:     Compute $\mathcal{I}^{\hat{D}_j}$     ▷ Dimension instance integration; see Subsection 7.2.3
3: end for
4: for all $\hat{C}_i \in \{\hat{C}_1, ..., \hat{C}_n\}$ do
5:     Compute $\mathcal{I}^{\hat{C}_i}$          ▷ Cube instance integration; see Subsection 7.2.3
6: end for                                        ▷ Instance integration finished
```

7.2.1 Resolve Schema-Instance Conflicts

The goal of the first phase of Data Mart integration is to recognize how the *dimensional entities* of the global multi-dimensional schema—building the *context* of facts in the Federated DW—are represented in the cubes of each autonomous DM. If different import schemas of Data Marts represent the dimensional entities once with schema elements and once with instances, schema–instance conflicts occur (cf. Section 5.1, page 62). It is necessary to repair these conflicts to model the dimensional context uniformly across all Data Marts. Otherwise, heterogeneity at the subsequent schema integration and instance integration phases possibly remains undetected (e.g., the schema–instance conflict could hide a dimensionality conflict).

Dimensional entities should be modelled as dimension members—i.e., at the instance level (cf. Definition 4.2 in Chapter 4, page 54). In practice, however, Data Mart schemas represent dimensional context indirectly within measure variables—i.e., at the schema level of facts. For example, the measures cost_p and cost_m in cube dwh1::treatment contain costs figures for the dimensional entities "personnel" and "material". Certainly more appropriate would be an additional dimension schema for the entity *costs category* with members personnel and material, such as in Data Mart dwh2 (cf. Chapter 2, Figure 2.1).

Schema–instance conflicts can be repaired with *pivoting* operations in two directions, depending on the desired schema of the global data cube. Pivoting either increases *dimensionality* or *arity*—i.e., respectively, the number of dimension and measure attributes—of cube import schemas. On the one hand, transforming the dimensional entities hidden in measure variables leads to the definition of a new context dimension. Thus, the import schema becomes more concise by merging several measures into a single one, while increasing its dimensionality. On the other hand, converting the members of an existing dimension into new measure variables is exactly the opposite operation, enriching the import schema's arity while reducing its dimensionality.

In particular, we specify the two pivoting operations as follows:

- *Merge measure variables, generate context dimension:* the introduction of a *context dimension* converts implicit fact context into members of the new dimension. Consequently, the previously hidden context of measure variables at the schema level is partially transformed into context information at the instance level (i.e., to the members of the newly generated context dimension). This operation increases the dimensionality of the import schema, but reduces the number of measure variables. In turn, the expressivity of the multi-dimensional schema decreases, leading to an "inflation" of cube cells. Thus, the number of cells in the import schema grows in order to express the same information as in the original schema.
- *Split measure variables, generate contextualized facts:* the conversion of existing dimension members into new, specialized measure variables generates *contextualized facts.* Consequently, the context information at the instance level (i.e., the dimension members) is being partially transformed into context information at the schema level (i.e., the measure variables). This operation decreases the dimensionality of the import schema, but increases the number of measure variables. In turn, the expressivity of the

multi-dimensional schema increases, leading to a “deflation” of cube cells. As a result, the number of cells in the import schema is reduced without any loss of information.

If a schema–instance conflict is detected among C^{DM} and $\hat{C}$, identify and apply the adequate pivoting operation. If the dimensionality of $\hat{C}$ is higher, merge measures and generate a context dimension for import schema C^{DM} (line 3 of Algorithm 7.2, see Definition 7.13). Alternatively, if the expressivity of $\hat{C}$ is higher, split measures and generate contextualized facts for import schema C^{DM} (line 5 of Algorithm 7.2, see Definition 7.14).

Example 7.14 (Data Mart integration – schema–instance conflicts): The global schema of the health insurance organization defines cube treatment with two measures cost_p and cost_m (cf. Figure 2.2). In contrast, local cube dwh2::treatment contains the only measure costs, plus dimension cost_cat. Thus, according to line 5 of Algorithm 7.2, we add the following operator to mapping $\mathcal{M}_{dwh2::treatment \rightsquigarrow g::treatment}$ to repair the conflict: $\xi_{costs \Rightarrow cost\_cat}$(dwh2::treatment). Besides the costs figures of the treatment cubes, no other schema–instance conflicts need to be repaired. As specified in Algorithm 7.2, the integration process among the autonomous Data Marts proceeds to the next phase (multi-dimensional schema integration). ◇

7.2.2 Integrate Dimension and Fact Schemata

In the second phase of the integration methodology the Federated DW administrator defines the appropriate mappings at the schema level in order to obtain homogeneous import schemas for both, the dimensions and facts of the autonomous Data Marts. Within this phase, the integration of dimension schemas has priority over the integration of fact schemas because cubes logically depend on dimensions in the multi-dimensional data model (see Chapter 4). Therefore, the dimension schemas should reach a “fixed” homogeneous state before examining the cube schemas for heterogeneities.

In general, the FedDW methodology for Data Mart integration follows the *minimum use strategy*, i.e. the reduction of import schemas to the parts in common with the global schema. In contrast, a *maximum use strategy* attempts to fully import the local schemas, keeping even elements that do not match the global schema. *Minimum match* is an intermediate strategy, amending minimum use import schemas with all information that can be safely reconstructed from the local data (e.g. calendar hierarchies, see Figure 7.3).

Minimum use integration ideally copes with the typical, general aims of OLAP analysis—precision and quality of data is more important than its quantity. For the OLAP analyst, fewer variables that are comparable over the entire data set of the federated DMs are more valuable than the maximum available data over all DMs. In order to utilize as much of the source data as possible, several slices of one and the same local cube can be defined as import schemas.

For each dimension D of some local Data Mart DM, the *dimension schema integration* process consists of the steps given in the following Algorithm 7.4. The result of dimension schema integration is the mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$, which contains the sequence of conversion operators specified during the integration process. If applied to the original import schema S_D, the mapping produces the

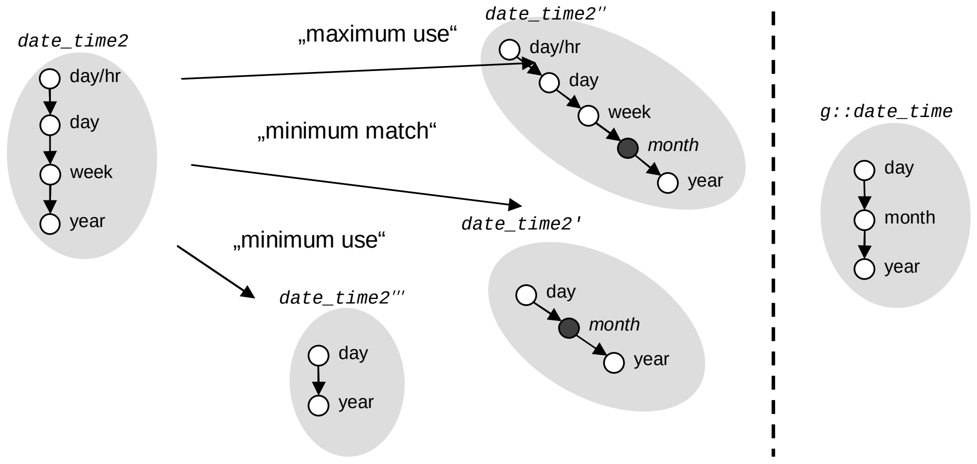


Figure 7.3: Mapping strategies—maximum use vs. minimum match vs. minimum use—demonstrated for dimension dwh2::date_time2. Notice that the month level must be generated – "reconstructed" – in the case of maximum use and minimum match strategies.

transformed dimension schema S'_D that is *homogeneous* with global dimension schema $S_{\hat{D}}$—i.e., $\mathcal{M}_{D \rightsquigarrow \hat{D}}(S_D) = S'_D$ (cf. Definition 7.16). Let operator $\overset{\cap}{\equiv}$ denote the intersection of equivalent levels between the level sets of two dimension schemas S_D and $S_{\hat{D}}$, such that $L_D \overset{\cap}{\equiv} L_{\hat{D}} = \{l \in L_D \mid \exists \hat{l} \in L_{\hat{D}} : l \equiv \hat{l}\}$.

In order to map the multi-dimensional space of local Data Mart DM to the global schema, the Federated DW administrator repeats the schema integration process for all pairs of dimensions $D \in DM$ and $\hat{D} \in \hat{DM}$ that model the same dimensional entity. For each such dimension D of the local, autonomous Data Mart, Algorithm 7.4 creates a semantic mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$. All remaining dimensions $D' \in DM$ that cannot be mapped to the global schema—$\{D' \in \{D_1, ..., D_m\} \mid \nexists \hat{D} \in \{\hat{D}_1, ..., \hat{D}_{\hat{m}}\} : S'_D \equiv S_{\hat{D}} \wedge d'(S'_D) \equiv d(S_{\hat{D}})\}$—are deleted from the import schema of Data Mart DM. Since the dimensions D' cannot be matched with the dimensional entities context of the global schema, mappings for these dimensions would be useless.

Algorithm 7.4 Determine mapping for local dimension schema S_D.

Input: Dimension schemas S_D (of local dimension $D \in DM$) and $S_{\hat{D}}$ (of dimension $\hat{D} \in \hat{DM}$ in global schema)

Output: Mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$

Description: Integrates S_D with $S_{\hat{D}}$ in three phases: (1) naming conflicts, (2) hierarchies of aggregation levels, (3) domain conflicts. The algorithm determines the appropriate Dimension Algebra operators to compose the mapping.

```
1:  **if** $name(D) \neq name(\hat{D})$ **then**
2:      Add $\zeta_{name(\hat{D}) \leftarrow name(D)}(D)$ to $D \rightsquigarrow \hat{D}$          ▷ Rename dimension
3:  **end if**
4:  **for all** $l \in L_D \mid \exists \hat{l} \in L_{\hat{D}} : l \simeq \hat{l}$ **do**
5:      **if** $name(l) \neq name(\hat{l})$ **then**
6:          Add $\zeta_{name(\hat{l}) \leftarrow name(l)}(l)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ ▷ Rename levels where necessary
7:      **end if**
8:      **if** $name(l.K) \neq name(\hat{l}.K)$ **then**
9:          Add $\zeta_{name(\hat{l}.K) \leftarrow name(l.K)}(S_l)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ ▷ Rename roll-up attribute
10:     **end if**
11:     **for all** $l.N_i \in S_l \mid \exists \hat{l}.\hat{N}_i \in S_{\hat{l}} : N_i = \hat{N}_i \wedge name(N_i) \neq name(\hat{N}_i)$ **do**
12:         Add $\zeta_{name(\hat{N}_i) \leftarrow name(N_i)}(S_l)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ ▷ Rename N-attributes of $l$
13:     **end for**
14: **end for**                                         ▷ All naming conflicts repaired
15: **if** $\neg(L_D \equiv L_{\hat{D}} \wedge H_D = H_{\hat{D}})$ **then**
16:     **for all** $l \in H_D, \hat{l} \in H_{\hat{D}} \mid l \simeq \hat{l} \wedge dom(l.K) \neq dom(\hat{l}.K)$ **do**
17:         Add $\gamma_{\theta(l.K)}(D)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$          ▷ Convert roll-up attribute domains
18:     **end for**
19:     **if** $L_D \overset{\cap}{\equiv} L_{\hat{D}} = \emptyset$ **then**
20:         goto end;                                   ▷ non-corresponding; abort
21:     **else if** $L_D \overset{\cap}{\equiv} L_{\hat{D}} \neq \emptyset$ **then**
22:         **if** $l_0^D \not\equiv l_0^{\hat{D}}$ **then**                         ▷ inner-level corresponding
23:             Identify $k \in H_D, \hat{k} \in H_{\hat{D}} \mid k \equiv \hat{k} \wedge \nexists(k' \in H_D, \hat{k}' \in H_{\hat{D}})$ :
    $k' \equiv \hat{k}' \wedge k \mapsto^+ k'$                                          ▷ New base level
24:             Add $\psi_k(S_D)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$              ▷ Roll-up to new base level
25:         **end if**                      ▷ $l_0^D \equiv l_0^{\hat{D}}$ – base-level corresponding
26:         Add $\pi_{(L_D \overset{\cap}{\equiv} L_{\hat{D}})}(S_D)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$          ▷ Minimum-use of levels
27:     **end if**
28: **end if**                    ▷ Hierarchies repaired – homogeneous level schemas
29: **for all** $l \in H_D, \hat{l} \in H_{\hat{D}} \mid l \simeq \hat{l}$ **do**           ▷ Check all pairs of conceptually
    corresponding levels $l \simeq \hat{l}$ for heterogeneous N-attribute domains
30:     **for all** $l.N_i \in S_l, \hat{l}.\hat{N}_i \in S_{\hat{l}} \mid N_i = \hat{N}_i \wedge dom(N_i) \neq dom(\hat{N}_i)$ **do**
31:         Add $\gamma_{\theta(l.N_i)}(D)$ to $\mathcal{M}_{D \rightsquigarrow \hat{D}}$          ▷ Convert domains of N-attributes
32:     **end for**
33: **end for**                                         ▷ Domain conflicts repaired
34: **return** $\mathcal{M}_{D \rightsquigarrow \hat{D}}$
```

Informally, the schema mapping process (Algorithm 7.4) performs the following steps to compute the mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$:

1. Check the dimension schemas for naming conflicts. Whenever necessary, change conflicting names in the following elements of S_D: (1) dimension name of D [line 2], (2) levels L_D in hierarchies [line 6], (3) level attributes $\forall l \in L_D : l.K$ in level schemas [line 9], and (4) non-dimensional attributes $\forall l \in L_D : l.N_1, ..., l.N_i, ..., l.N_k$ in level schemas [line 12].

2. For the conceptually corresponding levels $\forall l \in H_D, \hat{l} \in H_{\hat{D}} \mid l \simeq \hat{l}$ check the domains of roll-up attributes $l.K$ and $\hat{l}.K$. If it mismatches, i.e. $dom(l.K) \neq dom(\hat{l}.K)$, specify the necessary conversion function θ for level attribute $l.K$, converting its domain to $dom(\hat{l}.K)$—[line 17].

3. Check the correspondence among the sets of levels L_D. If the level sets L_D and $L_{\hat{D}}$ are intensionally corresponding ($L_D \equiv L_{\hat{D}}$—Definition 5.5, page 66) and have identical hierarchies (i.e., $L_D \equiv L_{\hat{D}} \wedge H_D = H_{\hat{D}}$—see [line 15]), proceed directly with step 4. Otherwise, eliminate heterogeneities among the two level sets as follows:

   (a) If L_D and $L_{\hat{D}}$ are non-corresponding ($L_D \overset{\cap}{\equiv} L_{\hat{D}} = \emptyset$), stop the integration process for dimension D [line 20]. In this case, the local dimension D^{DM} and global dimension $\hat{D}$ do not have any common content (see Example 5.5).

   (b) If L_D and $L_{\hat{D}}$ are inner-level corresponding ($L_D \overset{\cap}{\equiv} L_{\hat{D}} \neq \emptyset \wedge l_0^D \not\equiv l_0^{\hat{D}}$—see Example 5.6), (1) identify the most fine-grained, common level $k \in H_D, \hat{k} \in H_{\hat{D}} : k \equiv \hat{k}$ [line 23], and (2) identify all common, intensionally equivalent levels $L_D \overset{\cap}{\equiv} L_{\hat{D}}$ while deleting the other, remaining levels of L_D [lines 21–26].

   (c) If L_D and $L_{\hat{D}}$ are base-level corresponding ($L_D \overset{\cap}{\equiv} L_{\hat{D}} \neq \emptyset \wedge l_0^D \equiv l_0^{\hat{D}} \wedge L_D \not\equiv L_{\hat{D}}$—see Example 5.7), identify all common, intensionally equivalent levels $L_D \overset{\cap}{\equiv} L_{\hat{D}}$ while deleting the other, remaining levels of L_D [lines 21–26].

   (d) If L_D and $L_{\hat{D}}$ are flat corresponding ($L_D \equiv L_{\hat{D}} \wedge l_0^D \equiv l_0^{\hat{D}} \wedge H_D \neq H_{\hat{D}}$—see Example 5.8), keep only the base level l_0^D and delete all other levels of L_D [lines 21–26].

4. By now, the levels among L_D and $L_{\hat{D}}$ are *homogeneous*, i.e. corresponding (cf. Definition 5.5) and with identical hierarchies. Finally, check the domains of non-dimensional attributes among the level schemas $S(L_D)$ and $S(L_{\hat{D}})$. For all overlapping non-dimensional attributes N_i $(1 \leq i \leq k)$ of every level $l \in L_D$, i.e. $\forall l.N_i \in S_l, \hat{l}.\hat{N}_i \in S_{\hat{l}} \mid N_i = \hat{N}_i$, check if the attribute domains are the same. If not, i.e. $dom(l.N_i) \neq dom(\hat{l}.\hat{N}_i)$, specify the necessary conversion function $\theta(l.N_i)$ for the N-attribute $l.N_i$, converting its domain to $dom(\hat{l}.\hat{N}_i)$—[line 31].

   Notice that domain conflicts of roll-up attributes must be repaired before checking the correspondence of level sets ($L_D \overset{\cap}{\equiv} L_{\hat{D}}$, see step 2). Otherwise, the algorithm would ignore *conceptually corresponding* levels that become equivalent after repairing domain conflicts between their level schemas (cf. Definitions 5.3 and 5.4, page 65).

Now, the dimension schema integration process among S_D and $S_{\hat{D}}$ is complete. Its result is the mapping $D \rightsquigarrow \hat{D}$, which contains the sequence of all DA operators specified in the process explained above. Having mapped all dimensions of local, autonomous Data Mart DM to the global multi-dimensional schema, the Data Mart integration algorithm proceeds to the next phase, mapping the cubes of DM to $\hat{DM}$ (cf. line 12 in Algorithm 7.2, page 102).

Example 7.15 (Dimension schema integration process): Algorithm 7.4 creates the following semantic mappings for the dimensions of local Data Mart dwh1 (cf. Figure 2.1):

- All mappings $\mathcal{M}_{dwh1::date\_time \rightsquigarrow g::date\_time}$, $\mathcal{M}_{dwh1::method \rightsquigarrow g::method}$, $\mathcal{M}_{dwh1::patient \rightsquigarrow g::patient}$ and $\mathcal{M}_{dwh1::drug \rightsquigarrow g::drug}$ remain empty (no conflicts to repair).
- Dimension dhw1::phys is deleted from the import schema.

Moreover, Algorithm 7.4 creates the following semantic mappings for the dimensions of local Data Mart dwh2:

- $\mathcal{M}_{dwh2::date\_time2 \rightsquigarrow g::date\_time}$: rename dimension $\zeta_{date\_time \leftarrow date\_time2}$(dwh2::date_time2) – line 2; roll-up to new base level $\psi_{[day]}$(dwh2::date_time2) – lines 23–24; project levels $\pi_{\{[day],[year]\}}$(dwh2::date_time2) – line 26.
- $\mathcal{M}_{dwh2::method \rightsquigarrow g::method}$: rename non-dimensional attribute $\zeta_{description \leftarrow descrpt}$(dwh2::method.l_method) – line 12.
- Mappings $\mathcal{M}_{dwh2::patient \rightsquigarrow g::patient}$ and $\mathcal{M}_{dwh2::drug \rightsquigarrow g::drug}$ remain empty (no conflicts to repair).
- Finally, dimension dwh2::cost_cat is not considered, because it has been converted to measure attributes in the previous Example 7.14. ◇

For each cube $C \in DM$, the *cube integration process* consists of the steps given in the following Algorithm 7.5. The result of cube schema integration is the mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$, which contains the sequence of conversion operators specified during the cube integration process. If applied to the original import schema S_C, the mapping produces the transformed cube schema S'_C that is *homogeneous* to some global cube schema $S_{\hat{C}}$—i.e., $\mathcal{M}_{C \rightsquigarrow \hat{C}}(S_C) = S'_C$. Let operator $\overset{\cap}{\equiv}$ denote the intersection of equivalent dimensional attributes between two cube schemas S_C and $S_{\hat{C}}$, such that $A_C \overset{\cap}{\equiv} A_{\hat{C}} = \{A \in S_C \mid \exists \hat{A} \in S_{\hat{C}} : A \equiv \hat{A}\}$.

To finish the schema integration phase for autonomous Data Mart DM, the cube schema integration process is repeated for all pairs of cubes $C \in DM$ and $\hat{C} \in \hat{DM}$ whose schemas contain at least one identical dimensional entity. (Intuitively, this means that the integration methodology considers all cubes that are drill-across compatible with one of the cubes defined in the global schema.) For each such cube C of the local, autonomous Data Mart, Algorithm 7.5 creates a semantic mapping. All other cubes $C' \in DM$ that cannot be mapped to the global, multi-dimensional schema—i.e., $\{C' \in \{C_1, ..., C_n\} \mid \nexists \hat{C} \in \{\hat{C}_1, ..., \hat{C}_{\hat{n}}\} : S'_C \equiv S_{\hat{C}} \wedge c'(S'_C) \equiv c(S_{\hat{C}})\}$—are deleted from the import schema of Data Mart DM. Since the dimensional contexts of all cubes C' are distinct from the global multi-dimensional schema, mappings for these cubes would be useless.

Algorithm 7.5 Determine mapping for local cube schema S_C.

Input: Cube schemas S_C (of local cube $C \in DM$ and $S_{\hat{C}}$ (of cube $\hat{C} \in \hat{DM}$ in global schema)

Output: Mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$

Description: Integrates S_C with $S_{\hat{C}}$ in three phases: (1) naming conflicts, (2) dimensionality, (3) domain conflicts. The algorithm determines the appropriate Cube Algebra conversion operators to compose the mapping.

```
1: if $name(C) \neq name(\hat{C})$ then
2:     Add $\zeta_{name(\hat{C}) \leftarrow name(C)}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$    ▷ Rename cube
3: end if
4: for all $A[l] \in S_C \mid \exists \hat{A}[\hat{l}] \in S_{\hat{C}} : A \simeq \hat{A} \wedge name(A) \neq name(\hat{A})$ do
5:     Add $\zeta_{name(\hat{A}) \leftarrow name(A)}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ ▷ Rename dimensional attributes
6: end for
7: for all $M \in S_C \mid \exists \hat{M} \in S_{\hat{C}} : M = \hat{M} \wedge name(M) \neq name(\hat{M})$ do
8:     Add $\zeta_{name(\hat{M}) \leftarrow name(M)}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$    ▷ Rename measures of $S_C$
9: end for    ▷ All naming conflicts repaired
10: if $\neg(A_C \equiv A_{\hat{C}})$ then
11:     if $A_C \overset{\cap}{\equiv} A_{\hat{C}} = \emptyset$ then
12:         goto end;    ▷ non-corresponding; abort
13:     else if $A_C \overset{\cap}{\equiv} A_{\hat{C}} \neq \emptyset$ then
14:         for all $A[l] \in A_C \overset{\cap}{\equiv} A_{\hat{C}}, \hat{A}[\hat{l}] \in A_{\hat{C}} \mid A \simeq \hat{A} \wedge l \mapsto^{+} \hat{l}$ do
15:             Add $\varrho_{A[l \Rightarrow \hat{l}]}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$    ▷ Roll-up all $A$ to equivalent
    aggregation levels with $\hat{A}$
16:         end for
17:         Add $\pi_{(A_C \overset{\cap}{\equiv} A_{\hat{C}})}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$    ▷ Roll-up all $A' \notin A_C \overset{\cap}{\equiv} A_{\hat{C}}$ to [all]
18:     end if
19: end if    ▷ Dimensionality repaired – homogeneous cube schemas
20: for all $M \in S_C, \hat{M} \in S_{\hat{C}} \mid M = \hat{M}$ do ▷ Check all pairs of measures $M, \hat{M}$
    for heterogeneous attribute domains
21:     if $dom(M) \neq dom(\hat{M})$ then
22:         Add $\gamma_{M\theta(v)}(C)$ to $\mathcal{M}_{C \rightsquigarrow \hat{C}}$    ▷ Convert domains of measures with
    appropriate function $\theta$ and fixed value $v$ (cf. Definition 7.10)
23:     end if
24: end for    ▷ Domain conflicts repaired
25: return $\mathcal{M}_{C \rightsquigarrow \hat{C}}$
```

Informally, the cube mapping process (Algorithm 7.5) performs the following steps to compute the mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$:

1. Check the cube schemas for naming conflicts. Whenever necessary, change conflicting names in the following elements of schema S_C: (1) cube name of C [line 2], (2) dimensional attributes A_C [line 5], (3) measure attributes M_C [line 8].

2. Check the correspondence among of the cube schemas S_C and $S_{\hat{C}}$, based on the two sets of dimension attributes. If the dimension attribute sets A_C and $A_{\hat{C}}$ are corresponding ($A_C \equiv A_{\hat{C}}$—Definition 5.8, page 69), proceed

directly with step 3 [line 10]. Otherwise, eliminate heterogeneities among the dimension attribute sets as follows:

(a) If S_C and $S_{\hat{C}}$ are non-corresponding ($A_C \overset{\cap}{\equiv} A_{\hat{C}} = \emptyset$), stop integration for cube C [line 12]. In this case, local cube C^{DM} and global cube $\hat{C} \in \hat{DM}$ do not have any comparable content since they represent facts in ontologically different dimensional context (see Example 5.12).

(b) If S_C and $S_{\hat{C}}$ are partially corresponding ($A_C \overset{\cap}{\equiv} A_{\hat{C}} \neq \emptyset$—see Example 5.13), (1) identify the common dimension attributes $A_C \overset{\cap}{\equiv} A_{\hat{C}}$ [line 13], and (2) specify the appropriate aggregation functions to roll-up the other, remaining dimension attributes $A' \notin A_C \overset{\cap}{\equiv} A_{\hat{C}}$ to the [all]-level [line 17].

(c) For all pairs of $A[l] \in A_C \cap A_{\hat{C}}$ and $\hat{A}[\hat{l}] \in A_{\hat{C}}$ with $A \simeq \hat{A}$, check whether $l \equiv \hat{l}$ is true. If $l \mapsto^{+} \hat{l}$, roll-up dimension attribute A to level $\hat{l}$, such that A becomes equivalent to $\hat{A}$, i.e. $A \equiv \hat{A}$ [line 15].

3. By now, the dimensional attributes among cube schemas S_C and $S_{\hat{C}}$ are *homogeneous*, i.e. fully corresponding (Definition 5.8). Next, inspect the domains of all measure attributes among the cube schemas: for all overlapping measures $M \in M_C, \hat{M} \in M_{\hat{C}} \mid M = \hat{M}$ check if $dom(M) = dom(\hat{M})$. If domain conflicts among the measure attributes are detected, specify the appropriate conversion functions for the measure attributes M, converting their domains to the respective domains of $\hat{M}$ [line 22].

   Notice that the non-overlapping measure attributes $M' \in M_C \setminus M_{\hat{C}} = \{M \in M_C \mid \nexists \hat{M} \in M_{\hat{C}} : M = \hat{M}\}$ remain in the cube import schema S_C, i.e. they are not deleted. Importantly, *every* measure of facts probably contains information that is potentially valuable to the business analysts, even if this measure exists only at a subset of Data Marts in the federation. Merging cube import schemas among which the measures do not overlap exactly, creates facts with null values, but the resulting facts nonetheless contain more information.

Now, the cube schema integration process among S_C and $S_{\hat{C}}$ is complete. Its result is the mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$, which contains the sequence of FA operators specified in the process explained above. As soon as the schemas of all pairs of cubes $C \in DM, \hat{C} \in \hat{DM}$ have been processed, the schema integration phase of the FedDW methodology is completed.

Example 7.16 (Cube schema integration process): Algorithm 7.5 determines the following semantic mappings for the cubes of local Data Mart dwh1 (cf. Figure 2.1): $\mathcal{M}_{dwh1::treatment \rightsquigarrow g::treatment}$ and $\mathcal{M}_{dwh1::medication \rightsquigarrow g::medication}$, both of which remain empty (no conflicts to repair).

Moreover, Algorithm 7.5 determines the following semantic mappings for the cubes of local Data Mart dwh2:

- $\mathcal{M}_{dwh2::treatment \rightsquigarrow g::treatment}$: convert measure domain $\gamma_{usd2eur(costs)}$(dwh2::treatment) with function usd2eur(), which must be pre-defined within the ROLAP platform of the global Data Mart; rename dimensional attribute $\zeta_{date \leftarrow date/hr}$(dwh2::treatment).
- $\mathcal{M}_{dwh2::medication \rightsquigarrow g::medication}$ remains empty (no conflicts to repair). $\diamond$

7.2.3 Consolidate Dimension and Fact Instances

The goal of the third and final phase of Data Mart integration is converting the instances of the transformed dimensions and facts defined previously (see Subsections 7.2.1 and 7.2.2), such that they conform to the global schema. At the beginning of this phase, both the transformed dimension schemas S'_D and the transformed cube schemas S'_C of every autonomous Data Mart $DM_1, ..., DM_k$ are homogeneous with the global Data Mart $\hat{DM} = \{\hat{C}_1, ..., \hat{C}_{\hat{n}}, \hat{D}_1, ..., \hat{D}_{\hat{m}}\}$. All heterogeneities at the instance level that still exist among the dimension members and cells between each $DM \in \{DM_1, ..., DM_k\}$ and $\hat{DM}$ are now resolved. Again, the removal of the conflicts among dimension members has priority over the conflicts among cube cells.

Recall that dimension integration and cube integration repairs any existing heterogeneity between one of the autonomous Data Marts and the global multi-dimensional schema (cf. Algorithms 7.4 and 7.5). In turn, the unified sets of transformed dimension members and cube cells generated by the mappings $\bigcup_{i=1...k}\{D^{DM_i} \rightsquigarrow \hat{D}\}$ and $\bigcup_{i=1...k}\{C^{DM_i} \rightsquigarrow \hat{C}\}$ constitute the extensions of the global dimensions $\hat{D}$ and global cubes $\hat{C}$. In order to correctly integrate all members and cells, any heterogeneous instances must be resolved (e.g., overlapping subsets of facts, or heterogeneous roll-up functions for members).

In contrast to schema integration (Subsection 7.2.2), the integration of dimension and fact instances defines *set-oriented mappings* $\mathcal{I}^{\hat{D}_j}$ (with $j = 1...\hat{m}$) and $\mathcal{I}^{\hat{C}_i}$ (with $i = 1...\hat{n}$)—for the dimensions $\{\hat{D}_1, ..., \hat{D}_j, ..., \hat{D}_{\hat{m}}\}$ and cubes $\{\hat{C}_1, ..., \hat{C}_i, ..., \hat{C}_{\hat{n}}\}$ of $\hat{DM}$, respectively—between the local Data Marts $\{DM_1, ..., DM_k\}$ and the global schema $\hat{DM}$. The integration of dimension and cube instances aims at defining the correct extension of the global multi-dimensional schema by unifying the extensions of all autonomous Data Marts. Only the set-oriented view on all Data Marts participating in the federation—i.e., the simultaneous integration of all local Data Mart extensions—is appropriate for this purpose.

For each dimension $\hat{D} \in \hat{DM}$, the *member integration process* unifies the dimension instances $DM_1.d(S'_D), ..., DM_k.d(S'_D)$ across the autonomous Data Marts $DM_1, ..., DM_k$ in the Federated DW system, thus specifying the global dimension instance $\hat{d}(S_{\hat{D}})$. Recall that dimension schema integration defines a transformed schema $S'_D = \mathcal{M}_{D \rightsquigarrow \hat{D}}(S_D)$ that is homogeneous for dimension D across all autonomous Data Marts $DM_1, ..., DM_k$. The following steps are repeated for every level $\hat{l} \in H_{\hat{D}}$:

1. For all sets of levels $\{l_1 \equiv \hat{l} \in DM_1.H_D, ..., l_k \equiv \hat{l} \in DM_k.H_D\}$ check the correspondence among their member sets:
   (a) If the level extensions overlap ($members(l_1) \in DM_1.d \cup ... \cup members(l_k) \in DM_k.d \neq \emptyset$, cf. Example 5.18), decide which set operation to apply on the overlapping subsets of members (e.g., UNION, MINUS, INTERSECT, etc.). Disjoint member subsets are merged using the UNION operator by default.
   (b) If the level extensions are disjoint ($members(l_1) \in DM_1.d \cup ... \cup members(l_k) \in DM_k.d = \emptyset$), always apply the UNION operation to merge their member sets. Skip the following step 2.

2. Repair value conflicts that occur among single members of overlapping level extensions: for $i = 1...k$ let $l_i = \{l \in DM_i.H_D \mid l \equiv \hat{l}\}$, and let $ext(\hat{l}) = \bigcup_{i=1...k}\{members(l_i)\}$. For all members $\hat{m} \in ext(\hat{l})$, let $t_{\hat{m}} = \{\hat{m} \in ext(\hat{l}) \mid \exists m_1 \in members(l_1) \times ... \times \exists m_k \in members(l_k) : \hat{m} \equiv m_1 \equiv ... \equiv m_k\}$ be the tuple of equivalent members across the local Data Marts $DM_1, ..., DM_k$.

   (a) Check if the $\hat{l}$-members consistently roll-up to the same parent member. Overwrite the wrong roll-up function(s) accordingly: $\forall m' \in t_{\hat{m}} : \rho_D(m') \not\equiv \rho_D(\hat{m})$, add operator $\Omega_{m' \to \rho_D(\hat{m})}(level(m'))$ to mapping $\mathcal{I}^{\hat{D}}$ (cf. Definition 7.5, Example 5.16).

   (b) Check if the non-dimensional attribute values of the $\hat{l}$-members are consistent. For each N-attribute $\hat{N} \in S_{\hat{l}}$, repair all contradictory values accordingly: $\forall m' \in t_{\hat{m}} : m'(\hat{N}) \neq \hat{m}(\hat{N})$ add operator $\delta_{\hat{m}(\hat{N}) \leftarrow m'(\hat{N})}(level(m').\hat{N})$ to mapping $\mathcal{I}^{\hat{D}}$ (cf. Definition 7.3, Example 5.17).

Now the member integration process for the dimension instances $d(S'_D)$ of dimension D among the autonomous Data Marts $DM_1, ..., DM_k$ is complete. The result of member set integration is the mapping $\mathcal{I}^{\hat{D}}$, which contains the sequence of the set-oriented and value-oriented conversion operators defined in the process above. If applied to the dimension instances $DM_1.d(S'_D), ..., DM_k.d(S'_D)$, mapping $\mathcal{I}^{\hat{D}}$ generates the global dimension instance $\hat{d}(S_{\hat{D}})$—i.e., $\mathcal{I}^{\hat{D}}(DM_1.d, ..., DM_k.d) = \hat{d}(S_{\hat{D}})$.

Repeat the member integration process for all further dimensions $\hat{D}$ shared by the local, autonomous Data Marts $DM_1, ..., DM_k$. Thus, the number of iterations of the member integration process corresponds to the number $\hat{m}$ of dimensions that are defined in *every* local Data Mart. As soon as the members of all dimensions are integrated, the instances of all dimensions in the global multi-dimensional schema can be computed.

Example 7.17 (Member integration process): The member integration process described above creates the following semantic mappings for the dimensions of global Data Mart g (cf. Figure 2.2):

- $\mathcal{I}^{g::date\_time}$: merge dimensions μ ({dwh1::date_time, dwh2::date_time2}); no other conflicts to repair.
- $\mathcal{I}^{g::method}$: μ ({dwh1::method, dwh2::method}); no other conflicts to repair.
- $\mathcal{I}^{g::patient}$: μ ({dwh1::patient, dwh2::patient}); no other conflicts to repair.
- $\mathcal{I}^{g::drug}$: overwrite roll-up function $\Omega_{‘Novartis’ \mapsto ‘Bayer’}$(dwh1::drug.l_drug); correct N-attribute value $\delta_{‘30pcs.’ \leftarrow ‘25pcs.’}$(dwh2::drug.l_drug.pkg_size); μ ({dwh1::drug, dwh2::drug}). ◇

Finally, for each cube $\hat{C} \in \hat{DM}$ the subsequent *cells integration process* unifies the cube instances $DM_1.c(S'_C), ..., DM_k.c(S'_C)$ across the autonomous Data Marts $DM_1, ..., DM_k$, thus specifying how to fill the global cube cells $\hat{c}(S_{\hat{C}})$. Cells integration consists of the following steps:

1. In case that the cube instances $DM_1.c(S'_C), ..., DM_k.c(S'_C)$ overlap (i.e., $\{DM_1.M_C \cap ... \cap DM_k.M_C\} \neq \emptyset$—see Example 5.19) and the cells should be merged using the UNION set operation, determine the correspondence among the entities of the universe of discourse modelled by the cubes. Similarly to the so-called "stock" and "flow" semantics of measure variables [Lenz and Shoshani, 1997], overlapping facts express either an identical-related or context-related semantics:

   - *Identical-related*: overlapping, identical-related facts describe conflicting measures on the same real-world entity in the same context (e.g., weather statistics for the same time and some particular location, or stock-market prices for a particular day and company share). In most cases, measure values of identical-related facts cannot be meaningfully summarized. Thus, identical-relationship is similar to measures with stock semantics [Lenz and Shoshani, 1997].

     In case of overlapping cube instances with identical-related cells either (a) prefer the measure values of one given cube instance $\{DM_1.c, ..., DM_k.c\}$, or alternatively (b) extend the cells with a new dimension, called the *context dimension*:

     (a) If preferring measure values of some cube $\bar{c}$, add operator $\mu(\mathbb{C}[N, \theta])$ to mapping $\mathcal{I}^{\hat{C}}$, whereby $\mathbb{C} = \{DM_1.C, ..., DM_k.C\}$, $N = \bigcap_{i=1...k}(DM_i.M_C)$, and $\theta = prefer(\bar{c})$ with $\bar{c} \in \{DM_1.c, ..., DM_k.c\}$ (cf. Definition 7.15).

     (b) Conversely, if extending the cube instances, add operator $\varepsilon_{\hat{A}[\hat{l}]=name(c')}(c')$ to mapping $\mathcal{I}^{\hat{C}}$, enriching each cube instance $c' \in \{DM_1.c, ..., DM_k.c\}$ with some degenerate context dimension named $\hat{A}$, having only level $\hat{l}$ (cf. Definition 7.11).

   - *Context-related*: overlapping, context-related facts describe either (1) the same real-world entity in different contexts (e.g., weather statistics for different times and some particular location, or stock-market prices for a particular company share on different stock exchanges), or (2) different real-world entities in similar contexts (e.g., sales figures of several subsidiaries). The challenging problem with these overlapping cells is to correctly recognize *what* they describe, and if aggregating their measures across autonomous cubes produces meaningful results. Sometimes the measures of such overlapping cells are summarizable, sometimes not, depending on the entities the cells represent. Only an expert of the underlying application domain—e.g., the Federated DW administrator—can reliably decide on the summarizability problem. Thus, context-relationship is similar to measures with flow semantics [Lenz and Shoshani, 1997].

     In case of overlapping cube instances with context-related cells either (a) specify the appropriate aggregation operation for merging the measure values of the given cube instances $\{DM_1.c, ..., DM_k.c\}$, if the *aggregated value* is the interesting piece of information for the analyst; or alternatively (b) extend the cells with a new dimension, the so-called context dimension, if the *original source and context* of the cells is important for the analyst:

(a) If aggregating measure values, add operator $\mu(\mathbb{C}[N, \theta])$ to mapping $\mathcal{I}^{\hat{C}}$, whereby $\mathbb{C} = \{DM_1.C, ..., DM_k.C\}$, $N = \bigcap_{i=1...k}(DM_i.M_C)$, and $\theta \in \{sum, avg, min, max\}$ with (cf. Definition 7.15).

(b) Conversely, if extending the cube instances, add operator $\varepsilon_{\hat{A}[\hat{l}]=name(c')}(c')$ to mapping $\mathcal{I}^{\hat{C}}$, enriching each cube instance $c' \in \{DM_1.c, ..., DM_k.c\}$ with some degenerate context dimension named $\hat{A}$, having only level $\hat{l}$ (cf. Definition 7.11).

2. If the cube instances $DM_1.c(S'_C), ..., DM_k.c(S'_C)$ are disjoint (i.e., $DM_1.M_C \cap ... \cap DM_k.M_C = \emptyset$—see Example 5.20), simply merge their cells. Add operator $\mu(\mathbb{C}[\emptyset, \theta])$ to mapping $\mathcal{I}^{\hat{C}}$, whereby $\mathbb{C} = \{C \in DM_1, ..., C \in DM_k\}$. Notice that operator θ is arbitrary because it has no effect (overlapping cells cannot occur). As explained in Example 5.20, the μ operator applied to disjoint cube instances computes a global cube instance $\hat{c}(S_{\hat{C}})$ with cube schema $A_{\hat{C}}$, $M_{\hat{C}} = DM_1.M_C \cup ... \cup DM_k.M_C$, which accommodates all measure values. Thus, measure values from different cube instances can never collide since every cube instance fills only a subset of measures $M_{\hat{C}}$. However, the cells will contain null values.

Now the cells integration process for the cube instances $c(S'_C)$ among the autonomous Data Marts DM_1, ..., DM_k is complete. Its result is mapping $\mathcal{I}^{\hat{C}}$, containing the sequence of set-oriented conversion operators specified in the process explained above. If applied to the local cube instances $DM_1.c(S'_C), ..., DM_k.c(S'_C)$, mapping $\mathcal{I}^{\hat{C}}$ generates the global cube extension $\hat{c}(S_{\hat{C}})$—i.e., $\mathcal{I}^{\hat{C}}(DM_1.c, ..., DM_k.c) = \hat{c}(S^G_{\hat{C}})$. Repeat the cells integration process for all further cubes $\hat{C}$ shared by the local, autonomous Data Marts $DM_1..., DM_k$.

Example 7.18 (Cells integration process): The cells integration process described above creates the following semantic mappings for the cubes of global Data Mart g (cf. Figure 2.2):

- $\mathcal{I}^{g::treatment}$: merge cubes $\mu(\{$dwh1::treatment, dwh2::treatment$\}$ $[\{$method, date, cost_cat$\}$, sum$])$.
- $\mathcal{I}^{g::medication}$: merge cubes $\mu(\{$dwh1::medication, dwh2::medication$\}$ $[\{$patient, drug, date_time$\}$, sum$])$

The Data Mart integration is now finished for the treatment and medication cubes of the health insurance case study. Notice that an operator ξ (pivot split measures) has been added to mapping $\mathcal{I}^{g::treatment}$ in the previous Example 7.14. Thus, the final phase of the Data Mart integration methodology only adds the μ (merge cubes) operators. $\diamond$

7.3 Summary

In order to integrate autonomous Data Mart schemas, numerous heterogeneities must be considered, as analyzed in [Berger and Schrefl, 2006] and discussed in the Section 7.2. The Dimension and Fact Algebra introduced in [Berger and Schrefl, 2008] predefine a rich set of conversion operators that address all heterogeneities discussed in Chapter 5. The following Table 7.1 lists the conversion operators available in the FedDW approach with the conflicts addressed. The table first specifies all unary operators of Fact respectively Dimension Algebra, then gives all n-ary operators.

Table 7.1: Predefined conversion operators of Dimension Algebra / Fact Algebra for facts respectively dimensions

Dimensions: conflicts	**Relevant operator of Dimension Algebra**
Heterogeneous hierarchies	Project levels $\pi_{P \subset L_D}(d)$, or aggregate hierarchy $\psi_{l \in L_D}(d)$ (cf. Examples 7.2 and 7.3, respectively)
Domain conflicts (levels, non-dimensional attributes)	Convert attribute domains: $\gamma_{\theta \bar{N}}(d)$ (Definition 7.4)
Naming conflicts (levels)	Rename attributes: $\zeta_{l' \leftarrow l}(D)$ (Definition 7.2)
Naming conflicts (non-dimensional attributes)	Rename non-dimensional attributes: $\zeta_{N' \leftarrow N}(S_l)$ (Definition 7.2)
Overlapping members (dimension extensions)	Merge sets of members: $\mu(\mathbb{D})$ (Definition 7.6)
Heterogeneous roll-up functions in hierarchies	Overwrite roll-up hierarchies: $\Omega_{m \mapsto v}(d.l)$ (Definition 7.5)
Conflicting values of non-dimensional attributes	Correct attribute values: $\delta_{w \leftarrow v}(d.l.\bar{N})$ (Definition 7.3)
Facts: conflicts	**Relevant operator of Fact Algebra**
Schema-instance conflicts	Merge measures: $\chi_{L \Rightarrow \bar{M}, \bar{A}}(c)$ (Definition 7.13)
Schema-instance conflicts	Split measures: $\xi_{\bar{M} \Rightarrow A'}(c)$ (Definition 7.14)
Dimensionality	Project: $\pi_{(L \subset A_C)}(C)$ (Definition 7.9)
Different measures	Delete measure(s): $\lambda_{(N \subset M_C)}(C)$ (Definition 7.9)
Domain conflicts (measures)	Convert domains: $\gamma_{\bar{M} \theta v}(c)$ (Definition 7.10)
Naming conflicts (measures, dimension attributes)	Rename: $\zeta_{A' \leftarrow A}(C)$ and $\zeta_{M' \leftarrow M}(C)$ (Definition 7.9)
Heterogeneous base levels	Roll-up dimension attributes: $\varrho_{A[l \Rightarrow \hat{l}]}(c)$ (Definition 7.12)
Overlapping cube cells (fact extensions)	Merge facts: $\mu(\mathbb{C}[N, \theta])$ (Definition 7.15)
	Enrich facts with context dimension (Def. 7.11): $\varepsilon_{\hat{A}[l]=v}(c)$ – refer to existing level l $\varepsilon_{\hat{A}[\hat{l}]=v}(c)$ – new dimension $\hat{A}$ with $H_{\hat{A}} = \{\hat{l} \mapsto l_{all}\}$

The definition of semantic mappings between some local, autonomous Data Mart $DM \in \{DM_1, ..., DM_k\}$, with $DM = \{C_1, ..., C_n, D_1, ..., D_m\}$, and global Data Mart $\hat{DM} = \{\hat{C}_1, ..., \hat{C}_{\hat{n}}, \hat{D}_1, ..., \hat{D}_{\hat{m}}\}$—whereby n and $\hat{n}$, as well as m and $\hat{m}$ are not necessarily equal—comprises of the following parts:

- For each dimension $\{D \in \{D_1, ..., D_m\} \mid \exists \hat{D} \in \{\hat{D}_1, ..., \hat{D}_m\} : S'_D \equiv S_{\hat{D}} \wedge d'(S'_D) \equiv d(S_{\hat{D}})\}$ of DM, the dimension schema integration process defines mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$. The converted schema of D with mapping $\mathcal{M}_{D \rightsquigarrow \hat{D}}$ is intentionally equivalent to $S_{\hat{D}}$: $\mathcal{M}_{D \rightsquigarrow \hat{D}}(S_D) = S'_D \equiv S_{\hat{D}}$.

  $\rightarrow$ See Algorithm 7.4 (page 106) in Subsection 7.2.2.
- The members of all dimensions D corresponding to global dimension $\hat{D}$ among the local, autonomous Data Marts $DM_1, ..., DM_k$ are unified with the mapping $\mathcal{I}^{\hat{D}}$, specifying how to repair heterogeneities among member subsets and single instances. Mapping $\mathcal{I}^{\hat{D}}$ is defined by the member integration process.

  $\rightarrow$ See Subsection 7.2.3, pp. 111.
- For each cube $\{C \in \{C_1, ..., C_n\} \mid \exists \hat{C} \in \{\hat{C}_1, ..., \hat{C}_n\} : S'_C \equiv S_{\hat{C}}\}$ of DM, the cube schema integration process defines mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$. The converted schema of C with mapping $\mathcal{M}_{C \rightsquigarrow \hat{C}}$ is intentionally equivalent to $S_{\hat{C}}$: $\mathcal{M}_{C \rightsquigarrow \hat{C}}(S_C) = S'_C \equiv S_{\hat{C}}$.

  $\rightarrow$ See Algorithm 7.5 (page 109) in Subsection 7.2.2.
- The cells of all cubes C corresponding to global cube $\hat{C}$ among the local, autonomous Data Marts $DM_1, ..., DM_k$ are merged with the mapping $\mathcal{I}^{\hat{C}}$, specifying how to handle overlapping subsets of cube cells among all Data Marts. Mapping $\mathcal{I}^{\hat{C}}$ is defined during the cells integration process.

  $\rightarrow$ See Subsection 7.2.3, pp. 111.

Chapter 8

SQL-MDi—the Distributed OLAP Language

Contents

This chapter introduces *SQL-MDi* (SQL for Multi-Dimensional Integration), a query language that provides for both, the integration of autonomous Data Marts, and OLAP querying of the integrated multi-dimensional data, based on the SQL standard. SQL-MDi supports the general integration methodology for autonomous Data Marts introduced in the previous Chapter 7. It extends a standard OLAP query—formulated in SQL—with a *prologue* that computes a *virtual global cube* from several local cubes of autonomous Data Marts. The query clauses of the proposed SQL-MDi language provide numerous conversion operators, for which the Dimension Algebra and Fact Algebra are one possible, procedural implementation (introduced in Section 7.1). The current chapter specifies the syntax of SQL-MDi, explains how its operators address the heterogeneities analyzed in Chapter 5 and correspond to the Dimension/Fact Algebra. Numerous example statements illustrate the use of the SQL-MDi prologue.

The query language *SQL-MDi* (SQL for Multi-Dimensional Integration) was introduced in [Berger and Schrefl, 2006] to support the integration of several autonomous Data Marts. As its name suggests, SQL-MDi is based on the well-known SQL standard [(ISO), 1992]. The SQL-MDi language supports distributed OLAP queries, i.e. OLAP queries evaluated across several data cubes of autonomous Data Marts. As such, SQL-MDi can be employed as the query language in a Federated DW environment conforming to the reference architecture introduced in Chapter 6 (see pp. 81).

An SQL-MDi *statement* acts as the *prologue* of a standard SQL query with grouping and aggregation operations, as commonly used for OLAP. Thus, the SQL-MDi prologue complements the OLAP query by specifying the data structure over which to evaluate the query result. The syntax of SQL-MDi is inspired by the SQL standard [(ISO), 1992] to facilitate the analyst's work by providing a familiar notation.

Using the clauses provided by the SQL-MDi language, the OLAP analyst specifies how to generate a "virtual" global cube from two or more local cubes across autonomous Data Marts. In order to overcome possible heterogeneities among the local cubes, a rich set of conversion operators is available within the clauses of the SQL-MDi statement. The Dimension Algebra and Fact Algebra operators proposed in the previous Chapter 7 correspond exactly to the SQL-MDi conversions. Thus, the DA/FA represent a procedural implementation of these conversion clauses.

The basic structure of an SQL-MDi statement consists of the following three main clauses in the given, fixed order:

```
 DEFINE {[GLOBAL] CUBE   <cube-declarations>}
{MERGE DIMENSIONS        <merge-dim-subclauses>}
 MERGE CUBES             <merge-cube-subclauses>
```

Succeeding the SQL-MDi statement, the OLAP analyst formulates the analytic business question with a standard SQL query. The SQL query refers to the global cube schema, conforming to the following syntactic pattern [(ISO), 1992]:

```
SELECT     <dimension attributes>, <aggregated measures>
FROM       <fact tables>, <dimension tables>
WHERE      <selection criteria>
GROUP BY   <dimension attributes>
[HAVING    <group selection criteria>]
```

Each SQL-MDi query computes one result cube (or "output cube"—GLOBAL CUBE clause) from two or more corresponding source cubes (or "input cubes"—CUBE clauses). Notably, the output cube schema is not specified explicitly, but rather composed from the elements of the input cube schemas. The GLOBAL CUBE clause only declares the name of the output cube, referenced from the subsequent SQL query. Therefore, we denote the output cube as "virtual"—it is not necessarily materialized physically.

In general, SQL-MDi maps corresponding schema elements of both dimensions and cubes through *name equality* across all source cubes of the current statement. Consequently, the analyst's responsibility is to carefully unify the name strings of all corresponding schema elements to be merged, applying the appropriate operators within the DEFINE ... clauses and its various sub-clauses.

This approach avoids awkward pairwise mappings from input cubes to the output cube, facilitating the simultaneous integration of more than two input cubes. Notice that the standard database query language SQL [(ISO), 1992] employs exactly the opposite paradigm. In an SQL query, the user explicitly specifies the attributes of the output schema in the SELECT clause. Moreover, tables have to be "mapped" pairwise with "join" conditions in the WHERE clause of the query. Although this approach has its own merits, the specification of pairwise joins is appropriate for "flat" tables, but far too complex for the "hierarchical" dimension and cube structures.

Firstly, the DEFINE ... clauses specify both, the local source cubes and the name of the global, target cube. Every cube is assigned a mandatory *alias name*, used in the other clauses for referencing its properties (e.g., measure attributes). Each CUBE clause describes the *import schema* of a local cube by explicitly listing all dimension and fact attributes required in the global cube. SQL-MDi provides the import operators MEASURE and DIM as sub-clauses of CUBE for this purpose. All elements of local schemas not referenced with the import operators are ignored. The optional WHERE sub-clause allows to restrict the facts imported from the local cube to the specified selection predicates (i.e., slice and dice). Moreover, in order to repair domain and naming heterogeneities among local cubes, the conversion operators CONVERT MEASURES, ROLLUP and PIVOT as well as the renaming operators MAP LEVELS, '–>' (for attribute names) and '>>' (for attribute values) are available as sub-clauses of CUBE.

The GLOBAL CUBE clause declares only the name of the global, target cube, but none of its schema elements. Instead, the MERGE CUBES and MERGE DIMENSIONS clauses refer, respectively, to the fact and dimension attributes of the input cubes, and specify in detail their conversion to the facts and dimensions of the global cube. Thus, it is essential to carefully reconcile the number and names of all local dimension and measure attributes.

Example 8.1 (Basic concepts—DEFINE CUBE clause): The following SQL-MDi fragment imports all measure attributes and the two dimension attributes patient and drug of the medication cubes in the two Data Marts dwh1 and dwh2. Moreover, the statement specifies g::medication as the target cube:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, MEASURE c1.cost,
  DIM c1.patient, DIM c1.drug)
CUBE dwh2::medication AS c2
 (MEASURE c2.qty, c2.cost,
  DIM c2.patient, DIM c2.drug)
GLOBAL CUBE g::medication AS c0
...
```

Secondly, one MERGE DIMENSIONS clause is required for each dimension referenced by the input schemas. The MERGE DIMENSIONS clause specifies the renaming and conversion operators necessary to obtain homogeneous schemas and instances of the local dimensions. Moreover, the clause merges the local dimensions into one dimension of the global cube. All dimensions created this way are added to the schema of the global cube and referenced in the MERGE CUBES clause (explained in the next paragraph). In order to repair heterogeneities among the local dimensions at the schema or at the instance level, the conversion operators CONVERT ATTRIBUTES and RELATE <levels>, as well as the renaming operators RENAME (with '–>' for attribute names) and '>>' (for attribute values) are available as sub-clauses.

Example 8.2 (Basic concepts—MERGE DIMENSIONS clause): Assuming that the dimensions of the medication cubes were free of conflicts, the following SQL-MDi fragment would merge the patient and drug dimensions of the local cubes, computing two dimensions patient and drug of the target cube:

```
...
MERGE DIMENSIONS UNION c1.patient AS p1, c2.patient AS p2
     INTO c0.patient AS p0
MERGE DIMENSIONS UNION c1.drug AS d1, c2.drug AS d2 INTO
    c0.drug AS d0
...
```

Thirdly and finally, the MERGE CUBES clause completes the definition of the global, target cube. It specifies the global cube cells as "superset" of all cells of the local cubes defined in the current SQL-MDi statement. Importantly, the measure attributes of the local import schemas—specified by the DEFINE CUBE clauses—must match exactly, regarding both their number and their names. If necessary, the conversion operators PREFER, AGGREGATE MEASURE and TRACKING SOURCE AS DIMENSION are available as sub-clauses of MERGE CUBES to repair heterogeneity among local cube cells. Moreover, the ON sub-clause explicitly connects the global dimensions—generated by the MERGE DIMENSIONS clauses—with the global cube.

Example 8.3 (Basic concepts—MERGE CUBES clause): Assuming that the medication cube cells were free of conflicts, the following SQL-MDi fragment would merge the cells of the local cubes, computing the target cube cells and linking the cells to the dimensions patient and drug:

```
...
MERGE CUBES UNION c1, c2 INTO c0 ON patient, drug
```

Renaming operators of SQL-MDi

In order to map corresponding schema elements within SQL-MDi statements, the OLAP analyst must specify identical names in the import schemas of local data cubes. All naming conflicts are resolved by changing the conflicting name(s) of attributes, such that equivalent attributes receive identical names across the cube import schemas. Thus, renaming operations are frequently needed in SQL-MDi statements.

For that purpose, SQL-MDi provides two different rename operators, which emphasize the separation between properties at the schema and at the instance level syntactically. On the one hand, the rename operator "–>" changes *names of attributes*. It is generally used in the context of other keywords. In particular, the "–>" operator is available in conjunction with the MEASURE, DIM and MAP LEVELS keywords of the DEFINE CUBE clause as well as with the ATTRIBUTE keyword of the MERGE DIMENSIONS clause (cf. Example 8.4). On the other hand, rename operator '>>' is occasionally needed to change conflicting *values* *of attributes.* It is used together with the RENAME keyword of the MERGE DIMENSIONS clause, as Section 8.3 will explain (cf. Example 8.14).

The following Example 8.4 illustrates various usages of attribute renaming with operator "–>". Notice that Section 8.2 will detail, above others, the attribute rename operator and demonstrate several other occurrences. In contrast, the value rename operator '>>' will be explained later in Section 8.3, since it occurs less often.

Example 8.4 (Basic concepts continued—rename operators): The following SQL-MDi code snippet illustrates how to use the rename operator "–>" to change the names of measures (line 2), dimension attributes (line 3), level names (line 4), and non-dimensional attributes (line 9):

```
1 DEFINE CUBE dwh1::treatment AS c1
2  (MEASURE c1.cost_p -> costs, ...
3   DIM c1.date_time -> calendar_date ...
4    (MAP LEVELS date_time ([day -> day/time], [year]))
5  )
6 CUBE dwh2::treatment AS c2
7  ...
8 MERGE DIMENSIONS c1.calendar_date AS d1, c2.calendar_date
      AS d2 INTO c0.calendar_date AS d0
9  (ATTRIBUTE d2.descrpt -> description)
```

In the following Sections we illustrate how to repair the conflicts described in Chapter 5 using SQL-MDi. The Sections 8.1, 8.2 and 8.3 will explain the subclauses and conversion operators available in SQL-MDi and how to overcome schema-instance conflicts, heterogeneities at the schema level, and at the instance level, respectively, using these constructs.

8.1 Repairing Schema-Instance Heterogeneities

The following Section illustrates how to overcome schema-instance conflicts among facts and dimensions of autonomous data cubes. Schema-instance conflicts are the most complex heterogeneities, affecting both the schema level and instance level. As explained in Section 5.1 of the multi-dimensional conflict taxonomy (see pp. 62), these conflicts are caused by the use of different modelling elements to represent the same part of the real world. We explain the adequate use of the PIVOT sub-clause below DEFINE CUBE with example SQL-MDi query prologues, referring again to the treatment fact tables of the case study (see Figure 2.1, page 16 and Figure 2.4, page 19).

Typically, schema-instance conflicts affect the representation of *fact context* information in multi-dimensional models. Recall that the context of facts can be modelled either explicitly (as dimensions) or implicitly (within the facts themselves, e.g. as additional, specialized measures). The different representation of real-world data either within schema or instance modelling elements considerably impede the interpretation of multi-dimensional data by human analysts, so that they should always be repaired.

In order to remove schema-instance conflicts, two different approaches are available: either convert dimension members into specialized, "contextualized" measures or convert (part of) the fact context into members of a new dimension, called the "context dimension". SQL-MDi provides the PIVOT sub-clause below the DEFINE CUBE directive to perform these conversions. In order to support both approaches, two variants of PIVOT exist, that are distinguished by different keywords.

- **Dimension members into contextualized facts.**

  The first approach is characterized by a transformation of part of the "explicit" context (i.e., one of the dimensions), splitting up one measure variable into several new "contextualized" measure variables. This way, the dimension members become part of the fact schema. As a consequence of converting dimension members into the "domain" of new measure variables, the dimensionality of the cube decreases and the number of measures (the "expressivity") increases. Therefore, the higher expressivity of the fact schema allows to merge several cells of the old cube into a single cell of the new cube without any loss of information.

  The PIVOT sub-clause of DEFINE CUBE together with the SPLIT MEASURE keyword is used to specify this conversion. It conforms to the following syntax: PIVOT SPLIT MEASURE <cube-alias>"."<measure-attr-name> BASED ON <cube-alias>"."<dim-attr-name>. The operator splits up <measure-attr-name> to several new measures, converting the member identifiers of <dim-attr-name> into the names of the new measure variables, according to the name pattern "<measure-attr-name>_<dim-attr-value>". Finally, the <dim-attr-name> succeeding the BASED ON keyword is removed from the cube import schema.

  Example 8.5 (converting dimension members to measure variables): The following fragment of SQL-MDi code converts the costs measure of cube dwh2::treatment into two new costs measure variables, based on the members found in dimension dwh2::cost_cat:

```
1 DEFINE CUBE dwh1::treatment AS c1
2  (MEASURE c1.cost_p -> costs_personnel, MEASURE c1.
      cost_m -> costs_material,
3   DIM c1.method, DIM c1.date_time)
4 CUBE dwh2::treatment AS c2
5  (MEASURE c2.costs,
6   DIM c2.method, DIM c2.date_time, DIM c2.cost_cat,
7   PIVOT SPLIT MEASURE c2.costs BASED ON c2.cost_cat)
```

  As shown in Figure 2.4 on page 19, the cost_cat dimension in dwh2 contains two members, "personnel" and "material". Therefore, the PIVOT operator generates two new measure variables in the import schema of dwh2::treatment, named "costs_personnel" and "costs_material". Consequently, the measure attributes of cube c1 need to be renamed to match the new schema of cube c2 (line 2). Moreover, whenever the method and date_time values of two original cube cells match, these two cells will be merged into a single cell of the new cube. The cost value of an original cell linked to the "material" cost_cat-member will be written in the new cost_m measure; accordingly, the cost value of a cell associated with the "personnel" cost_cat-member is written in the new cost_p measure of the new cube. ◇

- **Fact context into dimension members.**

  In the second approach, several measure variables of the fact schema are converted into a single new measure. This way, parts of the fact schema—the names of the measure variables being transformed—become the members of a new dimension. The previously implicit context—hidden in the measure variables—is preserved and "externalized" within the new members. As a consequence of converting fact context into dimension members, the dimensionality of the cube increases and the number of measure variables (the "expressivity") decreases. Therefore, the lower expressivity of the fact schema requires the cells of the old cube to be split into several cells not to lose any information.

  The PIVOT sub-clause of DEFINE CUBE together with the MERGE MEASURES keyword is used to specify this conversion. It conforms to the following syntax: PIVOT MERGE MEASURES <measure-attr-list> INTO <new-measure-attr-name> USING <new-dim-attr-name>. The token <measure-attr-list> specifies the names of the measures to be converted in a comma-separated list, each name conforming to the pattern <cube-alias> "." <measure-attr-name>. The PIVOT operator generates the context dimension <new-dim-attr-name> and populates it automatically, generating the member values from the measure attribute names given in <measure-attr-list>.

Example 8.6 (converting measure variables to dimension members): The following snippet of SQL-MDi code converts the cost_m and cost_p measures of cube dwh1.treatment into a single, new measure variable named costs:

```
DEFINE CUBE dwh1::treatment AS c1
 (MEASURE c1.cost_p, MEASURE c1.cost_m,
  DIM c1.method, DIM c1.date_time,
  PIVOT MERGE MEASURES c1.cost_p, c1.cost_m INTO c1.
      costs USING c1.category
CUBE dwh2::treatment AS c2
 (MEASURE c2.costs,
  DIM c2.method, DIM c2.date_time, DIM c2.cost_cat ->
      category)
```

As the result of the above SQL-MDi code, the cost_m and cost_p measure attributes in the import schema of cube dwh1::treatment are replaced by the single, new measure variable costs. The PIVOT operator generates two members of the new dimension c1.category, named "cost_m" and "cost_p". Moreover, for every cell of the original cube two new cube cells are generated; all cost_m measures are written in a cell connected to the cost_m instance of the new context dimension c1.category. Analogously, all cost_p measures are written in cells connected to the cost_p instance of the new context dimension. ◇

8.2 Repairing Schema Level Heterogeneities

The following Section illustrates how to overcome heterogeneities at the schema level among facts and dimensions of autonomous data cubes. We illustrate the adequate use of the SQL-MDi sub-clauses below DEFINE CUBE and MERGE DIMENSIONS with example SQL-MDi statements, referring to the treatment fact tables of the case study (see Figure 2.1, page 16 and Figure 2.4, page 19).

8.2.1 Naming Conflicts

The following naming conflicts among the schemas of source cubes can be repaired using the rename operators of SQL-MDi in several sub-clauses of DEFINE CUBE and MERGE DIMENSIONS:

- **Naming conflicts in measure attributes.**
  The renaming of measures conforms to the following syntax: DEFINE CUBE "(" MEASURE <cube-alias> "." <measure-attr-name> ["–>" <new-measure-attr-name>] ")".
- **Naming conflicts in dimension attributes.**
  The renaming of dimension attributes conforms to the following syntax: DEFINE CUBE "(" DIM <cube-alias> "." <dim-attr-name> ["–>" <new-dim-attr-name>] ")".
- **Naming conflicts of levels within hierarchies.**
  The renaming of levels conforms to the following syntax: DEFINE CUBE ... "(" MAP LEVELS <dim-name> [<level-name> "–>" <new-level-name>] ")".
- **Naming conflicts in non-dimensional attributes.**
  The renaming of non-dimensional attributes conforms to the syntax: MERGE DIMENSIONS "(" ATTRIBUTE <dim-alias>"."<non-dim-attr-name> "–>" <new-non-dim-attr-name> ")".

  In contrast to the MEASURE and DIM keywords of the DEFINE CUBE clause, the ATTRIBUTE keyword is only specified for those non-dimensional attributes that should be renamed. The MERGE DIMENSIONS clause implicitly adds all other non-dimensional attributes of the source dimensions to the global dimension schemas. Eliminating non-dimensional attributes from cube import schemas is not supported because they often contain useful information on the dimension members. If a non-dimensional attribute is regarded unnecessary, recall that it can always be eliminated within the standard SQL query succeeding the SQL-MDi prologue.

The following example illustrates how to use the rename operator in the various clauses in order to remove the naming conflicts among the treatment data cubes of the case study.

Example 8.7 (Rename operators): Several elements of the treatment cube schemas are heterogeneous (see Figure 2.4, page 19): (1) the measure attributes, (2) the time dimensions (although the dimension attributes in the cube schemas have the same name), and (3) the non-dimensional attributes of the method dimension. The following SQL-MDi code fragment uses all of the aforementioned variants of the rename operator to repair these conflicts:

```
 1 DEFINE CUBE dwh1::treatment AS c1
 2  (MEASURE c1.cost_p -> costs,
 3   DIM c1.method, DIM c1.date_time -> calendar_date
 4    (MAP LEVELS date_time ([day -> day/time], [year]))
 5  )
 6 CUBE dwh2::treatment AS c2
 7  (MEASURE c2.costs, DIM c2.method, DIM c2.date_time ->
       calendar_date
 8    WHERE c2.cost_cat = 'personnel'
 9    (MAP LEVELS date_time ([day/hr -> day/time], [year]))
10  )
11 MERGE DIMENSIONS c1.calendar_date AS d1, c2.calendar_date
       AS d2 INTO c0.calendar_date AS d0
12  (ATTRIBUTE d2.descrpt -> description)
```

In the above SQL-MDi statement, we made the following assumptions: (1) the analyst wants to investigate only the personnel costs of all treatments. Therefore, he renames the cost_p attribute of dwh1. In order to accordingly restrict local cube dwh2 to cells with measures on personnel costs, a slice operation on the cost_cat dimension of dwh2 is necessary, using the WHERE sub-clause of DEFINE CUBE (line 8). (2) The level [day] of dwh1::date_time records a calendar date together with the time of day, so that the levels dwh1::date_time [day] and dwh2::date_time2 [day/hr] are mapped by using the rename operator in the MAP LEVELS clause. Consequently, the MERGE DIMENSIONS clause (sketched in line 11) would interpret the levels dwh1::date_time [day] and dwh2::date_time2 [day/hr] as semantically equivalent (as opposed to Example 8.8 on page 126, in which both [day] levels will be mapped as equivalent).

Moreover, the dimension attributes date_time were renamed to calendar_date in lines 3 and 7. Finally, the non-dimensional attribute descrpt in dimension dwh2::method is renamed to match the name of the description attribute in dimension dwh1::method. ◇

8.2.2 Diverse Aggregation Hierarchies

When importing local cubes, their dimension hierarchies have to correspond across all import schemas specified with the DEFINE CUBE clauses. Heterogeneities in aggregation hierarchies of partially corresponding dimensions are resolved by a "projection" on the common levels among all hierarchies (see the π operator of Dimension Algebra, Example 7.2 on page 91). This often means restricting the hierarchies among inner-level corresponding dimension schemas to only a few levels (cf. Examples 5.6 and 5.7, page 66). Base-level corresponding dimension schemas must even be reduced to a "degenerate" dimension—i.e., only the base level remains in the level schema.

For that purpose, the DEFINE CUBE clause provides the MAP LEVELS sub-clause, restricting the import schema of local dimensions to the subset of levels selected explicitly. It conforms to the following syntax: MAP LEVELS <dim-name> "(" <mapped-levels-list> ")". The token <dim-name> refers to dimension <node>::<dim-name> in the import schema of local cube <cube-alias>.

Since the <node> name is inherited automatically from the DEFINE CUBE clause, it is omitted in the MAP LEVELS sub-clause. In <mapped-levels-list>, the analyst gives a comma-separated list of level names (enclosed in square brackets), being a subset of all of <dim-name>'s levels. This way, the <mapped-levels-list> specifies which levels to include in the import schema of dimension <dim-name>.

Deleting levels from a dimension's import schema implicity changes its hierarchy. In order to maintain valid aggregation hierarchies, the MAP LEVELS operator transitively adjusts both the hierarchies and the roll-up functions in the import schema of <dim-name> as follows. Let D denote <dim-name>, $l, l' \in L_D$ be two levels of D in <mapped-levels-list> and $l_r \in L_D$ be some level of D not in <mapped-levels-list>. (1) The new hierarchy H'_D is computed from the old hierarchy H_D as follows: $(l \mapsto l_r) \in H_D \wedge (l_r \mapsto l') \in H_D \Rightarrow (l \mapsto l') \in H'_D$. (2) The new family of roll-up functions ρ'_D is adjusted accordingly, i.e. $\forall m \in members(l) : \rho_D^{l \mapsto l'}(m) := \rho_D^{l_r \mapsto l'}(\rho_D^{l \mapsto l_r}(m))$. In other words, the ROLLUP operator maintains all "finer than" respectively "coarser than" relationships $l \mapsto^{+} l'$ among the levels in <mapped-levels-list> of the original hierarchy H_D (see Definition 4.8, page 56).

The MAP LEVELS operator is an optional sub-clause of DEFINE CUBE. If a dimension d is imported using the DIM keyword (see Example 8.9), but no MAP LEVELS operator specified for d, the DEFINE CUBE clause implicitly includes all levels of d in the import schema.

Example 8.8 (Deleting unneeded levels in dimension hierarchies):
The date dimensions of the treatment data cubes are heterogeneous, since both their number of levels and hierarchies are different. The following code fragment illustrates how to apply the MAP LEVELS sub-clause on the date dimensions of the case study:

```
1 DEFINE CUBE dwh1::treatment AS c1
2  (MEASURE c1.cost_p, MEASURE c1.cost_m,
3    DIM c1.method, DIM c1.date_time
4    (MAP LEVELS date_time ([day], [year]))
5  )
6 CUBE dwh2::treatment AS c2
7  (MEASURE c2.costs, DIM c2.method, DIM c2.date_time
8    (MAP LEVELS date_time2 ([day], [year]))
9  )
```

The above query code imports the levels [day] and [year] of the date dimensions using the MAP LEVELS sub-clause (lines 4 and 8). Consequently, the levels [month] of dwh1::date_time as well as [day/hr], [week] of dwh2::date_time2 are eliminated from the import schemas.

Notice that "c2.date_time" in line 7 above refers to dimension attribute date_time of the local cube dwh2::treatment, whereas "date_time2" in line 8 denotes the dimension dwh2::date_time2. Moreover, this example ignores the additional domain conflict among both date dimension base levels. In order to overcome this problem, the analyst has to perform a roll-up of the dwh2::date_time2 dimension to level [day], as illustrated in Example 8.10 on page 128. ◇

8.2.3 Dimensionality Conflicts

Heterogeneous dimensionality among autonomous data cubes with partially corresponding cube schemas is resolved by restricting the global cube schema to the common, local dimensions. It is important to consider, though, that a reduction of the dimensionality entails the merging of cube cells, analogously to the *dice* OLAP operation. Therefore, any change in the dimensionality has far-reaching consequences on the fact instance, i.e. the cells of the cube. Note that the reduction of the dimensionality always leads to loss of information and may cause new overlapping cells conflicts among the autonomous DMs (see Section 5.3).

For the purpose of reducing the dimensionality of a cube, the DEFINE CUBE clause provides the DIM sub-clause for the import of dimensions from local data cubes. One DIM sub-clause is specified for every dimension of a local cube that should be available in the global cube. The DEFINE CUBE clause eliminates all measures and dimensions that are not explicitly mentioned in a MEASURE or DIM sub-clause. As such, the DEFINE CUBE operator performs a “projection” on the common schema elements.

Example 8.9 (Repairing dimensionality conflicts): The schemas of the treatment data cubes provide different sets of dimensions, as shown in Figure 2.1 (page 16). In order to use only the two common “overlapping” dimensions method and date_time in an OLAP query, the analyst would use the DIM sub-clause, as illustrated in the following code fragment:

```
DEFINE CUBE dwh1::treatment AS c1
 (MEASURE c1.cost_p, MEASURE c1.cost_m,
  DIM c1.method, DIM c1.date_time)
CUBE dwh2::treatment AS c2
 (MEASURE c2.costs,
  DIM c2.method, DIM c2.date_time)
```

The above SQL-MDi code eliminates the dimensions dwh1::phys and dwh2::cost_cat from the import schemas of the treatment cubes, because they are not imported by a DIM sub-clause. Consequently, the dimensionality of both data cubes' import schemas is reduced from three to two.

Notice that this example ignores the schema-instance conflict that persists among the costs measures of both treatment cubes. Since here the costs categories seem to be irrelevant for the OLAP query (dimension dwh2.cost_cat is eliminated), the analyst could simply calculate the sum of c1.cost_p and c1.cost_m in the MERGE CUBES clause to obtain a valid global cube. ◇

8.2.4 Domain Conflicts

All domain conflicts are resolved by applying a conversion function, such that equivalent attributes have identical domains in the import schemas. For that purpose, SQL-MDi provides the ROLLUP operator as sub-clause of MERGE DIMENSIONS as well as the operators CONVERT ATTRIBUTES and CONVERT MEASURES as sub-clauses of DEFINE CUBE. The following examples explain how to use these conversion operators.

Domain conflicts among base levels of hierarchies.

Heterogeneous domains of base levels among the hierarchies of inner-level corresponding dimension schemas are resolved by performing a roll-up to the lowest common level. For that purpose, SQL-MDi provides the ROLLUP sub-clause below DEFINE CUBE. It conforms to the following syntax: ROLLUP <cube-alias>"."<dim-attr-name> TO LEVEL <dim-name> "["<level-name>"]" WITH <aggr-function> FOR <cube-alias>.<measure-attr-name>. The token <cube-alias>.<dim-attr-name> refers to one of the dimensional attributes specified in the import schema of the local cube <cube-alias>, using the DIM keyword. The ROLLUP operator changes the domain of <dim-attr-name> to the target level <level-name> in dimension <dim-name>. The cube schema <cube-alias> has to be linked to <dim-name> with the attribute <dim-attr-name>. Additionally, <level-name> must be a part of <dim-name>'s hierarchy and specified in the MAP LEVELS operator of the cube import schema. Finally, the WITH keyword precedes the specification of an aggregation function <aggr-function> for each <measure-attr-name> of the local cube.

The semantics of ROLLUP is similar to the MAP LEVELS operator in that it performs a projection on dimension levels in the import schema of the local cube given in the same DEFINE CUBE clause. In particular, ROLLUP changes both the dimension hierarchy and the cells as follows: (1) All levels in the hierarchy from the base level up to the given "target level"—succeeding the TO LEVEL keyword in the ROLLUP clause—are deleted from the import schema of the dimension <dim-name>. Thus, the target level becomes the new base level in the dimension's import schema. (2) The domain of the dimension attribute <dim-attr-name> is changed from the original base level to the new base level <level-name> in dimension <dim-name>. (3) At the instance level, the cells of the data cube are recomputed, such that they link the measure values with the members of the new base level <level-name> instead of the members of the original base level. Consequently, several original cube cells will be merged into a single one, according to the roll-up hierarchy of the members in dimension <dim-name>. The measure values in the merged cells are computed using the specified aggregation functions. Thus, the SQL-MDi ROLLUP sub-clause reduces the detail level of the cube cells in the import schema—analogously to the OLAP roll-up operation.

Example 8.10 (Roll-up of a dimensional attribute): The base levels of both time dimensions in the treatment cubes are heterogeneous, as already explained in Example 8.8 on page 126. However, level [day]—the parent level of [day/hr]—in dimension dwh2::date_time2 is compatible to level [day] in dimension dwh1::date_time because their members are of the same granularity. Therefore, the simplest solution to solve the domain conflict among the base levels of the time dimensions is a roll-up of dimension dwh2::date_time2 from level [day/hr] to level [day]. The following code snippet illustrates how to perform this conversion using the ROLLUP sub-clause of SQL-MDi:

```
DEFINE CUBE dwh1::treatment AS c1
 (MEASURE c1.cost_p, MEASURE c1.cost_m,
  DIM c1.method, DIM c1.date_time)
CUBE dwh2::treatment AS c2
 (MEASURE c2.costs,
6    DIM c2.method, DIM c2.date_time)
7   (ROLLUP c2.date_time TO LEVEL date_time2[day] WITH SUM()
        FOR c2.costs)
```

The ROLLUP clause in the above query code deletes the level [day/hr] from the import schema of dimension dwh2::date_time2. Moreover, the roll-up operation is computed on the cells of cube c2, changing its detail level. The name of the dimensional attribute c2.date_time in the import schema remains the same, but its domain is changed from dwh2::date_time2[day/hr] to dwh2::date_time2[day]. Note that "c2.date_time" in line 7—succeeding the ROLLUP keyword—refers to the dimensional attribute in the fact schema of c2, whereas "date_time2" in the same line—next to the TO LEVEL keyword—denotes the dimension schema dwh2::date_time2. ◇

Domain conflicts among inner levels of hierarchies.

If domain conflicts occur among inner levels of the hierarchies of base-level corresponding dimensions, these conflicts cannot be resolved in SQL-MDi, only ignored using the MAP LEVELS sub-clause of the DEFINE CUBE clause. With the MAP LEVELS operator, the hierarchies are constrained to the levels with corresponding domains, deleting all other levels with incompatible domains from the import schema (*minimum use* integration strategy, see Figure 7.3 on page 105). To see an application of the MAP LEVELS sub-clause, albeit in the context of diverse hierarchies, refer to Example 8.8 on page 126.

SQL-MDi does not support domain conversion of inner level attributes since the roll-up functions to the parent level(s) could become invalid by the conversion of the lower level. Moreover, it is often impossible to find and implement a generalized conversion function for repairing these domain conflicts, the reason being that dimension members usually model very specialized knowledge or narrow concepts of the real-world. For example, consider members of a "product" dimension on the "product group" level. Hierarchies of products and product groups are very specific to one particular organization and the domains of independent product groups can hardly be harmonized among diverse domains.

Domain conflicts among non-dimensional attributes.

Domain conflicts may also occur among dimension schemas with overlapping or identical non-dimensional attributes. Analogously to measure attributes, heterogeneous domains among non-dimensional attributes are resolved by applying a conversion function on the cube import schema. SQL-MDi provides the CONVERT ATTRIBUTES sub-clause of the MERGE DIMENSIONS directive for that purpose. It conforms to the following syntax: CONVERT ATTRIBUTES APPLY <function-name> FOR <dim-alias> "." <dim-attr-name> (DEFAULT | WHERE <conditions>). The operator applies function <function-name> on the values of non-dimensional attribute <dim-attr-name> in dimension <dim-alias>. The WHERE keyword optionally restricts the application of the conversion function to the <dim-attr-name>-values in the subset of members matching the selection criteria given in <conditions>. Alternatively, the DEFAULT keyword performs the conversion function for the <dim-attr-name>-values of all members.

Example 8.11 (Converting the domain of non-dimensional attributes):
Assume that the treatment data cubes use different currencies for recording the costs figures. Say, dwh1::treatment records costs in Euros, whereas dwh2::treatment contains costs in US-$. In this case, the analyst would have to reconcile the domains of the non-dimensional attribute hourly_costs of the [method] level in the method dimensions. For this scenario, the CONVERT ATTRIBUTES sub-clause provided by SQL-MDi is illustrated in the following code snippet:

```
DEFINE CUBE dwh1::treatment AS c1
 (MEASURE c1.cost_p, MEASURE c1.cost_m,
  DIM c1.method, DIM c1.date_time)
CUBE dwh2::treatment AS c2
 (MEASURE c2.costs,
  DIM c2.method, DIM c2.date_time)
GLOBAL CUBE g::treatment AS c0
MERGE DIMENSIONS c1.method AS d1, c2.method AS d2 INTO c0
    .method AS d0
 (CONVERT ATTRIBUTES APPLY usd2Eur() FOR d1.hourly_costs
     DEFAULT)
```

Notice that function usd2Eur() used in the above SQL-MDi statement is independently defined, and must be available within the global ROLAP system.◇

Domain conflicts among measure attributes

Among the cells of autonomous cubes, domain conflicts may occur if the measure attributes in the cube schemas overlap or are identical. Heterogeneous domains among measure attributes are resolved by applying a conversion function on the cube import schema. SQL-MDi provides the CONVERT MEASURES sub-clause below the DEFINE CUBE directive for that purpose. It conforms to the following syntax: CONVERT MEASURES APPLY <function-name> FOR <cube-alias> "." <measure-name> (DEFAULT | WHERE <conditions>). The operator applies function <function-name> on the values of attribute <measure-name> in local cube <cube-alias>. The WHERE keyword allows to restrict the application of the conversion function to the subset of members matching the selection criteria specified in <conditions>. In contrast, the DEFAULT keyword performs the conversion function for all members.

Example 8.12 (Converting the domain of measure attributes):
Again, let us assume that the treatment data cubes use different currencies for recording the costs figures. For example, dwh1::treatment may record costs in Euros, whereas dwh2::treatment contains costs in US-$. In that case, the analyst would have to reconcile not only the domains of the non-dimensional attribute hourly_costs of dimension level [method], but also the domains of the costs attributes in order to obtain a semantically meaningful global data cube. The use of the CONVERT MEASURES sub-clause for this purpose is illustrated in the following code snippet:

```
DEFINE CUBE dwh1::treatment AS c1
 (MEASURE c1.cost_p, MEASURE c1.cost_m,
  DIM c1.method, DIM c1.date_time)
  (CONVERT MEASURES APPLY usd2Eur() FOR c1.cost_p DEFAULT,
       usd2Eur() FOR c1.cost_m DEFAULT)
CUBE dwh2::treatment AS c2
 (MEASURE c2.costs,
  DIM c2.method, DIM c2.date_time)
```

Notice that function usd2Eur() used in the above SQL-MDi statement is defined independently for the domain of numerical attributes. Furthermore, this example ignores the schema-instance conflict that persists among the costs measures of both treatment cubes. In order to overcome this conflict, the analyst could simply calculate the sum of c1.cost_p and c1.cost_m in the MERGE CUBES clause if the costs categories are not relevant. ◇

8.3 Repairing Instance Level Heterogeneities

Using the medication fact table of the case study (see Figure 2.1 on page 16 and Figure 2.3 on page 18), the following Section illustrates how to overcome heterogeneities at the instance level among facts and dimensions of autonomous data cubes. Various example query prologues will illustrate the adequate use of the SQL-MDi operators in the sub-clauses below MERGE DIMENSIONS and MERGE CUBES.

8.3.1 Heterogeneous Roll-up Functions

Overlapping dimension members may cause a conflict if their roll-up functions deliver different parent members. In order to repair heterogeneous roll-up functions, SQL-MDi provides the RELATE operator as sub-clause of the MERGE DIMENSIONS directive. It conforms to the following syntax: RELATE <levels-list> <join-conditions> USING HIERARCHY OF <dim-alias>. The levels given in <levels-list> must be corresponding, that means it is necessary to specify a MAP LEVELS operator for these levels (see Section 8.2, Example 8.8 on page 126). The RELATE operator overwrites the roll-up functions of the corresponding levels <levels-list> to their parent level with the roll-up function used in dimension <dim-alias>, considered to be the most trustworthy source dimension. The expressions in <join-condition> restrict the RELATE operator to only those members matching the specified selection criteria. Thus, the RELATE operator allows a fine-grained removal of heterogeneous roll-up functions.

Example 8.13 (Repairing heterogeneous roll-up functions):
Dimension drug_dim in DMs dwh1 and dwh2 illustrates a typical example of conflicting roll-up functions. Whereas member 'B' rolls-up to member 'Novartis' on level manufacturer in dimension dhw1::drug, the same member 'B' rolls-up to member 'Bayer' in dimension dhw2::drug. Obviously, only one of these roll-up mappings can be true in the real world. In order to repair the conflict, the analyst could estimate that dwh1 is the more trustworthy source DM and, thus, to use the roll-up functions specified in dwh1::drug. Consequently, all other conflicting roll-up functions in the semantically equivalent drug levels of the source dimensions would be overwritten. The use of the SQL-MDi operator RELATE is illustrated in the following code snippet:

```
DEFINE CUBE dwh1::medication AS c1 ...
CUBE dwh2::medication AS c2 ...
GLOBAL CUBE gdw::medication AS c0
MERGE DIMENSIONS c1.drug AS d1, c2.drug AS d2 INTO c0.
    drug AS d0
 (RELATE d1.manufacturer, d2.manufacturer WHERE d1.drug =
      d2.drug USING HIERARCHY OF d1)
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 ...
```

When applied to the dimension tables in Figure 2.3 (page 18), the above SQL-MDi code overwrites roll-up mapping *'B'* ↦ *'Bayer'* (from dwh2::drug) with *'B'* ↦ *'Novartis'* (from dwh1::drug). ◇

8.3.2 Value Conflicts among Non-dimensional Attributes

Overlapping members may cause non-dimensional value conflicts. Conflicting values in non-dimensional attributes are repaired in order to give concise descriptions of dimension members. The contradictory values are harmonized using the SQL-MDi conversion operator RENAME, a sub-clause of the MERGE DIMENSIONS directive. It conforms to the following syntax: RENAME <dim.alias> "." <dim-attr-name> (<old-value> ">>" <new-value> | USING MAPPINGTABLE <table-name>). The RENAME operator replaces all occurrences of <old-value> in attribute <dim-attr-name> with <new-value> within the members of <dim-alias>. Alternatively, a mapping table can be used containing all necessary <old-value>/<new-value> pairs. The schema of such a mapping table has to define exactly two attributes, named old_val and new_val.

Example 8.14 (Renaming heterogeneous N-attribute values): As shown in Figure 2.3 on page 18, the non-dimensional attribute pkg_size is inconsistent among the members of the drug dimension in both medication data cubes. Assuming that a data entry error happened in dwh2, the analyst could repair the false values of the non-dimensional attribute pkg_size for all members of dimension dwh2.drug using the following SQL-MDi code:

```
DEFINE CUBE dwh1::medication AS c1 ...
CUBE dwh2::medication AS c2 ...
GLOBAL CUBE gdw::medication AS c0
MERGE DIMENSIONS c1.drug AS d1, c2.drug AS d2 INTO c0.
    drug AS d0
 (RENAME d2.pkg_size >> '25 pcs.' WHERE d2.drug = 'A')
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 ...
```

In practice, value conflicts are common, and typically happen often among autonomous data sources [Doan and Halevy, 2005]. When occurring more than occasionally, value conflicts are too awkward to repair with the manual '>>' rename operator. In such cases, the RENAME clause of SQL-MDi provides the alternative, more user-friendly *mapping table* method, as the following example illustrates.

Example 8.15 (Mapping table for heterogeneous N-attribute values): Analogously to the above Example 8.14, assume that non-dimensional attribute pkg_size of the drug dimension contains data entry errors in dwh2. Using the mapping table option of the RENAME clause, the analyst could repair the false pkg_size-values for all members of dimension dwh2.drug more elegantly than in the previous example with the following SQL-MDi statement:

```
DEFINE CUBE dwh1::medication AS c1 ...
CUBE dwh2::medication AS c2 ...
GLOBAL CUBE gdw::medication AS c0
MERGE DIMENSIONS c1.drug AS d1, c2.drug AS d2 INTO c0.
    drug AS d0
 (RENAME d2.pkg_size USING MAPPINGTABLE gdw::map_pkgsize)
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 ...
```

Whenever the current value of d2.pkg_size occurs in the old_val column of mapping table gdw::map_pkgsize, it is replaced by the appropriate new_val-value as specified in the mapping table. ◇

8.3.3 Overlapping Sets of Dimension Members

If member sets overlap, the analyst has to decide both how to handle the conflicting members and how to merge the non-overlapping members. It is crucial to recognize *what* real-word object (e.g., a product) or concept (e.g., a date of year) is modelled by the members. In most cases, overlapping member subsets are identical-related, since a dimension represents non-summarizable objects or concepts of the real world (e.g., geographical locations, products, etc.—refer also to the discussion in Section 5.3, pp. 71).

Therefore, since the members describe the same entities of the real world, only one of the existing and heterogeneous instances can be true. The analyst has to elect one of the data sources as the most trustworthy. In order to resolve the overlapping members, the conflicting instances will be overwritten with the values stored in the trusted source.

Regarding the disjoint subsets of overlapping member sets, it is important to decide which of the local members to include in the dimensions of the global data cube. In most cases, a simple union of all dimension members will be the adequate solution. Therefore, the semantics of the MERGE DIMENSIONS clause is the union set operation by default. Moreover, the standard set operations UNION, MINUS, [(LEFT | RIGHT | FULL) OUTER] JOIN are available like in the SQL standard [(ISO), 1992]. The set operation desired by the analyst is specified as optional keyword in the MERGE DIMENSIONS clause.

In order to overwrite overlapping members with the instances stored in the trusted source, SQL-MDi provides the PREFERRING sub-clause of the MERGE DIMENSIONS directive. It conforms to the following syntax: MERGE DIMENSIONS ... "(" PREFERRING <cube-alias> (DEFAULT | WHERE <conditions>) ")". For the subset of members that overlap, the PREFERRING operator will set the value of the dimension attribute given in the MERGE DIMENSIONS clause to the value of this dimension attribute in <cube-alias>. The expressions in <join-condition> restrict the PREFERRING operator to only those members matching the specified selection criteria. The semantics of the PREFERRING operator combines the RELATE and RENAME operators (see the previous examples), applied on all overlapping members. Thus, the PREFERRING operator allows the set-oriented resolution of both heterogeneous roll-up functions and non-dimensional values conflicts among overlapping members.

Example 8.16 (Handling overlapping members): The members in all dimensions of the global data cube g::medication should simply be computed as the union of the local dimension members in dwh1 and dwh2. For the subset of overlapping members, c1 is regarded as more trustworthy source and should be preferred. The UNION set operation and the PREFERRING operator within the MERGE DIMENSIONS directive of SQL-MDi are illustrated in the following code fragment:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, MEASURE c1.cost,
  DIM c1.patient, DIM c1.drug, DIM c1.date_time)
CUBE dwh2::medication AS c2
 (MEASURE c2.qty, c2.cost,
  DIM c2.patient, DIM c2.drug, DIM c2.date_time)
GLOBAL CUBE gdw::medication AS c0
MERGE DIMENSIONS UNION c1.drug AS d1, c2.drug AS d2 INTO
    c0.drug AS d0
 (PREFERRING c1 DEFAULT)
...
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 ...
```

Example 8.17 (Applying the MINUS operation on overlapping members): In some cases, however, the semantics of cross-DM queries require to apply a different set operation among the member sets. For example, a global cube retrieving the costs caused by drugs that are used in Vienna (dwh1), but not in Salzburg (dwh2), would be specified using a MINUS set operation. The necessary SQL-MDi code is illustrated below:

```
DEFINE CUBE dwh1::medication AS c1 ...
CUBE dwh2::medication AS c2 ...
GLOBAL CUBE gdw::medication AS c0
MERGE DIMENSIONS MINUS c1.drug AS d1, c2.drug AS d2 INTO
    c0.drug AS d0
 ...
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 ...
```

The above code creates a dimension c0.drug with only a single member ('C', 'Merck', 'ALL'). Note that drug 'C' is the only dimension member of dwh1.drug that is not a member of dwh2.drug (see Figure 2.3, page 18). The import schemas of c1 and c2 (i.e. the DEFINE CUBE clauses) and the MERGE CUBES clause match those of the previous Example 8.16. ◇

8.3.4 Disjoint Sets of Dimension Members

If member sets are disjoint, the only sensible option is computing the union of all the local dimension members. Any set operation other than UNION would compute an empty member set in the global dimension. Thus, a standard MERGE DIMENSIONS clause without any sub-clause is sufficient for integrating disjoint member sets.

8.3.5 Overlapping Cells

Overlapping subsets of cube cells express either an identical-related or context-related semantics (see Section 5.3, pp. 71). When identical-related overlapping cells occur, the measures of these cells model conflicting information on one and the same real-word entity. Consequently, either one of the facts' measure values has to be *preferred*, overwriting the values of the overlapping cells, or the origin of the fact data is an essential part of its context information. In the latter case, the origin is regarded as an additional, implicit dimension—such that the measures become *source-identified* by means of a newly generated context dimension.

The challenge with context-related overlapping cells is to correctly recognize *what* entity or entities they describe. Depending on the semantic context among the overlapping cells, the measures might be summarizable or not. On the one hand, if the overlapping cell subsets represent measures of different entities or concepts of the real world, aggregating the measure values and merging the cells might give witless results (the measures are "non-summarizable"). In this case, the analyst will extend the cells with a *context dimension* preserving the previous semantics of the isolated DMs—analogously to extending identical-related overlapping cells with the context dimension. On the other hand, if the cells refer to the same real-world entity or concept, summarizing the measure values using an aggregation function like SUM or AVG (thus merging the overlapping cells) is the appropriate solution.

In the following, we will describe all three approaches: (1) preferring sources, (2) generating the context dimension, and (3) aggregating overlapping cells.

- *Preferring sources:* in the case of identical-related overlapping cube instances, describing same real-world entities in the same context, the data designer usually will elect one of the data cubes in the federation as the most trustworthy source and prefer the measure values stored in this source. SQL-MDi provides the PREFER sub-clause below the MERGE CUBES directive for that purpose. It conforms to the following syntax: PREFER <cube-alias> "." <measure-name> (DEFAULT | WHERE <conditions>). If a subset of local cube cells overlap, the PREFER operator will set the value of <measure-name> in the global cube to the value of <measure-name> in the local cube <cube-alias> for all cells in the overlapping subset. The WHERE keyword allows to specify selection predicates on measure and/or dimension attributes of <cube-alias>, restricting the application of the PREFER operator on a subset of the <measure-name>-values. If the DEFAULT keyword is given, the PREFER operator will be evaluated on all overlapping cell subsets.

Example 8.18 (Preferring particular sources of overlapping facts): Assuming that the overlapping facts depicted in Figure 2.3 on page 18 give conflicting descriptions of one and the same medical remedy (*identical*-relationship), the analyst would have to decide which of both DMs dwh1 and dwh2 is the more trustworthy source of data. When preferring the measures of, say, dwh1, the analyst would use the PREFER sub-clause of MERGE CUBES as illustrated in the following code snippet:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, MEASURE c1.cost,
  DIM c1.patient, DIM c1.drug, DIM c1.date_time)
CUBE dwh2::medication AS c2
 (MEASURE c2.qty, c2.cost,
  DIM c2.patient, DIM c2.drug, DIM c2.date_time)
GLOBAL CUBE g::medication AS c0
MERGE DIMENSIONS  ...
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 (PREFER c1.qty DEFAULT,
  PREFER c1.cost DEFAULT)
```

Whenever some cells of the local cubes c1 and c2 overlap, the values of measures c0.qty and c0.cost in the global cube will be equal to the values of c1.qty and c1.cost—the more trusted source c1 "overwrites" the measure values of cube c2. ◇

- *Extending overlapping facts with the context dimension:* For both, non-summarizable context-related cells and identical-related cells of overlapping cube instances, SQL-MDi provides the TRACKING operator as sub-clause below the MERGE CUBES directive. It conforms to the following syntax: TRACKING SOURCE AS DIMENSION <dim-name> "("<schema-spec>")" (IS <value> WHERE <source-condition> | DEFAULT). The token <dim-name> specifies a new dimension attribute—the "context dimension"—that will be added to the global cube schema. The context dimension is always *degenerated*, i.e. it consists of only a single level, also named <dim-name>. Level <dim-name> is composed of only one dimensional attribute, the data type of which is specified in <schema-spec>. Finally, the member set is the union of all <value>s chosen by the human analyst. Each member is associated with the cell subset satisfying the selection predicates given in the WHERE keyword. In contrast, the DEFAULT member is linked to all cells that do not match any of the selection predicates given in the WHERE clauses.

Example 8.19 (Context dimension extending overlapping facts): Assuming that both sources dwh1 and dwh2 contain data of the health insurance's sub-organizations in different federal states, the analyst could want to compare the costs across the states. In this case, the original source of local cells is an important piece of context information that has to be preserved. In order to generate the context dimension, the analyst would use the following SQL-MDi code with the TRACKING operator:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, MEASURE c1.cost,
  DIM c1.patient, DIM c1.drug, DIM c1.date_time)
```

```
CUBE dwh2::medication AS c2
 (MEASURE c2.qty, c2.cost,
  DIM c2.patient, DIM c2.drug, DIM c2.date_time)
GLOBAL CUBE g::medication AS c0
MERGE DIMENSIONS ...
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 (TRACKING SOURCE AS DIMENSION source(VARCHAR(16))
     IS 'salzburg' WHERE SOURCE()='c2', IS 'vienna'
         DEFAULT)
```

In the above code we assume that dwh1 and dwh2 be located in cities Vienna and Salzburg, respectively. Note that when sending a query individually to one of the autonomous cubes dwh1::medication or dwh2::medication, it is clear for the analyst that the quantity and costs figures refer to the cities Vienna resp. Salzburg. Using the TRACKING operator, this semantics is preserved in the context dimension of the global data cube. ◇

- *Aggregating overlapping facts:* if context-related overlapping cells describe "different" entities in similar contexts, the measure values of these cell subsets can be aggregated. In that case, the aggregated measure values in the global cube are the important piece of information. In SQL-MDi, the AGGREGATE MEASURE sub-clause of MERGE CUBES allows the analyst to specify the desired aggregation function. It conforms to the following syntax: AGGREGATE MEASURE <measure-attr-name> IS <aggregation-function> OF <local-measure-attr-name> [WHERE <conditions>]. In order to compute <measure-attr-name> of the global cube, function <aggregation-function> is applied on the values of all local cubes' <local-measure-attr-name>. The OF clause is needed to compute derived measures.

  Example 8.20 (Aggregating overlapping facts): If the business analyst wants to know the overall costs across the sub-organizations, the original source of data as part of the context information is not important. In contrast, the overlapping cells can simply be aggregated to compute the global cube cells. Using the AGGREGATE MEASURE sub-clause, the overlapping cell subsets can be merged using SQL-MDi as in the following statement:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, MEASURE c1.cost,
  DIM c1.patient, DIM c1.drug, DIM c1.date_time)
CUBE dwh2::medication AS c2
 (MEASURE c2.qty, c2.cost,
  DIM c2.patient, DIM c2.drug, DIM c2.date_time)
GLOBAL CUBE g::medication AS c0
MERGE DIMENSIONS ...
MERGE CUBES c1,c2 INTO c0 ON patient,drug,date_time
 (AGGREGATE MEASURE qty IS SUM OF qty,
  AGGREGATE MEASURE cost IS SUM OF cost)
```

  Instead of SUM, other standard aggregation functions—such as AVG, MIN, MAX, etc.—can be applied to compute the measure values in the global cube from the overlapping cells. ◇

8.3.6 Disjoint Cells

If the cells of cube instances are disjoint, the only sensible option is computing the union of all the local cube cells. Any set operation other than union would compute an empty global cube. Thus, a standard MERGE CUBES clause without any operators in the sub-clauses is sufficient for integrating the disjoint cell sets.

Example 8.21 (Merging disjoint facts): The following SQL-MDi statement uses disjoint measures of the dimension-homogeneous cubes dwh1::medication and dwh2::medication to compare the quantity of given medications in dwh1 with the medication costs recorded in dwh2 for all patients:

```
DEFINE CUBE dwh1::medication AS c1
 (MEASURE c1.qty, DIM c1.patient)
CUBE dwh2::medication AS c2
 (MEASURE c2.cost, DIM c2.patient)
GLOBAL CUBE g::medication AS c0
MERGE DIMENSIONS  ...
MERGE CUBES c1,c2 INTO c0 ON patient
```

The result of the above statement will be the global cube g::medication with two measures, qty and cost. Notably, the measures c0.qty and c0.cost will contain *null* values for every cell of either source cube that does not overlap with a corresponding cell of the other source cube.

8.4 Summary of SQL-MDi

The SQL-MDi language uses three main clauses for defining the global schema and semantic matches: (1) Define [Global] Cube, (2) Merge Dimensions, and (3) Merge Cubes. The Cube clauses specify the attribute structure of both the virtual global cube and the import schemas of the autonomous Data Marts. For each dimension of the global schema the Merge Dimensions clause defines its members and hierarchy from the imported dimensions. Analogously, the Merge Cubes clause determines how to compute the cells of the global cube from the imported facts.

Within each of its main clauses SQL-MDi defines a number of operators to repair heterogeneous schemas and instances of facts and dimensions (cf. Tables 9.3 and 9.4). While the CUBE clause allows to repair heterogeneities among the Data Mart import *schemas*, the MERGE clauses mainly provide operators for the definition of *tuple* matchings and translations in dimensions and facts.

Semantic mappings between elements of autonomous Data Marts are generally expressed by equal naming. Thus, renaming operations often occur in SQL-MDi, as illustrated by the example statements of this chapter. It is important to check the import schemas carefully for name equality between corresponding schema elements in order to produce the desired results.

To conclude this chapter, the following Example 8.22 summarizes all previous SQL-MDi statement fragments for the treatment cubes presented earlier:

Example 8.22 (Complete SQL-MDi statement): The following SQL-MDi statement contains the "superset" of all conversions used for the two treatment cubes during the previous examples of this chapter. It computes global cube g::treatment (as given in Figure 2.5, page 21) from the local treatment cubes of Data Marts dwh1 and dwh2.

```
DEFINE CUBE dwh1::treatment AS c1
  (MEASURE c1.cost_p -> costs_personnel,
   MEASURE c1.cost_m -> costs_material,
   DIM c1.method, DIM c1.date -> calendar_date
   (MAP LEVELS dwh1::date_time ([ day ], [ year ]) )
)
CUBE dwh2::treatment AS c2
  (MEASURE c2.cost-$,
   DIM c2.method, DIM c2.date_time -> calendar_date
   (MAP LEVELS dwh2::date_time2 ([ day ], [ year ]) ),
   DIM c2.cost_cat,
  PIVOT SPLIT MEASURE c2.cost-$ BASED ON c2.cost_cat )
  (ROLLUP c2.calendar_date TO LEVEL [day] WITH SUM() FOR
     c2.costs)
  (CONVERT MEASURES APPLY usd2eur() FOR c2.cost-$ DEFAULT
     )
GLOBAL CUBE g::treatment AS c0

MERGE DIMENSIONS c1.method AS md1, c2.method as md2 INTO
    c0.method AS md0

MERGE DIMENSIONS c1.calendar_date AS cd1, c2.calendar
    date AS cd2 INTO c0.calendar_date AS cd0
  (ATTRIBUTE cd2.descrpt -> description )

MERGE CUBES c1, c2 INTO c0 ON method, calendar_date
  AGGREGATE MEASURE costs_personnel IS SUM OF
     costs_personnel,
  AGGREGATE MEASURE costs_material IS SUM OF
     costs_material
```

Part IV

Realizing the Federated Data Warehouse

Chapter 9

Prototype Implementation of FedDW Tools

Contents

To demonstrate the practical viability of the FedDW approach, the FedDW tool suite was developed, comprising prototypes of two tools that implement the concepts introduced by this thesis. This chapter presents the implementation architecture and concepts behind the FedDW tool suite. On the one hand, FedDW *Global Schema Architect (GSA)* is a visual, model-driven design environment for the integration of autonomous Data Marts. GSA conforms to the Federated DW reference architecture introduced in Chapter 6, and implements the integration methodology proposed in Chapter 7. On the other hand, FedDW *Query Tool* allows to execute queries over global, federated Data Marts designed with GSA. Query Tool combines SQL-MDi statements with SQL OLAP queries.

Despite clear benefits from the business perspective, the integration of autonomous Data Marts is difficult and laborious for both, technical and organizational reasons. Technically, numerous heterogeneities among the schemas and data need to be resolved, as Chapter 7 of this thesis discussed in depth. From the organizational viewpoint, DW access is often restricted to ensure the privacy of confidential data. Thus, the complete physical combination of autonomous DWs—which would be the easiest solution—is often impractical, especially for large-scaled systems. Such problems commonly occur in practice, sometimes even among the divisions of a single company [Kimball, 2002].

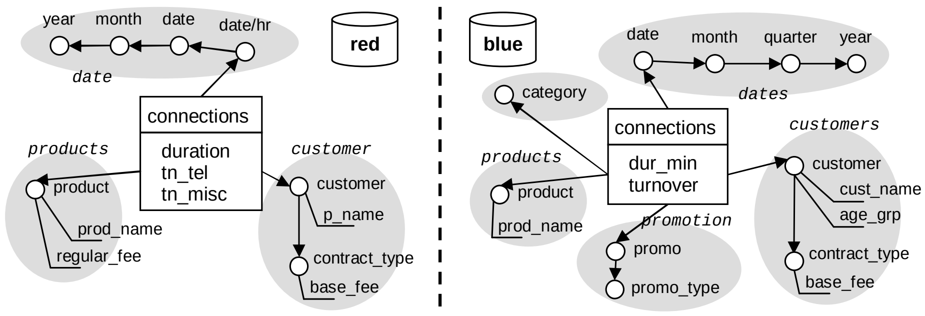


Figure 9.1: Conceptual Data Mart schemas of companies Red and Blue in Dimensional Fact Model notation [Golfarelli et al., 1998].

For example, assume that two competing mobile network providers—we call them "Red" and "Blue"—agree upon sharing their DW data for strategic decision making. This cooperation is advantageous for both partners since the knowledge base for their strategic business decisions broadens beyond the original organizational boundaries. Questions such as "Which service plans are most popular among 21–30 year old customers?" become easier to analyze for the two providers if more data is available for the OLAP applications.

Numerous heterogeneities exist among the Data Marts of Red and Blue, although representing the same domain (see Figure 9.1). For instance, cube schema "connections" of Red has modelled two measures, tn_tel and tn_misc for turnovers generated with telephony and other services, respectively, whereas Blue has chosen a single measure turnover plus the category dimension (*schema–instance* conflict). Besides, Blue's Data Mart records the promotion dimension with turnovers, which cannot be matched in Red's schema (*dimensionality* conflict). Moreover, the hierarchies and levels in the date dimensions, as well as the non-dimensional attributes within the customer and products dimensions are heterogeneous. Obviously the two cubes can only be unified after previous conversion of their schemas and their data (illustrated in Figures 9.2 and 9.3).

Notice that the conflicts contained in the "Red and Blue" example are very similar to the health care running example (Chapter 2), but in more condensed form. To present the prototype implementation of the FedDW tool suite, we combined almost all conflicts analyzed in Chapter 5 into only one pair of example cubes. Test results are easier to verify manually this way. In contrast, the medication and treatment cubes of the case study illustrated classes of heterogeneity at the schema and at the instance level separately, which allowed for an easier presentation of the conflicts and solutions throughout the thesis.

red::connections (date_hr: date [date_hr], cust_sscode: customer [cust_sscode], product: products [product]; duration, tn_tel, tn_misc)

date_hr	cust_sscode	product	duration	tn_tel	tn_misc
01.01.08 08:00	1234010180	MobCom	58	11.60	1.00
01.01.08 08:00	1234010180	HandyTel	20	5.50	2.22
03.01.08 08:00	4567040871	FiveCom	250	24.10	4.50
30.06.08 08:00	9876543210	HandyTel	130	14.20	4.00
30.06.08 10:30	9876543210	FiveCom	28	7.10	0.80

red::date (date_hr $\mapsto$ date $\mapsto$ month $\mapsto$ year)

date_hr	date	month	year
01.01.08 08:00	01.01.08	01/08	2008
03.01.08 08:00	03.01.08	01/08	2008
30.06.08 08:00	30.06.08	06/08	2008
30.06.08 10:30	30.06.08	06/08	2008

red::products (product)

product	prod_name	regular_fee
MobCom	Mobile Comm., Ltd.	0.10
HandyTel	Handy Telephony Co.	0.08
FiveCom	Five Communications, Inc.	0.12

red::customer (cust_sscode $\mapsto$ contract_type)

cust_sscode	name	contract_type	base_fee
1234010180	Doe, John	SparCom	15.0
4567040871	Stone, Jack	SparCom	15.0
9876543210	Miller, Alice	FlatRate	45.0

Figure 9.2: Fact and dimension tables of company "Red".

Schema integration among autonomous Data Marts addresses two major aspects: dimension integration and fact integration (cf. Section 3.5, pp. 39). The *dimension integration* problem is more complex than traditional, relational schema integration because dimension attributes are structured in hierarchies of aggregation levels. In contrast, *fact integration* corresponds to the traditional challenges of schema matching and data matching among relational entities [Doan and Halevy, 2005]. However, the interdependencies among dimensions and facts complicate the fact integration problem as well, as Chapter 3 has explained in depth. In particular, fact schemas reference dimension attributes as foreign keys to identify "coordinates" of cells in cubes.

Current design tools and integration frameworks for Federated DWs handle dimension integration and fact integration as isolated aspects, as discussed in Chapter 3. Hence, integration of autonomous Data Marts with the available DW approaches falls short. Instead, federated database technology—e.g., multi-database languages such as FISQL [Wyss and Robertson, 2005] or Schema-SQL [Lakshmanan et al., 2001]—is often used as compromise to physically integrate the multi-dimensional data. The Red and Blue example, however, illustrates that conflicts among multi-dimensional Data Marts are too complex for current technology, such as reported by [Kimball, 2002]. In particular, the hierarchies in dimensions are problematic, as studied in [Berger and Schrefl, 2006].

blue::connections (date: dates [date], customer: customers [customer_id], product: products [product], promotion: promotions [promo], category; dur_min, turnover)

date	customer	product	promotion	category	dur_min	turnover
01.01.08	1234010180	MobCom	noPromo	tn_tel	17	2.55
01.01.08	1234010180	MobCom	Fall 2008	tn_tel	23	8.50
01.01.08	1234010180	MobCom	Winter 2007	tn_tel	9	4.50
30.06.08	9876090875	HandyTelCo	noPromo	tn_tel	15	1.20
25.08.08	6785041070	HandyTelCo	Christmas 2008	tn_tel	50	4.77
25.08.08	6785041070	HandyTelCo	Christmas 2008	tn_misc	0	2.70

blue::products (product)

product	prod_name
MobCom	Mobile Comm.
HandyTelCo	Handy Telephony
MobileCall	Mobile Calling
WirelessCom	WirelessCom

blue::promotions (promo $\mapsto$ promo_type)

promotion	promo_type
noPromo	No promotion
Winter 2007	Radio spot
Fall 2008	Advertisement
Christmas 2008	TV spot

blue::customers (customer $\mapsto$ contract_type)

customer_id	cust_name	age_grp	contract_type	base_fee
1234010180	Doe, John	21–30	FullCom	15.0
6785041070	Tanner, Frank	31–40	2for0	20.0
9876090875	Miller, Alice	31–40	FullComLite	10.0

blue::dates (date $\mapsto$ month $\mapsto$ quarter $\mapsto$ year)

date	month	quarter	year
01.01.08	01/08	Q1/08	2008
02.01.08	01/08	Q1/08	2008
30.06.08	06/08	Q2/08	2008
25.08.08	08/08	Q3/08	2008

blue::category (category)

category
tn_tel
tn_misc

Figure 9.3: Fact and dimension tables of company “Blue”.

To improve tool support for the administrators of Federated DW systems, we developed the FedDW tool suite, comprising prototypes of two tools that implement the concepts introduced by this thesis. On the one hand, FedDW *Global Schema Architect (GSA)* is a visual, model-driven design environment for Federated DW systems conforming to the reference architecture proposed in Chapter 6. GSA supports the integration of autonomous Data Mart schemas with UML-based diagrams, including visual design of the global multi-dimensional schema, and allows to design the semantic mappings from local, autonomous Data Marts to the global schema. On the other hand, FedDW *Query Tool* evaluates analytic queries—formulated in SQL—over the global, federated schema designed with GSA. Thus, Query Tool encapsulates the core functionality of Federated DW systems from the business analyst's point of view.

In contrast to previous approaches, FedDW Global Schema Architect combines logical schema design—of both, the import schemas and the global schema—with semantic mapping design in one and the same tool. GSA provides a visual user interface for the design of integrated dimension and fact schemas over autonomous ROLAP Data Marts. By employing a customized notation of UML class and package diagrams, the schema diagrams of cubes in Data Marts are easily comprehensible for the Federated DW administrator. Moreover, GSA follows the model-driven approach for the design of mappings. Each conversion operator specified by the user is represented in an internal model, that can be translated to fragments of the SQL-MDi language. Thus, GSA allows to generate and export the semantic mappings as complete SQL-MDi statements that are executable with the Query Tool. The comprehensive design approach followed in GSA addresses the dependencies between dimensions and facts, and facilitates FDW administration.

9.1 Implementation Architecture of FedDW

The *"FedDW" tool suite* is the prototype implementation of the Federated Data Warehouse reference architecture proposed in Chapter 6. The implementation architecture comprises two front-end tools—Global Schema Architect and Query Tool—as well as the two auxiliary modules *Meta-data Dictionary* and *Dimension Repository*, as depicted in Figure 9.4. In this architecture, the GSA represents the *management interface* of the Federated DW system. GSA allows to design both, the global multi-dimensional schema and the semantic mappings, following the model-driven approach. In turn, the FedDW Query Tool provides a basic OLAP query interface for the business users and OLAP applications.

FedDW tool suite provides *tightly coupled* integration of autonomous Data Marts with a global, mediated schema (cf. the Federated DW reference architecture in Figure 6.1, page 83). Its users—typically the business analysts—simply access the global schema with OLAP query tools. The Query Tool prototype provides a basic OLAP query interface for SQL queries. Its intermediate *federation layer* stores semantic mappings between the import schemas of the Data Marts and the global schema, using a *canonical data model*. So-called wrappers manage the communication between the federation layer and the Data Marts since they send the reformulated queries and ship back the answers [Berger and Schrefl, 2008]. Figure 9.4 depicts the architecture of FedDW tool

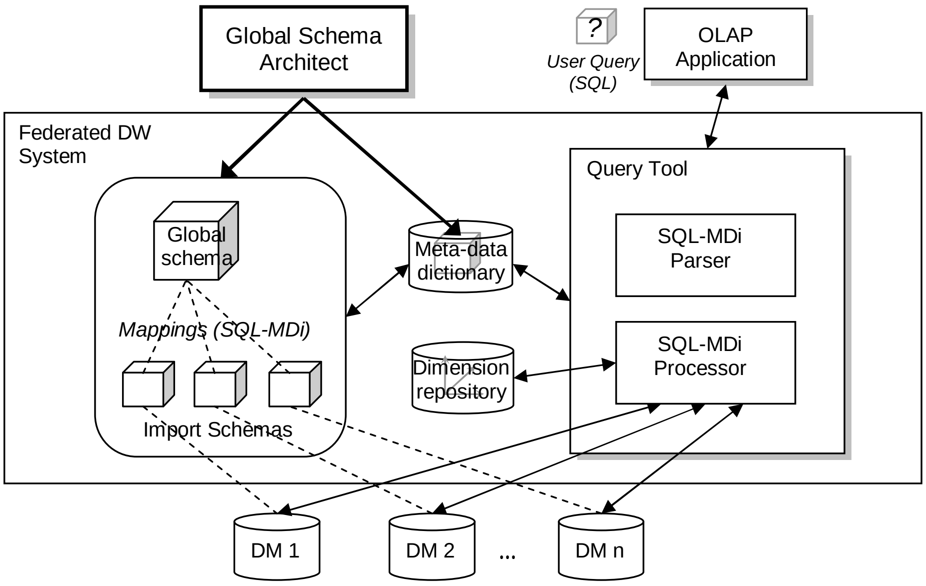


Figure 9.4: Implementation architecture of FedDW tool suite.

suite, comprising mainly of Global Schema Architect (GSA) and the FedDW query tool [Berger and Schrefl, 2009]. The federated approach isolates the global schema from changes in the Data Marts. If the autonomous schemas evolve, only the outdated mappings must be adapted accordingly.

Semantic mappings in the FedDW approach are expressed with the SQL-MDi language, introduced in Chapter 8. Recall that the SQL-MDi language uses three main clauses for defining the global schema and semantic matches: (1) Define [Global] Cube, (2) Merge Dimensions, and (3) Merge Cubes. The Cube clauses specify the attribute structure of both, the virtual global cube and the import schemas of the autonomous Data Marts. For each dimension of the global schema the Merge Dimensions clause defines its members and hierarchy from the imported dimensions. Analogously, the Merge Cubes clause determines how to compute the cells of the global cube from the imported facts [Berger and Schrefl, 2006].

From the analysts' point of view, *FedDW Query Tool* encapsulates the core functionality of the Federated Data Warehouse system. It acts as *mediator*, parsing the SQL-MDi mappings and rewriting each user query that is formulated over the global schema into a set of queries against the local Data Marts [Berger and Schrefl, 2009]. Internally, the FedDW tool expresses the semantic mappings as sequences of Dimension Algebra and Fact Algebra operators [Berger and Schrefl, 2008]. Moreover, the tool translates distributed data from the autonomous Data Marts into the global multi-dimensional schema. Using the data it receives from the autonomous Data Marts, Query Tool computes and instantiates the virtual global cube—as specified by the mapping operators—from which it answers the user queries [Berger and Schrefl, 2009, Rossgatterer, 2008, Brunneder, 2008].

Currently, the FedDW Query Tool prototype offers the following functions:

- FedDW's user interface accepts an SQL-MDi statement and an OLAP query—formulated in SQL—as input. The SQL-MDi statement resolves conflicts among the autonomous Data Marts by specifying the global schema and matching it against the Data Mart schemas. In turn, the SQL OLAP query formulates the user's business question over the virtual global cube, that adheres to the global schema (see Figure 9.12).
- On execution of the query, the prototype computes the virtual global cube as instance of the global schema. It then translates all local data as specified by the semantic matches in the SQL-MDi statement, and executes the SQL query. The user interface displays the current progress of query processing (see Figure 9.12 left).
- Finally, the query result pops up in an additional window that allows to drill into the details – facts and dimensions – of the virtual global cube (see Figure 9.12 right). In OLAP terminology, such operations are denoted as *drill through*.

Global Schema Architect (GSA) is an important administrative utility in the FedDW tool suite, as depicted in Figure 9.4. It stores the mappings between the import schemas of autonomous Data Marts and the global schema—modelled by the user—in the Meta-data Dictionary. Thus, GSA complements the FedDW query tool [Berger and Schrefl, 2009] that retrieves the SQL-MDi code of the mappings automatically without any user interaction. Permanently preserving the outcome of the laborious integration process in the Meta-data Dictionary is crucial for user acceptance of the approach overall.

The functionality of FedDW GSA comprises both, a *UML editor* and a *mapping editor*. On the one hand, UML editor allows to model and validate multi-dimensional schemas as UML class and package diagrams with the extensions defined in the *GSA profile* (see Section 9.2). On the other hand, mapping editor displays the elements of a Data Mart import schema together with buttons and drop-down lists that contain predefined conversion operators. This approach—denoted as *Master Detail* pattern—allowed us to provide a powerful yet clear user interface, displaying each mapping on a single editor pane. To facilitate the user's task, GSA mapping editor adapts the available operators according to the selected model element in context-sensitive manner.

Currently, the GSA prototype—implemented as Eclipse plug-in—provides the following functionality:

- Import the multi-dimensional schemas of local, autonomous Data Marts as UML diagrams.
- Design the global multi-dimensional schema as UML diagram. Optionally, the Federated DW administrator can generate the global schema from one of the import schemas of local Data Marts.
- Configure mappings between each local schema and the global schema, using a visual editor with predefined based on UML class diagrams. The mappings are specified as operators of the SQL-MDi language.
- Export the SQL-MDi code of mappings and the meta-data of (1) the global schema, (2) each of the local schemas, and (3) each mapping to the Meta-data Dictionary.

Two auxiliary components, *Meta-data Dictionary* and *Dimension Repository*, complement FedDW's tool suite. The former improves the usability of the system because it not only keeps track of the import schemas for each autonomous Data Mart, but it also persists the semantic mappings. The latter increases performance of user query answering by providing local copies of the dimensions in the Data Marts, as proven by the experiments conducted in two approaches named "DWS-AQA" [Bernardino et al., 2002] and "Skalla" [Akinde et al., 2003].

9.2 Modelling Primitives for Visual Integration of autonomous Data Marts

The main tasks of Data Mart integration for Federated Data Warehouses are twofold, as discussed in Chapter 7. On the one hand, the Federated DW administrator needs to import the logical, multi-dimensional schemas of the autonomous Data Marts, and model the global cube schema(s). On the other hand, the DW administrator needs a mechanism for defining semantic mappings between schemas, covering all conflicts that occur in practice.

Both these tasks are challenging for the tension between clarity of notation and modelling expressivity. In particular, *schema diagrams* of Data Marts should cover all multi-dimensional properties concisely, but be easy to comprehend. Previous approaches have shown that the UML is well suited for meeting these requirements [Luján-Mora et al., 2006, Prat et al., 2006].

Semantic mappings among Data Marts should also be modelled at the schema level, so to match the abstraction level of the schema diagrams. However, a sufficiently clear and powerful notation for mappings is challenging to define. Formal approaches—surveyed in [Lenzerini, 2002]—are powerful but difficult to comprehend for FDW administrators, while graphical notations—e.g., data mapping diagrams [Luján-Mora et al., 2004]—tend to be either confusingly complex or too less expressive for multi-dimensional data.

The following Section explains how the FedDW approach addresses these issues. (1) To represent Data Mart schemas, FedDW employs a canonical and generic, multi-dimensional data model. (2) For mappings among autonomous schemas, FedDW defines a rich set of conversion operators in the multi-dimensional data model.

9.2.1 Representing Multi-dimensional Schemas

Section 4 of this thesis defined a canonical data model for the FedDW approach to represent properties of multi-dimensional Data Mart schemas and their extensions [Berger and Schrefl, 2008]. The model formally introduced the concepts *cube*, *fact*, *measure*, *dimension*, *aggregation level*, *hierarchy*, *roll-up function* as well as the *dimension functions* members and level. Moreover, the *dimension schema* and *fact schema* hierarchically structure these constructs. Finally, the canonical model specifies *instances* of facts and dimensions, i.e. *members* respectively *cube cells*.

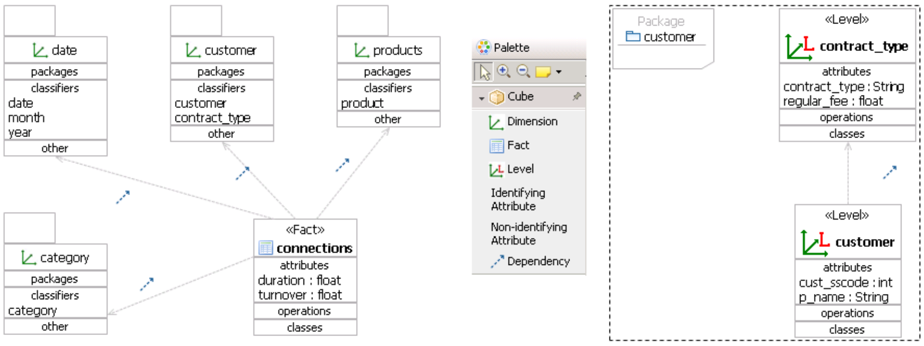


Figure 9.5: GSA schema editor—package diagram of global schema (left) with multi-dimensional schema UML palette; class diagram of customer dimension expanded (right)

For example, Data Mart "Red" consists of fact connections with measures {duration, tn_tel, tn_misc} and dimensions {date, products, customer}. Dimension customer, in turn, has levels [customer] and [contract_type] with non-dimensional attributes name and base_fee, respectively, and contains the hierarchy {customer $\mapsto$ contract_type}.

In the Federated DW reference architecture, the global schema is also represented in the canonical data model. Figure 9.5 shows the global schema over Data Marts Red and Blue, designed with GSA schema editor which is introduced later. While the symbols in Figure 9.5 differ from the notation in Figure 9.1, notice that the underlying concepts are the same as explained above.

Our development of an editor for schema diagrams in the canonical model [Berger and Schrefl, 2008] was guided by the following goals: (1) Facilitate import and exchange of Data Mart meta-data from common commercial ROLAP platforms. (2) Reflect the hierarchical structure of multi-dimensional concepts. (3) Ease the implementation of the visual schema editor as much as possible. If possible, extend an existing tool or modelling environment.

In order to meet these goals, we developed a generic, object-oriented meta-model. The *GSA meta-model (GMM)* defines meta-classes for the multi-dimensional constructs introduced in the FedDW canonical data model [Berger and Schrefl, 2008]. As depicted in Figure 9.6, Fact and Dimension are containers for their respective properties, while Cubes connect facts with dimensions. Schema, in turn, represents Data Marts, containing one or more Cube(s). The generic meta-model allows maximum flexibility for object-oriented design and implementation of a visual schema editor.

First, GSA meta-model complies to the CWM standard [Poole, 2003] in order to ease the import and exchange of Data Mart meta-data. Technically, we adapt the *Unified Multi-dimensional Meta-model* [Prat et al., 2006] and embed the GSA meta-classes to the CWM standard. While most GSA meta-classes correspond with the concepts of [Prat et al., 2006], GMM enhances many details of the previous proposal, as specified below. Most importantly, the Unified Meta-model of [Prat et al., 2006] is comprehensive and powerful, but it lacks support of CWM.

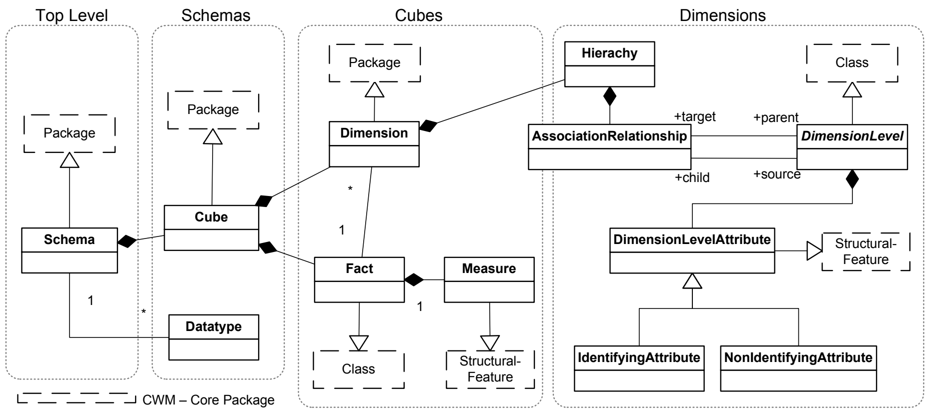


Figure 9.6: Representing Data Mart Schemas—Global Schema Architect meta-model (GMM) [Maislinger, 2009]

Second, GMM defines Schema, Cube and Dimension as extensions of the Package meta-class (cf. Figure 9.6) to represent the hierarchical structure of multi-dimensional concepts. This approach enables to "divide and conquer" multi-dimensional schema modelling since the user can concentrate on one of the detail levels shown in Figure 9.6. Thus, a clear and concise user interface of a visual schema editor is easier to design.

Third, GSA meta-model complies also to the UML standard to facilitate the implementation of a visual schema editor. UML compliance of GMM is easy to ensure by inheriting from the core package (see Fig. 9.6) that CWM and UML share [Poole, 2003]. GMM meta-classes can be embedded to the UML meta-model with *stereotypes* extending UML meta-classes, and bundled into a profile. Thus, instead of developing from scratch, the schema editor is realized as extension of any off-the-shelf UML design environment.

In detail, GMM augments the Unified Multi-dimensional Meta-model [Prat et al., 2006] as follows:

(1) The relationships between GSA meta-classes are mainly compositions, instead of generalizations in [Prat et al., 2006]. Composition-relationships represent multi-dimensional semantics more appropriately (e.g., dimensions "nesting" hierarchies, not "generalizing" hierarchies as in [Prat et al., 2006]). Besides, the approach followed in GMM is similar to the OLAP package of CWM [Poole, 2003].

(2) GMM associates Facts with Dimensions in GMM, while [Prat et al., 2006] connect Facts with DimensionLevels. Again, our meta-model applies a structure similar to the OLAP package of CWM [Poole, 2003]. Moreover, the direct association from Fact to Dimension is easier to implement within a modelling tool (e.g., as package import or dependency in UML).

(3) DimensionLevels in GMM are structured into an aggregation Hierarchy with associations. In contrast, the Unified Meta-Model builds hierarchies with classification relationships [Prat et al., 2006]. As above, the Association-Relationship seems more appropriate, given that hierarchies are basically associations of cardinality 1:n [Golfarelli et al., 1998].

(4) In GMM, DimensionLevels directly connect to the Hierarchy meta-class, while the Unified Multi-dimensional Meta-Model regards DimensionLevel as first class model construct—like Dimension and Fact [Prat et al., 2006]. Since aggregation levels cannot be shared among dimensions in the FedDW conceptual model [Berger and Schrefl, 2008], the former possibility is preferable.

(5) Finally, GMM omits the applicable aggregation functions for Measures in the scope of a Hierarchy. Clearly, these restrictions belong to the conceptual model of multi-dimensional Data Marts [Golfarelli et al., 1998].

9.2.2 Representing Semantic Mappings

In order to integrate autonomous Data Mart schemas, numerous heterogeneities must be considered, as discussed in Chapter 7 and analyzed in [Berger and Schrefl, 2006]. FedDW predefines a rich set of conversion operators at the schema level that apply the Dimension and Fact Algebra introduced in [Berger and Schrefl, 2008]. Table 9.1 lists the conversion operators available in FedDW with the conflicts addressed. The table first specifies all unary operators of GSA together with the meta-model entity on which they apply, then gives all n-ary operators. Table 7.1 in Chapter 7 (see page 115) explain how these operators map to the Fact respectively Dimension Algebra.

When modelling mappings with the FedDW methodology it is important to distinguish the unary from the n-ary operators of Fact Algebra and Dimension Algebra. Unary operators transform facts and dimensions of one autonomous Data Mart schema. In turn, n-ary operators specify how to merge sets of transformed facts and dimensions, so that the result conforms to the global schema [Berger and Schrefl, 2008]. The difference between unary and n-ary conversions in the FedDW methodology has been explained in depth in Chapter 7 (see Algorithms 7.4 on page 106 and 7.5 on page 109, and Subsection 7.2.3, pp. 111).

Each FedDW mapping simply arranges a sequence of these operators. For instance, assume that company Red adopts Blue's schema—except the promotion dimension—as global schema of the federation (Fig. 9.5). We show a possible mapping of Red's connections cube to the global schema in Figures 9.7 (p. 155) and 9.8 (p. 156), and give some examples on how to repair the heterogeneities in the Red and Blue example. Both Figures depict the GSA mapping editor which is introduced in Section 9.3. The SQL-MDi code of the mappings is visible in the upper right corner of the figures.

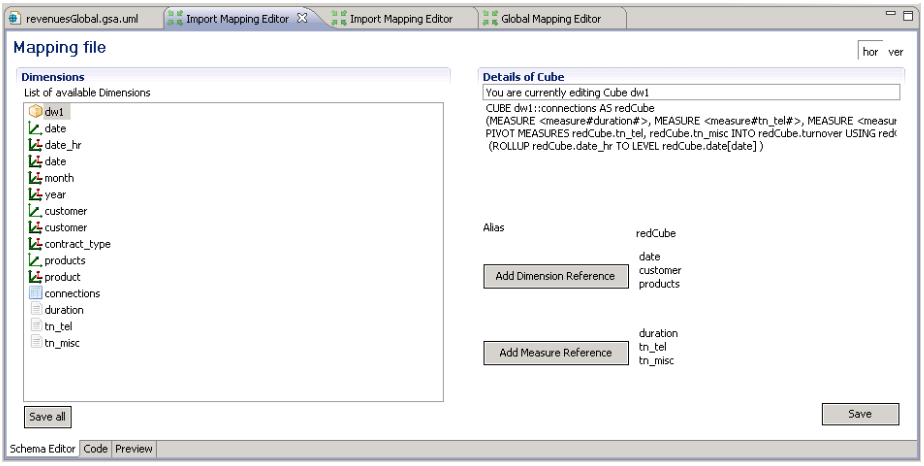
As explained in the introduction (cf. Figure 9.1), Red and Blue record comparable turnovers, but use different schemas. To conform to the global schema, the connections data of Red must be transformed with a PIVOT MEASURES operator (cf. Table 9.1) to address the schema–instance conflict among Red and Blue: it allows to merge the original measures tn_tel and tn_misc into one measure turnover, and to generate the category dimension. The PIVOT MEASURES operator applied in line 3 of the SQL-MDi preview in Figure 9.7 generates a new cube schema with measures {duration, turnover} and dimensions {date, products, customer, category}. It defines values "tn_tel" and "tn_misc" as members of the *context* dimension category with one level [category] [Berger and Schrefl, 2008].

Table 9.1: FedDW conflict resolution—predefined operators in GSA.

Facts: conflicts addressed	**Relevant operator of FedDW approach (implemented in GSA)**
Schema-instance conflicts	Merge measures: PIVOT MEASURES (Fact) (Converts fact context into members of a new dimension)
Schema-instance conflicts	Split measures: PIVOT SPLIT MEASURES (Fact) (Generates "contextualized facts" from members)
Dimensionality	Choose attributes: add DIM reference (Cube) (Number of references determines cube dimensionality)
Different measures	Choose measures: add MEASURE reference (Cube)
Domain conflicts (measures)	Convert domains: CONVERT MEASURES APPLY ... (Measure)
Naming conflicts (measures and dimension attributes)	Rename attributes: operator "–> ..." (with new name string—Measure, Dimension)
Heterogeneous base levels	Roll-up dimension attributes: ROLLUP TO LEVEL ... (Dimension) (Corrects domain conflicts among dimension attributes)
Overlapping cube cells (fact extensions)	Join cubes: MERGE CUBES (*n-ary*) Derive measure values: AGGREGATE MEASURE (*n-ary*)
Dimensions: conflicts	**Relevant operator of FedDW approach (implemented in GSA)**
Heterogeneous hierarchies	Map corresponding levels: add level reference [...] (Dimension) (Import levels of dimensions referenced by cube import)
Domain conflicts (level and non-dimensional attributes)	Convert attribute domains: CONVERT ATTRIBUTES APPLY ... (Dimension)
Naming conflicts (levels)	Rename attributes: operator "–> ..." (with new name string; Level)
Naming conflicts (non-dimensional attributes)	Map non-dimensional attributes: MATCH ATTRIBUTES (within Merge Dimensions clause—*n-ary*)
Overlapping members (dimension extensions)	Merge sets of members: MERGE DIMENSIONS (*n-ary*)
Heterogeneous roll-up functions in hierarchies	Overwrite roll-up hierarchies: RELATE Expression (within Merge Dimensions clause—*n-ary*)
Conflicting values of non-dimensional attributes	Correct attribute values: add RENAME function (within Merge Dimensions clause—*n-ary*)

The mapping in Figure 9.7 defines several other unary conversion operators for Red's Data Mart. For instance, the base levels of the date dimensions differ (cf. Fig. 9.1). Operator ROLLUP redCube.date_hr TO LEVEL [date] in line 4 ensures that the base level of Red's date dimension matches the granularity of the global schema (cf. Fig. 9.5). This operator treats dimension Red::date as if [date], [month] and [year] were its only levels.

In turn, the global mapping contains the n-ary conversion operators applied to the Red and Blue example. For instance, operator "Merge Dimensions 2" (shown in the SQL-MDi preview of Fig. 9.8) repairs heterogeneity among the customer dimensions of Red and Blue. In particular, it "joins" the members of the [customer] level (RELATE operator, line 2), and specifies that non-dimensional attribute p_name of Red maps to cust_name of Blue (MATCH ATTRIBUTES operator, line 3). Finally, line 4 converts the domain of attribute base_fee in Blue's customer dimension, employing function usd2Eur() with the CONVERT ATTRIBUTES operator.



List of available Dimensions

dw1
date
date_hr
date
month
year
customer
customer
contract_type
products
product
connections
duration
tn_tel
tn_misc

You are currently editing Cube dw1

```
CUBE dw1::connections AS redCube
(MEASURE <measure#duration#>, MEASURE <measure#tn_tel#>, MEASURE <measur
PIVOT MEASURES redCube.tn_tel, redCube.tn_misc INTO redCube.turnover USING red(
 (ROLLUP redCube.date_hr TO LEVEL redCube.date[date] )
```

Alias: redCube

date
customer
products

duration
tn_tel
tn_misc

Figure 9.7: GSA Import Mapping Editor (cube element "dw1" selected left)

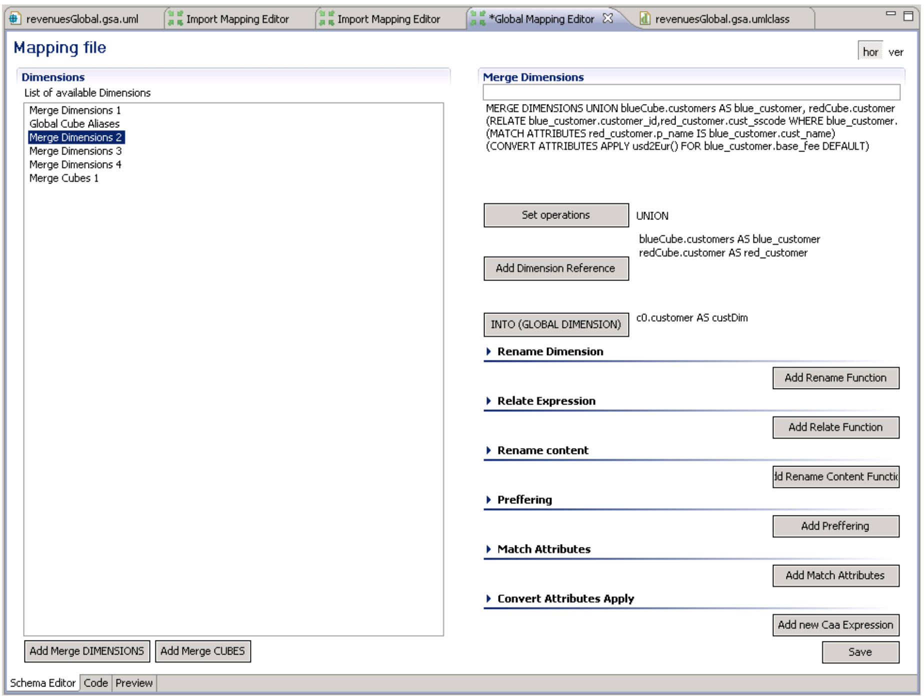
9.3 Global Schema Architect Prototype

FedDW Global Schema Architect is a design environment for Data Mart integration, providing the *schema editor* and *mapping editor* visual tools. Its architecture is depicted in Figure 9.9. The following section explains our design rationale and sketches GSA's functionality from the user's perspective [Maislinger, 2009]. (1) To represent Data Mart schemas, GSA adapts the UML standard for the DW domain, adding the concepts of the FedDW canonical data model [Berger and Schrefl, 2008]. (2) For defining the mappings among autonomous Data Mart schemas, GSA provides a *Master Detail* Editor with click-able buttons for the conversion operators listed in Table 9.1. (3) To embed GSA into a Federated DW infrastructure, the schema import and SQL-MDi/meta-data export modules provide external interfaces.

9.3.1 Design Rationale of GSA Schema Editor

In order to represent the GSA meta-model in the UML, we developed a *profile* called GSA (Global Schema Architect). Profiles are lightweight extensions to UML, customizing its standard meta-model for particular modelling domains [(OMG), 2009]. The UML allows the extension mechanisms *stereotype*, *tagged value* and *icon* for defining the features of a profile, whereas new meta-classes must not be introduced [Hitz et al., 2005].

The GSA profile inherits all properties of the UML 2.1 meta-model and extends the Class, Package and Property meta-classes with the stereotypes listed in Table 9.2 and depicted in Figure 9.10. Notice that the stereotypes introduced in the profile correspond to the concrete GSA meta-classes of the schema, cube and dimension layers introduced in Subsection 9.2.1 (cf. Figure 9.6), except for the Schema concept which is represented in the model itself. Thus, the GSA profile is the concretion of the generic GSA meta-model for the UML language. Compared to GMM, we preferred the Property meta-class to its superclass StructuralFeature because Property is more commonly used in the UML



Dimensions

List of available Dimensions

Merge Dimensions 1
Global Cube Aliases
Merge Dimensions 2
Merge Dimensions 3
Merge Dimensions 4
Merge Cubes 1

Merge Dimensions

```
MERGE DIMENSIONS UNION blueCube.customers AS blue_customer, redCube.customer
(RELATE blue_customer.customer_id,red_customer.cust_sscode WHERE blue_customer.
(MATCH ATTRIBUTES red_customer.p_name IS blue_customer.cust_name)
(CONVERT ATTRIBUTES APPLY usd2Eur() FOR blue_customer.base_fee DEFAULT)
```

Set operations: UNION

blueCube.customers AS blue_customer
redCube.customer AS red_customer

INTO (GLOBAL DIMENSION): c0.customer AS custDim

Figure 9.8: GSA Global Mapping Editor (Merge Dimensions element selected left)

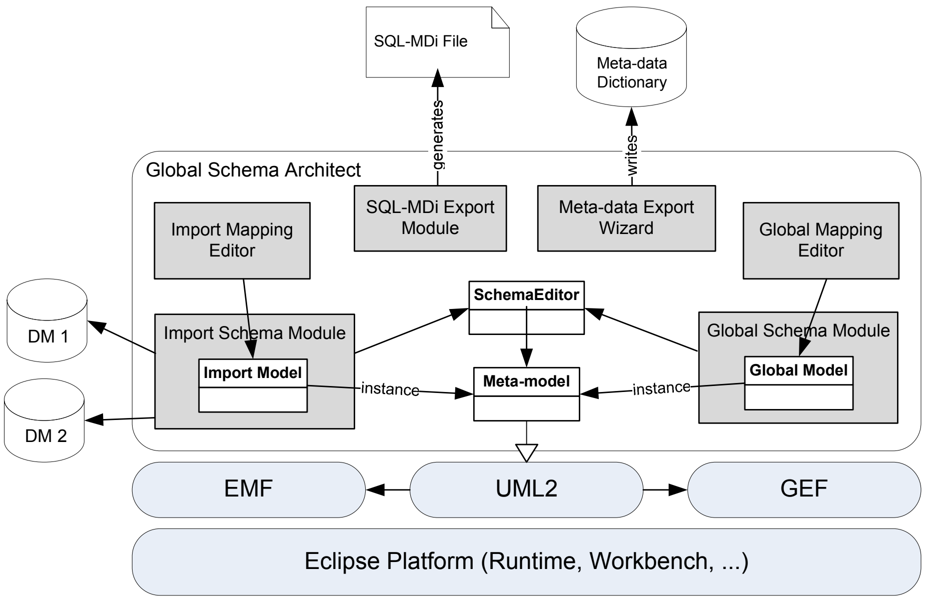


Figure 9.9: Modules of the FedDW GSA Eclipse plug-in [Maislinger, 2009]

specification [(OMG), 2009]. Moreover, the GSA profile introduces constraints on the stereotypes—defined below—but does not include any new data types or tagged values.

To ensure the extensions be used correctly, the GSA profile formulates *constraints* in the *Object Constraint Language (OCL)*. OCL constraints are declarative rules for the well-formedness of models that apply the extensions defined in a profile [Warmer and Kleppe, 1999]. The GSA profile specifies the following OCL invariants as constraints on its extensions:

Cube – check_has_adequate_subelements: each Cube package contains only dimensions, facts and UML dependencies, and is not nested into another package.

```
self.nestingPackage->size = 0
self.nestedPackage->forAll (oclIsTypeOf(Dimension))
self.contents->forAll (oclIsTypeOf(Fact) or
    oclIsKindOf(Dependency))
```

Dimension – check_has_levels: allows only Level classes within Dimension packages.

```
self.contents->forAll (oclIsTypeOf(Level))
```

Dimension – check_no_dependencies: disallows any UML dependencies in Dimension packages.

```
self.clientDependency->size = 0
```

Fact – check_has_measures: allows only ≥ 1 Measure properties in Fact classes.

```
self.ownedAttribute->size >= 1
self.ownedAttribute->forAll (oclIsTypeOf(Measure))
```

Fact – check_dependencies: all UML dependencies leaving from Fact classes must point to Dimension packages. At least one such dependency is mandatory.

```
self.clientDependency->size >= 1
self.clientDependency->forAll(supplier->forAll(
    oclIsTypeOf(Dimension)))
```

Level – check_hierarchy: forbids circles between the UML dependencies that form the hierarchy of Level classes in a Dimension.

```
not self.allSuppliers->includes(self)
```

Level – check_dependencies: all UML dependencies leaving from Level class must point to another Level.

```
self.clientDependency->forAll(supplier->forAll(
    oclIsTypeOf(Level)))
```

Level – check_one_identifying_attribute: allows only one attribute owned by the Level class to be an IdentifyingAttribute.

```
self.ownedAttribute->select(oclIsTypeOf(
    IdentifyingAttribute))->size=1
```

Schema editor provides a modelling environment for UML class and package diagrams for both, the *Import Schema* and *Global Schema*. Before defining any mappings, the user has to specify the global schema of the *GSA project*, which collects the meta-data of all schemas and mappings. To facilitate this task, the user can optionally generate the global schema from an import schema, and then adapt it.

Table 9.2: GSA profile—stereotypes, constraints and icons [Maislinger, 2009]

Stereotype	Constraints	UML meta-class	Icon
Cube	check_has_adequate_subelements	Package	📦
Dimension	check_has_levels check_no_dependencies	Package	📈
Level	check_hierarchy check_dependencies check_one_identifying_attribute	Class	📈
Fact	check_has_measure check_dependencies	Class	▦
IdentifyingAttribute	—	Property	—
NonIndentifyingAttribute	—	Property	—
Measure	—	Property	—

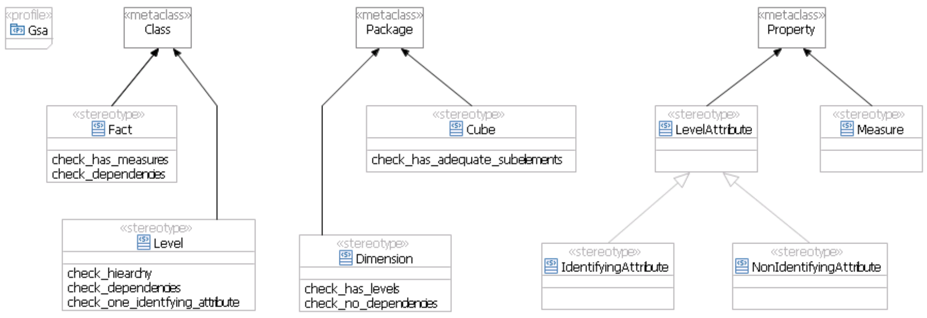


Figure 9.10: Profile Definition Diagram—extensions of the GSA profile

For high usability, GSA schema editor predefines a palette of multi-dimensional modelling elements (cf. Figure 9.5). The available elements correspond to the concepts of the GSA meta-model, and are defined by the extensions of the GSA profile (Table 9.2). Mouse clicks add new schema elements from the palette. Moreover, the user may “zoom in” to the class diagram of dimensions, that are visualized as packages in the main editor view (as shown in Figure 9.5). To ensure that the schema meets all constraints defined in the profile, the user should invoke the “validate model” function from the editor pane.

9.3.2 Design Rationale of the GSA Mapping Editors

The *Import Mapping Editor* and *Global Mapping Editor* of GSA predefine all conversion operators of the Dimension and Fact Algebra listed in Table 9.1 as click-able buttons. This approach offers two advantages. First, the interplay between fact and dimension integration is easy to consider by visualizing all dependencies among the facts and dimensions. Second, the predefined buttons allow for a powerful yet clear user interface design. In contrast to mapping diagrams structured in packages [Luján-Mora et al., 2004], mappings in the GSA approach easily fit onto one single editor screen.

While the Import mapping editor (Fig. 9.7) applies unary conversion operators to the import schema of one Data Mart, the Global mapping editor (Fig. 9.8) employs n-ary operators on the set of import schemas in the GSA project. Internally, the mapping editors relate the Fact Algebra and Dimension Algebra operators [Berger and Schrefl, 2008] to corresponding SQL-MDi fragments. GSA uses the generated SQL-MDi fragments to compose the code preview shown in the upper right corner of the mapping editors. Moreover, the SQL-MDi code can be exported to the file system (see export functionality in Subsection 9.3.3).

The user interface of both GSA mapping editors conforms to the Master Detail design pattern, splitting the user interface into *master page* and *detail page* [Beier et al., 2003]. While the master page on the left-hand side of the editor pane lists all model elements (e.g., dimensions, levels), the detail page on the right-hand side shows all properties of one selected element. GSA mapping editor embeds the predefined buttons and drop-down fields for the FedDW conversion operators in the detail page. The available operations adapt to the selection made on the master page in context sensitive manner. For example, notice that the master page in Figure 9.7 lists all elements of Red's schema, while the detail page contains the properties and available conversions for the selected element, cube "dw1". Analogously, the detail page of the global mapping editor in Figure 9.8 shows all n-ary operators of Dimension Algebra (cf. Table 9.1) for the customer dimensions selected in the master page (element "Merge Dimensions 2").

9.3.3 Import and Export Functionality of FedDW GSA

As depicted in Figure 9.9, Global Schema Architect contains an *Import Schema* module and *model export* modules (SQL-MDi generator and Meta-data Wizard). The import function enables the GSA to comfortably load Data Mart schemas. The export functionality defines the interface to FedDW query tool, introduced in [Berger and Schrefl, 2009].

The Import Schema module connects to autonomous ROLAP Data Marts to retrieve local meta-data. To infer the logical schema, it analyzes the public keys and primary keys. The import heuristics suggest adequate stereotypes for the schema elements (cube, measure, dimension, level, etc.) which the user can change manually. Each import schema is created as UML diagram applying the GSA profile (cf. Figure 9.5).

The *SQL-MDi Generator* assembles the SQL-MDi fragments from the conversions modelled in the mapping editors, and writes the SQL-MDi code file. The generator automatically recognizes the global mapping and all import mappings in the GSA project. If the export throws syntax warnings, the mappings have to be completed and/or fixed. Otherwise, the generated mapping file can be tested immediately with FedDW query tool [Berger and Schrefl, 2009] which is useful for verifying whether the mappings meet the user's intentions.

Meta-data wizard exports the meta-data modelled in the current GSA project to the Meta-data Dictionary upon user request (cf. Figure 9.9). Exporting meta-data is important for OLAP querying: FedDW query tool retrieves both, the connection meta-data and the SQL-MDi code of the mappings from the Dictionary [Berger and Schrefl, 2009].

Table 9.3: Conflict resolution strategies of FedDW for facts, and corresponding language support.

Conflict addressed	**Resolution technique**	**Relevant SQL-MDi clause of FedDW**
–	Import fact table ("cube")	CUBE operator; Example: see lines 1, 5, and 13 in Figure 9.11
Dimensionality conflicts	Import dimension attributes	DIM keyword; Example: see lines 2, 7, 9, and 11 in Figure 9.11
Overlapping / disjoint measures	Import measure attributes	MEASURE keyword; Example: see lines 2, 6, and 7 in Figure 9.11
Naming conflicts (measures and dimension attributes)	Rename attributes	Operator "–>"; Examples in Fig. 9.11: line 6 (measure attribute) line 9 (dimension attribute)
Heterogeneous base levels	Roll-up dimension attributes	ROLLUP clause of CUBE Example: see line 4 in Figure 9.11
Domain conflicts (measures)	Convert measure domains	CONVERT MEASURES APPLY ... clause of CUBE Example: see line 12 in Figure 9.11
Overlapping cube cells (fact table extensions)	Join fact tables ("cubes")	MERGE CUBES ... [set-operation] Example: see line 23 in Figure 9.11
Overlapping cube cells	Aggregate measure values	AGGREGATE MEASURE clause of MERGE CUBES operator Example: see line 24 in Figure 9.11
Schema–instance conflicts (Fact context ⇒ members of a new dimension)	Merge measures	PIVOT MEASURES... INTO clause Example: see line 3 in Figure 9.11
Schema–instance conflicts (Dimension members ⇒ "contextualized facts")	Split measures	PIVOT MEASURE... BASED ON clause (Not used in sample query.)

9.4 Query Tool Prototype

FedDW Query Tool is the prototype of an interpreter for SQL-MDi statements, combined with an SQL parser, for analytic queries across autonomous Data Marts in Federated DW systems. The prototype implementation of FedDW Query Tool integrates an OLAP application with the internal query processing component, as depicted in Figure 9.4. Thus, the Query Tool acts as both, the mediator of the global, federated schema, and the interface to the OLAP analyst. On the one hand, the tool parses SQL-MDi mappings and generates *query plans* for each user query. A query plan is a sequence of queries against the local Data Marts that is equivalent to the original query, formulated over the global schema. On the other hand, the tool translates data from the autonomous Data Marts to compute the virtual global cube, interprets the query, and presents the query answers back to the user [Berger and Schrefl, 2009, Rossgatterer, 2008, Brunneder, 2008].

The FedDW approach supports numerous strategies for resolving heterogeneities among the facts and dimensions of autonomous Data Marts, as explained in Chapter 7 of this thesis. The operators of the Dimension and Fact Algebra introduced in Chapter 7 are available in the FedDW Query Tool through the SQL-MDi language, proposed in Chapter 8. Tables 9.3 and 9.4 list the conflicts discussed in Chapter 5, and describe the corresponding resolution techniques employed in FedDW. The two tables also map the resolution techniques to the relevant SQL-MDi operators.

Table 9.4: Conflict resolution strategies of FedDW for dimensions, and corresponding language support.

Conflict addressed	**Resolution technique**	**Relevant SQL-MDi clause**
Heterogeneous hierarchies	Match corresponding levels	MAP LEVELS clause of DIM keyword Example: see lines 8 and 10 in Fig. 9.11
Naming conflicts (level attributes and non-dimensional attributes)	Rename attributes	Operator "–>" (level attributes) Example: see line 10 in Fig. 9.11 MATCH ATTRIBUTES clause of MERGE DIMENSIONS operator (non-dimensional attributes) Example: see line 18 in Fig. 9.11
Overlapping members (dimension extensions)	Merge dimension members	MERGE DIMENSIONS ... [set-operation] Example: see lines 15, 16, 20, and 22 in Fig. 9.11
Heterogeneous roll-up functions (i.e. hierarchies between members)	Overwrite roll-up hierarchies	RELATE ... USING HIERARCHY OF ... clause of MERGE DIMENSIONS operator Example: see line 17 in Fig. 9.11
Domain conflicts (level attributes and non-dimensional attributes)	Convert attribute domains	CONVERT ATTRIBUTES APPLY ... clause of MERGE DIMENSIONS operator Example: see line 19 in Fig. 9.11
Conflicting values of non-dimensional attributes	Correct attribute values	RENAME clause of MERGE DIMENSIONS Example: see line 21 in Fig. 9.11

9.4.1 FedDW Query Tool Usage Example

Let us now assume that the management of mobile network provider Red decides to integrate its sales Data Mart with Blue's under a federation. The fact and dimension tables of Red and Blue are shown in Figs. 9.2 and 9.3, respectively. In particular, the two Data Mart schemas are briefly characterized as follows:

- Provider "Red" stores the data of turnover within the measures duration, tn_tel and tn_misc, categorized by the dimensions date, customer and products (see Figure 9.2). The aggregation levels of the dimensions are given in brackets. As monetary unit for the measures Red's schema uses Euros.
- "Blue" stores its turnover data within the measures duration and turnover, structured by the dimensions date, customer, product, promotion and category (see Figure 9.3). Again, the aggregation levels of these dimensions are given in brackets. Blue's schema uses US-Dollars as monetary unit.

Using FedDW, the Data Marts of Red and Blue may be integrated with the SQL-MDi code listed in Figure 9.11. This statement matches the local facts and dimensions with the global schema, which is very similar to Red's schema. Figure 9.13 depicts the fact table of the virtual global cube, as specified by the sample SQL-MDi statement. For brevity, the figure omits the global dimension tables (except customer).

In what follows, we demonstrate how "Red" applies SQL-MDi to resolve the heterogeneities among the Red and Blue sales Data Marts, using the sample statement of Figure 9.11. For a systematic overview of the available options please refer to Tables 9.3 and 9.4. In detail, the sample SQL-MDi statement applies the following conflict resolution techniques:

[Lines 1–12]: In the first part of the query the user specifies the Data Marts' import schemas. Notice that the DEFINE keyword is needed only once. The two CUBE clauses (lines 1 and 5) import the example Data Marts, referencing their measure and dimension attributes with the adequate keywords (MEASURE, DIM). It is mandatory to specify an alias for each cube (AS keyword) to facilitate access to its properties later on.

[Lines 1–4]: The CUBE clause imports Red's Data Mart and performs two transformations. First, line 3 repairs the schema-instance conflict among the two fact tables by merging the existing measures tn_tel and tn_misc into a single turnover measure. This operation generates the *context dimension* "category". Second, line 4 rolls-up the dimension attribute date_hr to level date, so to match Blue's date granularity.

[Lines 5–12]: The other CUBE clause imports Blue's Data Mart and specifies four transformations. First, the MAP LEVELS clause in line 8 restricts the levels of the date dimension. Only those levels referenced in square brackets are kept in the import schema. Notice that the DIM keyword automatically imports all levels if MAP LEVELS is omitted (as for the dimensions of Red's Data Mart). Second, line 9 renames the dur_min measure attribute. Third, lines 6 and 10 rename dimension attribute customer and level customer_id, respectively. Fourth, line 12 calls the stored procedure usd2Eur() to convert the domain of Blue's turnover measure from US-$ to Euro.

[Lines 2 and 9–11]: Dimensionality of the import schema is determined by the number of explicit references to dimensions of the underlying Data Mart, using the DIM keyword. Recall that schema Blue defines the additional promotions dimension, which is impossible to match with any of Red's dimensions. Thus, dimension promotions is excluded from the import schema of Blue's Data Mart (see line 11, cf. with line 2) by omitting the DIM promotions import specification. This way, the dimensionality conflict among the two Data Marts is repaired; considering the category dimension generated in line 3, both import schemas contain four-dimensional cubes.

[Line 13]: The global schema is not defined immediately. Instead, the GLOBAL CUBE clause is a forward declaration, simply reserving its name and alias for later use (cf. line 23).

[Lines 15–22]: Next, the MERGE DIMENSIONS clauses specify how to populate the global dimensions with tuples ("members"). In our example, the customer dimensions need several directives for conflict resolution: (a) the member hierarchy of Red's customers should override Blue's hierarchy (line 17); (b) the non-dimensional attributes name and cust_name are matched (line 18); and (c) the domain of base_fee is converted from US-$ to Euro (line 19). Moreover, line 21 changes value 'HandyTel' in Red's product dimension to the correct 'HandyTelCo'.

[Lines 23–25]: Finally, the global fact table is determined by the MERGE CUBES clause (line 23), which completes the forward declaration given in line 13. In order to compute the correct values for all measures, line 24 specifies the adequate aggregation function to apply for *overlapping* cube cells.

```
DEFINE CUBE red::connections AS c1
  (MEASURE c1.duration, MEASURE c1.tn_tel, MEASURE c1.tn_misc, DIM
      c1.date_hr, DIM c1.cust_sscode, DIM c1.product,
   PIVOT MEASURES c1.tn_tel, c1.tn_misc INTO c1.turnover USING c1.
       category)
  (ROLLUP c1.date_hr TO LEVEL c1.date[date])
CUBE blue::connections AS c2
  (MEASURE c2.dur_min -> duration,
  MEASURE c2.turnover, DIM c2.date
    (MAP LEVELS c2.date( [date], [month], [year] )),
  DIM c2.customer -> cust_sscode
   (MAP LEVELS c2.customer ( [customer_id -> cust_sscode] , [
       contract_type] ),
  DIM c2.product, DIM c2.category)
   (CONVERT MEASURES APPLY usd2Eur() FOR c2.turnover DEFAULT) )
GLOBAL CUBE dw0::sales AS c0

MERGE DIMENSIONS c1.date_hr AS d1, c2.date AS d2 INTO c0.date AS d0

MERGE DIMENSIONS c1.cust_sscode AS d3, c2.cust_sscode AS d4 INTO c0
     .customer AS d5
  (RELATE d3.cust_sscode, d4.customer_id WHERE d3.cust_sscode=d4.
      customer_id USING HIERARCHY OF d3)
  (MATCH ATTRIBUTES d3.name IS d4.cust_name)
  (CONVERT ATTRIBUTES APPLY usd2Eur() FOR d4.base_fee DEFAULT)
MERGE DIMENSIONS c1.product AS d6, c2.product AS d7 INTO c0.product
     AS d8
  (RENAME d6.product >> 'HandyTelCo' WHERE c1.product='HandyTel')
MERGE DIMENSIONS c1.category AS d9, c2.category AS d10 INTO c0.
     category AS d11
MERGE CUBES c1, c2 INTO c0 ON date, customer, product, category
  AGGREGATE MEASURE duration IS SUM OF duration, AGGREGATE MEASURE
      turnover IS SUM OF turnover
 (MEASURE duration, MEASURE turnover, DIM date, DIM customer, DIM
     product, DIM category)
```

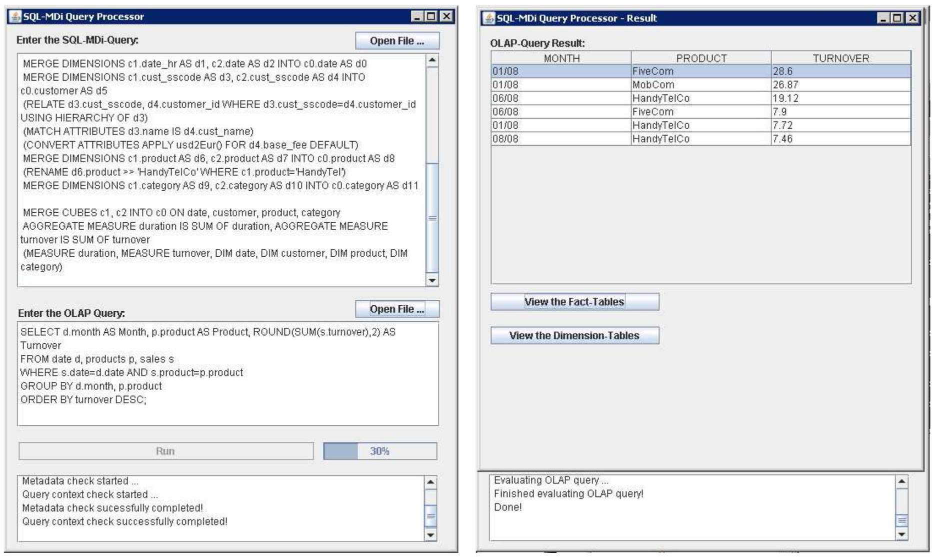
Figure 9.11: Example SQL-MDi statement, integrating Red's and Blue's Data Marts.

To examine the global cube defined by the SQL-MDi statement, the user enters an SQL OLAP query as well. In our example, the user has been interested in "How much was our company's turnover over the last year, grouped by month and product?", as depicted in Figure 9.12 (left). Upon successful execution of the sample SQL-MDi statement, FedDW returns the virtual global cube. Then it evaluates the SQL OLAP query and displays its result (Figure 9.12, right).

9.4.2 FedDW Query Tool Implementation

Having demonstrated the capabilities of the FedDW tool in our sample use case, the following section will briefly summarize its implementation. It was guided by the following aims: (1) Minimize the design effort through the reuse of well-known software design patterns. (2) Use standards whenever available for FedDW's internal data model in order to maximize the interoperability of FedDW. (3) Support multiple platforms (i.e. databases and operating systems).

The SQL-MDi parser component of FedDW checks the syntactic and semantic correctness of the SQL-MDi matching expressions, i.e. whether all clauses



```
 MERGE DIMENSIONS c1.date_hr AS d1, c2.date AS d2 INTO c0.date AS d0
 MERGE DIMENSIONS c1.cust_sscode AS d3, c2.cust_sscode AS d4 INTO
c0.customer AS d5
 (RELATE d3.cust_sscode, d4.customer_id WHERE d3.cust_sscode=d4.customer_id
USING HIERARCHY OF d3)
 (MATCH ATTRIBUTES d3.name IS d4.cust_name)
 (CONVERT ATTRIBUTES APPLY usd2Eur() FOR d4.base_fee DEFAULT)
 MERGE DIMENSIONS c1.product AS d6, c2.product AS d7 INTO c0.product AS d8
 (RENAME d6.product >> 'HandyTelCo' WHERE c1.product='HandyTel')
 MERGE DIMENSIONS c1.category AS d9, c2.category AS d10 INTO c0.category AS d11

 MERGE CUBES c1, c2 INTO c0 ON date, customer, product, category
 AGGREGATE MEASURE duration IS SUM OF duration, AGGREGATE MEASURE
turnover IS SUM OF turnover
 (MEASURE duration, MEASURE turnover, DIM date, DIM customer, DIM product, DIM
category)
```

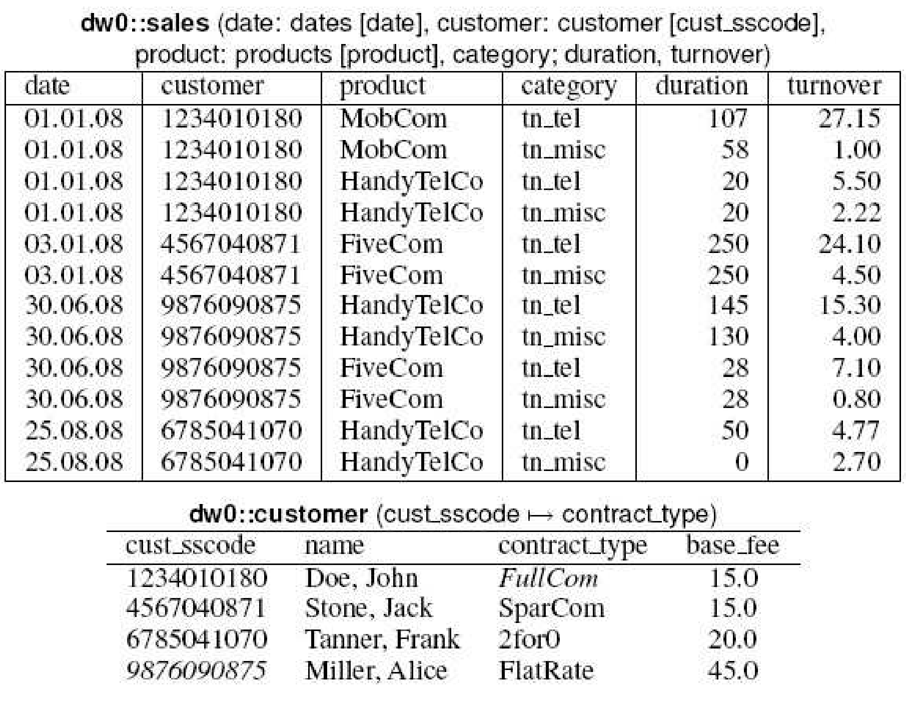
```
SELECT d.month AS Month, p.product AS Product, ROUND(SUM(s.turnover),2) AS
Turnover
FROM date d, products p, sales s
WHERE s.date=d.date AND s.product=p.product
GROUP BY d.month, p.product
ORDER BY turnover DESC;
```

```
Metadata check started ...
Query context check started ...
Metadata check sucessfully completed!
Query context check successfully completed!
```

MONTH	PRODUCT	TURNOVER
01/08	FiveCom	28.6
01/08	MobCom	26.87
06/08	HandyTelCo	19.12
06/08	FiveCom	7.9
01/08	HandyTelCo	7.72
08/08	HandyTelCo	7.46

```
Evaluating OLAP query ...
Finished evaluating OLAP query!
Done!
```

Figure 9.12: FedDW Query Tool: graphical user interface.



dw0::sales (date: dates [date], customer: customer [cust_sscode], product: products [product], category; duration, turnover)

date	customer	product	category	duration	turnover
01.01.08	1234010180	MobCom	tn_tel	107	27.15
01.01.08	1234010180	MobCom	tn_misc	58	1.00
01.01.08	1234010180	HandyTelCo	tn_tel	20	5.50
01.01.08	1234010180	HandyTelCo	tn_misc	20	2.22
03.01.08	4567040871	FiveCom	tn_tel	250	24.10
03.01.08	4567040871	FiveCom	tn_misc	250	4.50
30.06.08	9876090875	HandyTelCo	tn_tel	145	15.30
30.06.08	9876090875	HandyTelCo	tn_misc	130	4.00
30.06.08	9876090875	FiveCom	tn_tel	28	7.10
30.06.08	9876090875	FiveCom	tn_misc	28	0.80
25.08.08	6785041070	HandyTelCo	tn_tel	50	4.77
25.08.08	6785041070	HandyTelCo	tn_misc	0	2.70

dw0::customer (cust_sscode ↦ contract_type)

cust_sscode	name	contract_type	base_fee
1234010180	Doe, John	*FullCom*	15.0
4567040871	Stone, Jack	SparCom	15.0
6785041070	Tanner, Frank	2for0	20.0
9876090875	Miller, Alice	FlatRate	45.0

Figure 9.13: Fact table "sales" of virtual global cube "dw0".

conform to the EBNF grammar specification of the SQL-MDi language, and whether all referenced schema elements exist in the autonomous Data Marts. The parser communicates with the Meta-data Dictionary (cf. Section 9.1) to verify the query syntax and semantics. If the parser detects errors it returns a description and gives hints on possible reasons.

From the input SQL-MDi statement the parser component generates a data structure called the *operator tree*. It contains the sequence of Fact and Dimension Algebra specified by the SQL-MDi matching expression. The leaf nodes of the operator tree contain the unary algebraic transformations of facts and dimensions, whereas the internal nodes – called "structure nodes" – represent the binary operators, specifying how to merge the intermediate results to the virtual global cube. Upon completion of parsing, the ordering of operators in the tree is optimized algebraically such that the size of intermediate results is reduced as early as possible [Brunneder, 2008].

FedDW's query processor component receives the operator tree from the SQL-MDi parser and computes the query result over the virtual global cube. All unary transformations in the leaf nodes are traversed "from left to right". Then these intermediate results are combined step by step, according to the structure node operators. The ordering of nodes given in the operator tree remains unchanged [Rossgatterer, 2008].

The critical phase of the query processing algorithm is the generation of an optimal query plan. This means that the processor reformulates the original user query into a sequence of queries over the autonomous Data Marts. If the operator tree refers to dimensions that exist in the Dimension Repository (cf. Section 9.1), the processor eliminates the according sub-queries from the query plan and instead reads the intermediate results from the repository. Thus, the amount of data shipped between the autonomous DW systems is reduced, as shown by [Bernardino et al., 2002, Akinde et al., 2003].

In the current version of FedDW Query Tool prototype, the query processing algorithm combines *algebraic optimization* of the query plan with *data shipping* for query answering. That is, apart from checking the Dimension Repository, the query processor leaves the operator tree it receives from the query parser unchanged. Query answering, in turn, strictly conforms to the data shipping paradigm, since the Query Tool receives local answers of the Data Marts, translates everything to the global schema, and instantiates the virtual global cube. Only then the OLAP query result is evaluated.

The implementation of a language parser is known to be rather mechanical and thus easy to automatize. Several software tools – called *compiler generators* – have been developed that enable the automatized generation of parsers. FedDW's parser component has been generated with the JavaCC framework, based upon the formal description – the grammar – of the underlying SQL-MDi language, augmented with so-called production rules [Brunneder, 2008].

In order to optimize the runtime performance and extensibility of the parser and processor components, we applied several well-known software design patterns. As far as the parser is concerned, the generation of the operator tree from the SQL-MDi statement conforms to the *Visitor* design pattern. It consists of (1) a hierarchy of classes representing the nodes of some data structure to be traversed, and (2) the visitor class. Each node class provides an `Accept()`

method with the visitor class as input parameter. Moreover, the visitor class contains a corresponding `visitElement()` method for each `Accept()` method of the node classes [Gamma et al., 1995]. The JavaCC framework automatically applies the Visitor pattern in the source code it generates [Brunneder, 2008].

The operator tree itself is implemented as *Abstract Syntax Tree* (AST) data structure. An AST represents the syntax of some language string within the nodes of a tree, whereby each node denotes a construct of the language [Reilles, 2007]. By combining the Visitor pattern with the AST structure we cleanly separate the implementation of the parser from the definition of the underlying query language. Thus, neither syntax changes nor the introduction of new operators affect the existing implementation. Consequently, the effort necessary to support language extensions is kept minimal [Brunneder, 2008].

Finally, the processor component employs the *Iterator model* for the implementation of the Fact and Dimension Algebra transformation operators. According to the Iterator model, every operator class processes a single tuple instead of a data block per call. Moreover, every operator class works in isolation; this means that it must not communicate with other operator classes nor the overall query plan to compute its result [Graefe and McKenna, 1993]. The Iterator pattern offers two advantages. First, it allows to parallelize the traversal of the operators in the tree nodes. Second, the set of operators available in the underlying query language and algebra can easily be extended [Rossgatterer, 2008].

In order to maximize FedDW's compatibility with common databases, its internal data model complies to CWM, the Common Warehouse Metamodel [(OMG), 2003a, Medina and Trujillo, 2002]. Our FedDW prototype supports the Oracle and Microsoft SQL Server 2005 database systems. Both the parser and processor components are implemented in Java. Therefore, FedDW runs on multiple platforms.

9.5 Experimental Results

From the practical viewpoint, the viability of the FedDW tool suite proposed in the previous Sections of this Chapter mainly depends on the seamless interoperability of the Global Schema Architect and Query Tool. Therefore, the prototypes of the FedDW tool suite were tested using the "Red and Blue" example. As explained in Section 9.1, the interface between GSA and the Query Tool is the *Meta-data Dictionary.* The Global Schema Architect is responsible for generating the meta-data in the dictionary, following the model-driven approach. Besides the basic connection data, the Meta-data Dictionary also contains the complete *schema catalogue* of all autonomous Data Marts participating in the Federated DW system, which is accessed by the Query Tool when processing SQL-MDi statements together with SQL OLAP queries.

The prototypical implementation of the FedDW tool suite emphasized the *proof of concepts* proposed in the thesis. Additionally, several non-functional requirements—e.g. best possible usability, use of open standards, clear and extensible interfaces, and so forth—have been pursued (as explained in [Maislinger, 2009, Rossgatterer, 2008, Brunneder, 2008]). Thus, the tests performed with the "Red and Blue" example schemas and data have aimed at verifying the seamless integration between GSA and Query Tool. In contrast, the detailed examination of query performance using the GSA Query Tool has been left to future work.

The tests performed with FedDW Global Schema Architect followed the *evolutionary prototyping* approach. That is, during the integration test phase we implemented several revisions of GSA to improve its usability, based on the test experience. Our tests focused on proving the viability of the model-driven approach for the mapping editor, whereas UML is already known to provide an illustrative notation of multi-dimensional data models (e.g., see [Luján-Mora et al., 2006, Prat et al., 2006]). The tests conducted with GSA have shown good usability of the model-driven approach for modelling semantic mappings among autonomous Data Marts. The schema editor and mapping editors of GSA hide most details of the internal representation—see the GSA Meta-Model in Section 9.2—which otherwise the users would be responsible for. In particular, populating the Meta-data Dictionary with the correct meta-data—corresponding exactly to the semantic mappings—would be too tedious when done manually. A detailed test report of FedDW GSA is presented in [Maislinger, 2009] (note: report and test data available in German only).

As far as FedDW Query Tool is concerned, it is well known that the implementation of parsers is quite easy to automatize [Brunneder, 2008]. For that reason, the underlying query processing algorithm is considerably more interesting from both, the theoretical and practical viewpoints. After parsing the SQL-MDi statement, the query processor of FedDW Query Tool reads the Meta-data Dictionary to load the local fact data and instantiate the virtual global cube. Afterwards it executes the OLAP query against the extension of the virtual global cube, and presents the query results. The parser component generates an operator tree from the SQL-MDi statement, optimizing the order of operators in the tree, such to reduce the intermediate results size as early as possible [Brunneder, 2008]. In turn, the SQL-MDi query processor component of Query Tool uses the operator tree directly as query plan. Since the prototype implementation of the FedDW tool suite concentrated on the general proof of concepts, the generation of more sophisticated query plans, as well as the further optimization of query processing, is the subject of future work.

Thus, the evaluation of the FedDW Query Tool concentrated on its interoperability with the Global Schema Architect. The Red and Blue example Data Marts have been integrated successfully with GSA, and subsequently queried with FedDW Query Tool using the automatically generated SQL-MDi statement [Maislinger, 2009]. Moreover, the tests conducted with Query Tool have shown that the concept of a high level, implementation independent conversion language such as SQL-MDi is powerful. Detailed results of stand alone tests with Query Tool, using the Red and Blue example are reported in [Rossgatterer, 2008] (note: report and test data available in German only).

The latest builds of the FedDW tool suite prototypes (GSA, query tool) together with the Red and Blue example scenario and installation instructions are available for download on http://www.dke.jku.at/staff/sberger.html. Notice that both, the schema and data of the Red and Blue example Data Marts are available only in German. The prototypes of Global Schema Architect and Query Tool require version 1.6 or higher of the Java Runtime Environment. Additionally, GSA only runs as plug-in within the Eclipse rich-client platform, whereas Query Tool is a stand-alone Java application.

Chapter 10

Conclusions

Contents

This Chapter summarizes the concepts of the FedDW approach and the contributions presented in this thesis. Moreover, the most challenging questions to be investigated by future research are highlighted.

10.1 Summary of the FedDW Approach

In this thesis, we have described a comprehensive approach—named "FedDW"—for the integration of autonomous, multi-dimensional data sources into a Federated Data Warehouse system. The approach is motivated by the clear trend towards business integration in the modern economy. In turn, business integration often entails the integration of preexisting Data Warehouses and Data Marts (among other operational data sources). As far as possible, the FedDW approach uses well established technology from previous approaches in the field of databases and Data Warehousing. The analysis of related work, however, revealed a clear mismatch between current design tools and integration frameworks for Federated DWs, and the requirements for Federated DW systems.

Based on the current State of the Art, we formulated the following requirements for Federated Data Warehouse systems that allow to integrate several, autonomous multi-dimensional Data Marts (see Chapter 3):

R 1. Extended definition of multi-dimensional heterogeneity.

R 2. Tightly coupled architecture with a stable, global schema.

R 3. Methodology for conjoint integration of dimensions and facts.

R 4. High-level conversion language as mapping mechanism.

R 5. Visual tools for the business analysts with graphical user interfaces and model-driven code generation of semantic mappings.

In the following sections, we summarize the main contributions of the FedDW approach, and explain how it addresses these requirements.

10.1.1 Multi-dimensional Conflict Taxonomy

The FedDW conceptual multi-dimensional data model (Chapter 4, pp. 51) lays the formal foundation for the systematic classification of heterogeneity in multi-dimensional systems. The data model defines the typical concepts *data mart*, *cube*, *measure*, *dimension*, *hierarchy*, *level*, *roll-up attribute*, *non-dimensional attribute*, as well as the dimension functions *members* and *level* at the conceptual level, i.e. independent of any implementation issues. Thus, the FedDW conceptual model provides a generic formalism for representing the properties of multi-dimensional data sources. Although the previously proposed Unified Multi-dimensional Meta-model [Prat et al., 2006] is similarly expressive, it does not consider the Common Warehouse Metamodel (CWM) standard [Poole, 2003, Poole and Mellor, 2001, (OMG), 2003a]. In contrast, the GSA Meta Model (GMM) of the FedDW approach embeds the conceptual multi-dimensional data model to the CWM and UML standards, as explained in Chapter 9.

To meet requirement R1, we introduced a novel classification of heterogeneities in the multi-dimensional model. Chapter 5 of this thesis analyzed in depth a taxonomy of heterogeneities in five categories, defined along the two dimensions *modelling scope* (schema – instance) and *model entity* (dimension – fact), plus the "cross-categorial" schema versus instance category:

- Schema–instance conflicts: representation of dimensional context

- Dimension schemas: attribute names, diverse hierarchies and/or levels, attribute names, etc.
- Cube schemas: attribute names, dimensionality, measure domains, etc.
- Instances of dimensions (members): roll-up functions, non-dimensional values, etc.
- Instances of cubes (cells): overlapping cube extensions

10.1.2 Federated DW Reference Architecture

To address requirement R2, Chapter 6 introduced the reference architecture for Federated DW systems, which is based on the "classical", general five-level architecture for federated databases [Sheth and Larson, 1990]. This reference architecture provides four-tiered multi-dimensional schemas (component schema, export schema, import schema, application schema). The separation of schemas to several layers supports the integration of several Data Marts, while these retain full autonomy for local schema and data management.

Moreover, the reference architecture also defines a component architecture. Compared to previous approaches in the field of multi-dimensional source integration, we proposed to add (i) a stable, global schema, (ii) the Meta-data Dictionary, and (iii) the Dimension Repository to Federated DW systems. The stable global schema in combination with source-to-target mappings allows to combine the advantage of the well-known global-as-view and local-as-view approaches to data integration [Lenzerini, 2002]. The Meta-data Dictionary stores the catalogue of global schema and local Data Marts meta-data. Finally, the dimension repository stores consolidated dimension schemas and mirrors their members, which improves the performance of query processing among the autonomous Data Marts.

10.1.3 FedDW Integration Methodology

In Chapter 7, this thesis proposed a general methodology for the conjoint integration of dimensions and facts among multi-dimensional data sources (requirement R3). The integration methodology systematically addresses the categories of heterogeneity classified in Chapter 5. Its general idea is to produce source-to-target semantic mappings (i.e., from the autonomous Data Marts to a stable, global schema), such as demanded in the reference architecture.

We introduced the notion of homogeneous facts and dimensions as the desirable property of import schemas, produced by a set of semantic mappings among autonomous Data Marts. Homogeneity is achieved in the proposed integration methodology by a quite restrictive, *minimum use* strategy between autonomous, multi-dimensional schemas and extensions. Internally, the conversion operators of the Dimension Algebra and Fact Algebra provide the "building blocks" of these semantic mappings. Both the Dimension Algebra and Fact Algebra define transformations on the properties of multi-dimensional schemas and data, allowing to resolve the various classes of heterogeneity.

10.1.4 Conversion Language SQL-MDi

To address requirement R4, we introduced the novel language SQL-MDi in Chapter 8 of this thesis. SQL-MDi allows the integration of logical, relational Data Mart schemas (i.e., of ROLAP Data Marts), providing high-level conversion clauses for the heterogeneities we analyzed in Chapter 5. An SQL-MDi statement uses three main clauses for defining the global schema and semantic matches: (1) Define [Global] Cube, (2) Merge Dimensions, and (3) Merge Cubes. Thus, SQL-MDi is suited for both, the ad-hoc integration of multi-dimensional Data Marts, and the permanent definition of semantic mappings in the Federated DW reference architecture.

The conversion clauses available in SQL-MDi correspond exactly to the operators of the Dimension/Fact Algebra. Thus, the Dimension Algebra and Fact Algebra represent one possible implementation of the SQL-MDi clauses, that is more procedural than declarative in nature. We defined the syntax of SQL-MDi precisely, and illustrated its use with many examples in Chapter 8, showing solutions for every class of heterogeneity identified in the taxonomy of Chapter 5.

10.1.5 Prototypes of FedDW Tool Suite

Our approach meets the final requirement R5 by presenting the prototype implementation of the FedDW tool suite—comprising the Global Schema Architect and Query Tool—in Chapter 9 of this thesis. The motivation behind these two tools is to facilitate the business analysts' tasks. Ideally, the users want to integrate and query autonomous Data Marts without having to write SQL-MDi code of the semantic mappings by hand.

Therefore, FedDW Global Schema Architect demonstrates that the model-driven architecture is well suited for the Data Warehousing domain. GSA comprises two main tasks of Data Mart integration: global schema design, and mapping design. To support the design of a global multi-dimensional schema, and to represent the import schemas of autonomous Data Marts, GSA uses an easy-to-comprehend, UML-based notation. The GSA mapping editors, in turn, provide context-sensitive access to predefined conversion operators for dimension and facts. Since the predefined operators available in the mapping editors correspond to the Dimension/Fact Algebra operators, GSA ultimately allows to generate SQL-MDi statements from the modelled mappings. Thus, according to Model-Driven Architecture terminology, the mappings modelled with GSA correspond to a platform-specific model, which allows for automatic code generation from the model. Moreover, GSA populates the Meta-data Dictionary of the Federated DW reference architecture with the exact information on schema elements modelled with UML schema diagrams using GSA.

FedDW Query Tool, in turn, acts as the mediator of the Federated DW reference architecture. It takes an SQL-MDi statement and SQL OLAP query as input, from which it both instantiates the virtual global cube—conforming to the global, multi-dimensional schema—and evaluates the query result. The current Query Tool prototype focussed on demonstrating that it seamlessly integrates with GSA. We proved that Query Tool can process queries across autonomous Data Marts successfully, using the SQL-MDi statements with the Meta-data Dictionary designed and created with the Global Schema Architect. Thus, the

FedDW prototypes showed how the concepts proposed in the thesis can be implemented using open standards and off-the-shelf technology, such as the Eclipse rich-client platform, and popular commercial database systems.

10.2 Future Work

While this thesis shows how to realize model-driven integration of autonomous Data Marts, it opens further interesting research questions. The most challenging research questions to be investigated by future work are the following:

- **Optimized query plans:**
  The current query processing algorithm implemented in the FedDW Query Tool prototype simply combines algebraic optimization of Dimension/Fact Algebra operators with data shipping (see Chapter 9). Basically, the operator tree generated by the query parser remains unchanged, except that the Dimension Repository is checked for local copies of the dimensions used in the query. This approach has been reasonable for proving the concepts proposed in this thesis, but it is also only elementary. More efficient query plans can be generated when taking into account the selections performed in the WHERE clauses of the SQL OLAP query.
- **Query Rewriting:**
  Even better query performance over federations of autonomous Data Marts can be expected by implementing query rewriting algorithms within the Query Tool. This would mean extending the mediator functionality of the Query Tool. Instead of transferring detail data—which is quite ineffective—this approach would allow for shipping of aggregated fact data. Such an approach requires an investigation of possible inference mechanisms over the SQL-MDi statement and the SQL query, capable of disseminating the original OLAP query and replacing it with appropriate partial queries against the sources.
- **Caching mechanisms for fact data:**
  Another interesting question to investigate is how fact data of autonomous Data Marts can be efficiently cached at the federation layer. The motivation for such an approach is similar to the motivation for the Dimension Repository, but the potential performance gains are clearly more appealing than for dimension data. Local copies of fact data would tremendously improve the performance of queries against federated autonomous Data Marts. The challenges to be solved roughly correspond to the view refreshment problem, including questions such as the appropriate detail level for cached fact data, or incremental maintenance algorithm for the "fact repository" of the Federated DW system.
- **Performance studies:**
  This thesis has emphasized the proof of concepts, based on open standards and off-the-shelf technology. Future work should also thoroughly investigate the performance of the FedDW prototype implementations. In particular, the exact overhead caused by different query processing strategies, as well as the performance of query processing with large amounts of fact data among the autonomous Data Marts would be interesting.

Bibliography

[Abelló et al., 2002] Abelló, A., Samos, J., and Saltor, F. (2002). On relationships offering new drill-across possibilities. In [Theodoratos, 2002], pages 7–13.

[Abiteboul and Duschka, 1998] Abiteboul, S. and Duschka, O. M. (1998). Complexity of answering queries using materialized views. In *PODS*, pages 254–263. ACM Press.

[Adali et al., 1996] Adali, S., Candan, K. S., Papakonstantinou, Y., and Subrahmanian, V. S. (1996). Query caching and optimization in distributed mediator systems. In Jagadish, H. V. and Mumick, I. S., editors, *SIGMOD Conference*, pages 137–148. ACM Press.

[Akinde et al., 2003] Akinde, M. O., Böhlen, M. H., Johnson, T., Lakshmanan, L. V. S., and Srivastava, D. (2003). Efficient OLAP query processing in distributed data warehouses. *Inf. Syst.*, 28(1-2):111–135.

[Akoka et al., 1999] Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., and Métais, E., editors (1999). *Conceptual Modeling - ER '99, 18th International Conference on Conceptual Modeling, Paris, France, November, 15-18, 1999, Proceedings*, volume 1728 of *Lecture Notes in Computer Science*. Springer.

[Arens et al., 1993] Arens, Y., Chee, C. Y., Hsu, C.-N., and Knoblock, C. A. (1993). Retrieving and integrating data from multiple information sources. *Int. J. Cooperative Inf. Syst.*, 2(2):127–158.

[Aslan and McLeod, 1999] Aslan, G. and McLeod, D. (1999). Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution. *VLDB J.*, 8(2):120–132.

[Atzeni et al., 2004] Atzeni, P., Chu, W. W., Lu, H., Zhou, S., and Ling, T. W., editors (2004). *Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling, Shanghai, China, November 2004, Proceedings*, volume 3288 of *Lecture Notes in Computer Science*. Springer.

[Banek et al., 2007] Banek, M., Vrdoljak, B., Tjoa, A. M., and Skocir, Z. (2007). Automating the schema matching process for heterogeneous data warehouses. In Song, I. Y., Eder, J., and Nguyen, T. M., editors, *DaWaK*, volume 4654 of *Lecture Notes in Computer Science*, pages 45–54. Springer.

[Barbancon and Miranker, 2007] Barbancon, F. and Miranker, D. P. (2007). Sphinx: Schema integration by example. *J. Intell. Inf. Syst.*, 29(2):145–184.

[Batini et al., 1986] Batini, C., Lenzerini, M., and Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364.

[Bauer and Günzel, 2006] Bauer, A. and Günzel, H. (2006). *Data Warehouse Systeme*. dpunkt Verlag, Heidelberg, Germany, 2^{nd} edition.

[Beier et al., 2003] Beier, B., Serface, L., and Wong, R. (2003). *UI Models–Master Detail Templates.* Oracle Corporation, http://oracle.com/technology/tech/blaf/specs/masterDetail_template.html.

[Berger and Schrefl, 2006] Berger, S. and Schrefl, M. (2006). Analysing multidimensional data across autonomous data warehouses. In Tjoa, A. M. and Tho, N., editors, *DaWaK*, pages 120–133.

[Berger and Schrefl, 2008] Berger, S. and Schrefl, M. (2008). From federated databases to a federated data warehouse system. *HICSS*, 0:394.

[Berger and Schrefl, 2009] Berger, S. and Schrefl, M. (2009). FedDW: A tool for querying federations of data warehouses. In *ICEIS (1)*.

[Bernardino et al., 2002] Bernardino, J., Furtado, P., and Madeira, H. (2002). DWS-AQA: A cost effective approach for very large data warehouses. In Nascimento, M. A., Özsu, M. T., and Zaïane, O. R., editors, *IDEAS*, pages 233–242. IEEE Computer Society.

[Biskup and Embley, 2003] Biskup, J. and Embley, D. W. (2003). Extracting information from heterogeneous information sources using ontologically specified target views. *Inf. Syst.*, 28(3):169–212.

[Blaschka et al., 1998] Blaschka, M., Sapia, C., Höfling, G., and Dinter, B. (1998). Finding your way through multidimensional data models. In *DEXA Workshop*, pages 198–203.

[Bonifati et al., 2001] Bonifati, A., Cattaneo, F., Ceri, S., Fuggetta, A., and Paraboschi, S. (2001). Designing data marts for data warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4):452–483.

[Bonifati and Cuzzocrea, 2007] Bonifati, A. and Cuzzocrea, A. (2007). Efficient fragmentation of large xml documents. In Wagner, R., Revell, N., and Pernul, G., editors, *DEXA*, volume 4653 of *Lecture Notes in Computer Science*, pages 539–550. Springer.

[Boyd et al., 2004] Boyd, M., Kittivoravitkul, S., Lazanitis, C., McBrien, P., and Rizopoulos, N. (2004). Automed: A BAV data integration system for heterogeneous data sources. In Persson, A. and Stirna, J., editors, *CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 82–97. Springer.

[Bravo and Bertossi, 2005] Bravo, L. and Bertossi, L. E. (2005). Deductive databases for computing certain and consistent answers from mediated data integration systems. *J. Applied Logic*, 3(1):329–367.

[Breslin, 2004] Breslin, M. (2004). Data warehousing battle of the giants: Comparing the basics of the kimball and inmon models. *Business Intelligence Journal*, 9(1):6–20.

[Brunneder, 2008] Brunneder, W. (2008). Development of an SQL-MDi query parser (in german). Master's thesis, University of Linz.

[Buitelaar et al., 2008] Buitelaar, P., Cimiano, P., Frank, A., Hartung, M., and Racioppa, S. (2008). Ontology-based information extraction and integration from heterogeneous data sources. *Int. J. Hum.-Comput. Stud.*, 66(11):759–788.

[Cabibbo et al., 2006] Cabibbo, L., Panella, I., and Torlone, R. (2006). DaWaII: a tool for the integration of autonomous data marts. In Liu, L., Reuter, A., Whang, K.-Y., and Zhang, J., editors, *ICDE*, page 158. IEEE Computer Society.

[Cabibbo and Torlone, 1998] Cabibbo, L. and Torlone, R. (1998). From a procedural to a visual query language for OLAP. In Rafanelli, M. and Jarke, M., editors, *SSDBM*, pages 74–83. IEEE Computer Society.

[Cabibbo and Torlone, 2005] Cabibbo, L. and Torlone, R. (2005). Integrating heterogeneous multidimensional databases. In Frew, J., editor, *SSDBM*, pages 205–214.

[Calvanese et al., 2001] Calvanese, D., Giacomo, G. D., Lenzerini, M., Nardi, D., and Rosati, R. (2001). Data integration in data warehousing. *Int. J. Cooperative Inf. Syst.*, 10(3):237–271.

[Carey et al., 1995] Carey, M. J., Haas, L. M., Schwarz, P. M., Arya, M., Cody, W. F., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., II, J. T., Williams, J. H., and Wimmers, E. L. (1995). Towards heterogeneous multimedia information systems: The garlic approach. In *RIDE-DOM*, pages 124–131.

[Chamberlin, 2002] Chamberlin, D. D. (2002). XQuery: An XML query language. *IBM Systems Journal*, 41(4):597–615.

[Chaudhuri and Dayal, 1997] Chaudhuri, S. and Dayal, U. (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74.

[Chen et al., 1993] Chen, W., Kifer, M., and Warren, D. S. (1993). HiLog: A foundation for higher-order logic programming. *J. Log. Program.*, 15(3):187–230.

[Christophides et al., 2000] Christophides, V., Cluet, S., and Siméon, J. (2000). On wrapping query languages and efficient XML integration. In Chen, W., Naughton, J. F., and Bernstein, P. A., editors, *SIGMOD Conference*, pages 141–152. ACM.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Codd et al., 1993] Codd, E. F., Codd, S. B., and Salley, C. T. (1993). Providing OLAP (on-line analytical processing) to user analysts: An IT mandate. *White Paper, Arbor Software Cooperation.*

[Dalkilic et al., 1996] Dalkilic, M. M., Jain, M., Gucht, D. V., and Mendhekar, A. (1996). Design and implementation of reflective SQL. Technical Report TR451, Indiana University Computer Science.

[Das et al., 2008] Das, G., Sarda, N. L., and Reddy, P. K., editors (2008). *Proceedings of the 14th International Conference on Management of Data, December 17-19, 2008, IIT Bombay, Mumbai, India*. Computer Society of India / Allied Publishers.

[Dhamankar et al., 2004] Dhamankar, R., Lee, Y., Doan, A., Halevy, A. Y., and Domingos, P. (2004). imap: Discovering complex mappings between database schemas. In Weikum, G., König, A. C., and Deßloch, S., editors, *SIGMOD Conference*, pages 383–394. ACM.

[Doan and Halevy, 2005] Doan, A. and Halevy, A. Y. (2005). Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94.

[Duschka et al., 2000] Duschka, O. M., Genesereth, M. R., and Levy, A. Y. (2000). Recursive query plans for data integration. *J. Log. Program.*, 43(1):49–73.

[Embley et al., 2004] Embley, D. W., Xu, L., and Ding, Y. (2004). Automatic direct and indirect schema mapping: Experiences and lessons learned. *SIGMOD Record*, 33(4):14–19.

[Evermann, 2009] Evermann, J. (2009). Theories of meaning in schema matching: An exploratory study. *Inf. Syst.*, 34(1):28–44.

[Fernández-Medina et al., 2007] Fernández-Medina, E., Trujillo, J., Villarroel, R., and Piattini, M. (2007). Developing secure data warehouses with a UML extension. *Inf. Syst.*, 32(6):826–856.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Garcia-Molina et al., 1997] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8(2):117–132.

[Genesereth et al., 1997] Genesereth, M. R., Keller, A. M., and Duschka, O. M. (1997). Infomaster: An information integration system. In [Peckham, 1997], pages 539–542.

[Gingras and Lakshmanan, 1998] Gingras, F. and Lakshmanan, L. V. S. (1998). nD-SQL: A multi-dimensional language for interoperability and OLAP. In Gupta, A., Shmueli, O., and Widom, J., editors, *VLDB*, pages 134–145. Morgan Kaufmann.

[Gingras et al., 1997] Gingras, F., Lakshmanan, L. V. S., Subramanian, I. N., Papoulis, D., and Shiri, N. (1997). Languages for multi-database interoperability. In [Peckham, 1997], pages 536–538.

[Glorio and Trujillo, 2008] Glorio, O. and Trujillo, J. (2008). An MDA approach for the development of spatial data warehouses. In [Song et al., 2008], pages 23–32.

[Golfarelli et al., 1998] Golfarelli, M., Maio, D., and Rizzi, S. (1998). The dimensional fact model: A conceptual model for data warehouses. *Int. J. Cooperative Inf. Syst.*, 7(2-3):215–247.

[Golfarelli et al., 2001] Golfarelli, M., Rizzi, S., and Vrdoljak, B. (2001). Data warehouse design from XML sources. In *DOLAP*.

[Gou and Chirkova, 2007] Gou, G. and Chirkova, R. (2007). Efficiently querying large XML data repositories: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(10):1381–1403.

[Graefe and McKenna, 1993] Graefe, G. and McKenna, W. J. (1993). The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pages 209–218. IEEE Computer Society.

[Grant et al., 1993] Grant, J., Litwin, W., Roussopoulos, N., and Sellis, T. K. (1993). Query languages for relational multidatabases. *VLDB J.*, 2(2):153–171.

[Gruhn et al., 2005] Gruhn, V., Pieper, D., and Röttgers, C. (2005). *MDA – Effektives Software Engineering mit UML 2 und Eclipse (in German)*. Springer Verlag, Berlin, Germany.

[Gupta and Mumick, 2006] Gupta, H. and Mumick, I. S. (2006). Incremental maintenance of aggregate and outerjoin expressions. *Inf. Syst.*, 31(6):435–464.

[Hakimpour and Geppert, 2001] Hakimpour, F. and Geppert, A. (2001). Resolving semantic heterogeneity in schema integration. In *FOIS*, pages 297–308.

[Halevy, 2001] Halevy, A. Y. (2001). Answering queries using views: A survey. *VLDB J.*, 10(4):270–294.

[Halevy et al., 2005] Halevy, A. Y., Ashish, N., Bitton, D., Carey, M. J., Draper, D., Pollock, J., Rosenthal, A., and Sikka, V. (2005). Enterprise information integration: successes, challenges and controversies. In Özcan, F., editor, *SIGMOD Conference*, pages 778–787. ACM.

[Halevy et al., 2006] Halevy, A. Y., Rajaraman, A., and Ordille, J. J. (2006). Data integration: The teenage years. In Dayal, U., Whang, K.-Y., Lomet, D. B., Alonso, G., Lohman, G. M., Kersten, M. L., Cha, S. K., and Kim, Y.-K., editors, *VLDB*, pages 9–16. ACM.

[Hammer and McLeod, 1993] Hammer, J. and McLeod, D. (1993). An approach to resolving semantic heterogenity in a federation of autonomous, heterogeneous database systems. *Int. J. Cooperative Inf. Syst.*, 2(1):51–83.

[Han and Kamber, 2000] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann.

[Härder et al., 2007] Härder, T., Mathis, C., and 0002, K. S. (2007). Comparison of complete and elementless native storage of XML documents. In *IDEAS*, pages 102–113. IEEE Computer Society.

[Hitz et al., 2005] Hitz, M., Kappel, G., Kapsammer, E., and Retschitzegger, W. (2005). *UML@Work: Objektorientierte Modellierung mit UML2 (in German)*. dpunkt-Verlag, Heidelberg, 3^{rd} edition.

[Hümmer et al., 2003] Hümmer, W., Bauer, A., and Harde, G. (2003). XCube: XML for data warehouses. In *DOLAP*, pages 33–40. ACM.

[Inmon, 2005] Inmon, W. (2005). *Building the Data Warehouse*. John Wiley & Sons, New York, 4^{th} edition.

[(ISO), 1992] International Organization for Standardization (ISO), (1992). *ISO/IEC 9075:1992: Information Technology – Database languages – SQL*. http://www.iso.ch/cate/d16663.html.

[Jensen et al., 2001] Jensen, M. R., Møller, T. H., and Pedersen, T. B. (2001). Specifying OLAP cubes on XML data. *J. Intell. Inf. Syst.*, 17(2-3):255–280.

[Josifovski et al., 2002] Josifovski, V., Schwarz, P. M., Haas, L. M., and Lin, E. T. (2002). Garlic: a new flavor of federated query processing for DB2. In Franklin, M. J., Moon, B., and Ailamaki, A., editors, *SIGMOD Conference*, pages 524–532. ACM.

[Jukic, 2006] Jukic, N. (2006). Modeling strategies and alternatives for data warehousing projects. *Commun. ACM*, 49(4):83–88.

[Kedad and Métais, 1999] Kedad, Z. and Métais, E. (1999). Dealing with semantic heterogeneity during data integration. In [Akoka et al., 1999], pages 325–339.

[Kim and Seo, 1991] Kim, W. and Seo, J. (1991). Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18.

[Kimball, 2002] Kimball, R. (2002). *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Datawarehouses*. John Wiley & Sons, New York, 2^{nd} edition.

[Kimball and Merz, 2000] Kimball, R. and Merz, R. (2000). *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. John Wiley & Sons, New York.

[Kleppe et al., 2003] Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional.

[Koch, 2001] Koch, C. (2001). *Data Integration against Multiple Evolving Autonomous Schemata*. PhD thesis, TU Wien, Vienna, Austria.

[Krishnamurthy et al., 1991] Krishnamurthy, R., Litwin, W., and Kent, W. (1991). Language features for interoperability of databases with schematic discrepancies. In Clifford, J. and King, R., editors, *SIGMOD Conference*, pages 40–49. ACM Press.

[Lakshmanan et al., 1997] Lakshmanan, L. V. S., Sadri, F., and Subramanian, I. N. (1997). Logic and algebraic languages for interoperability in multidatabase systems. *J. Log. Program.*, 33(2):101–149.

[Lakshmanan et al., 2001] Lakshmanan, L. V. S., Sadri, F., and Subramanian, S. N. (2001). SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Trans. Database Syst.*, 26(4):476–519.

[Larsen et al., 2009] Larsen, T. J., Niederman, F., Limayem, M., and Chan, J. (2009). The role of modelling in achieving information systems success: UML to the rescue? *Inf. Syst. J.*, 19(1):83–117.

[Lee et al., 1995] Lee, C., Chen, C.-J., and Lu, H. (1995). An aspect of query optimization in multidatabase systems (extended abstract). *SIGMOD Record*, 24(3):28–33.

[Lee et al., 2007] Lee, K. Y., Son, J. H., and Kim, M.-H. (2007). Reducing the cost of accessing relations in incremental view maintenance. *Decision Support Systems*, 43(2):512–526.

[Lenz and Shoshani, 1997] Lenz, H.-J. and Shoshani, A. (1997). Summarizability in OLAP and statistical data bases. In Ioannidis, Y. E. and Hansen, D. M., editors, *SSDBM*, pages 132–143. IEEE Computer Society.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: A theoretical perspective. In Popa, L., editor, *PODS*, pages 233–246. ACM.

[Levy et al., 1996] Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996). Querying heterogeneous information sources using source descriptions. In [Vijayaraman et al., 1996], pages 251–262.

[Litwin and Abdellatif, 1986] Litwin, W. and Abdellatif, A. (1986). Multidatabase interoperability. *IEEE Computer*, 19(12):10–18.

[Litwin et al., 1989] Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., and Vigier, P. (1989). MSQL: A multidatabase language. *Inf. Sci.*, 49(1-3):59–101.

[Litwin et al., 1990] Litwin, W., Mark, L., and Roussopoulos, N. (1990). Interoperability of multiple autonomous databases. *ACM Comput. Surv.*, 22(3):267–293.

[Luján-Mora and Trujillo, 2004] Luján-Mora, S. and Trujillo, J. (2004). Physical modeling of data warehouses using UML. In Song, I.-Y. and Davis, K. C., editors, *DOLAP*, pages 48–57. ACM.

[Luján-Mora and Trujillo, 2006] Luján-Mora, S. and Trujillo, J. (2006). Physical modeling of data warehouses using UML component and deployment diagrams: Design and implementation issues. *J. Database Manag.*, 17(2):12–42.

[Luján-Mora et al., 2006] Luján-Mora, S., Trujillo, J., and Song, I.-Y. (2006). A UML profile for multidimensional modeling in data warehouses. *Data Knowl. Eng.*, 59(3):725–769.

[Luján-Mora et al., 2004] Luján-Mora, S., Vassiliadis, P., and Trujillo, J. (2004). Data mapping diagrams for data warehouse design with UML. In [Atzeni et al., 2004], pages 191–204.

[Maislinger, 2009] Maislinger, L. (2009). Visual tool for the integration of autonomous data mart schemas (in german). Master's thesis, University of Linz.

[Mangisengi et al., 2003] Mangisengi, O., Eßmayr, W., Huber, J., and Weippl, E. (2003). XML-based OLAP query processing in a federated data warehouses. In *ICEIS (1)*, pages 71–78.

[Masermann and Vossen, 2000] Masermann, U. and Vossen, G. (2000). SISQL: Schema-independent database querying (on and off the web). In Desai, B. C., Kiyoki, Y., and Toyama, M., editors, *IDEAS*, pages 55–64. IEEE Computer Society.

[Mazón and Trujillo, 2007] Mazón, J.-N. and Trujillo, J. (2007). A model driven modernization approach for automatically deriving multidimensional models in data warehouses. In Parent, C., Schewe, K.-D., Storey, V. C., and Thalheim, B., editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 56–71. Springer.

[Mazón and Trujillo, 2008] Mazón, J.-N. and Trujillo, J. (2008). An MDA approach for the development of data warehouses. *Decision Support Systems*, 45(1):41 – 58. Data Warehousing and OLAP.

[Mazón et al., 2006] Mazón, J.-N., Trujillo, J., and Lechtenbörger, J. (2006). A set of QVT relations to assure the correctness of data warehouses by using multidimensional normal forms. In Embley, D. W., Olivé, A., and Ram, S., editors, *ER*, volume 4215 of *Lecture Notes in Computer Science*, pages 385–398. Springer.

[Mazón et al., 2007] Mazón, J.-N., Trujillo, J., and Lechtenbörger, J. (2007). Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms. *Data Knowl. Eng.*, 63(3):725–751.

[Mazón et al., 2005] Mazón, J.-N., Trujillo, J., Serrano, M. A., and Piattini, M. (2005). Applying MDA to the development of data warehouses. In Song, I.-Y. and Trujillo, J., editors, *DOLAP*, pages 57–66. ACM.

[McBrien and Poulovassilis, 2002] McBrien, P. and Poulovassilis, A. (2002). Schema evolution in heterogeneous database architectures, a schema transformation approach. In Pidduck, A. B., Mylopoulos, J., Woo, C. C., and Özsu, M. T., editors, *CAiSE*, volume 2348 of *Lecture Notes in Computer Science*, pages 484–499. Springer.

[McBrien and Poulovassilis, 2003] McBrien, P. and Poulovassilis, A. (2003). Data integration by bi-directional schema transformation rules. In Dayal, U., Ramamritham, K., and Vijayaraman, T. M., editors, *ICDE*, pages 227–238. IEEE Computer Society.

[Medina and Trujillo, 2002] Medina, E. and Trujillo, J. (2002). A standard for representing multidimensional properties: The Common Warehouse Metamodel (CWM). In Manolopoulos, Y. and Návrat, P., editors, *ADBIS*, volume 2435 of *Lecture Notes in Computer Science*, pages 232–247. Springer.

[Miller, 1995] Miller, G. A. (1995). WordNet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

[Mohania and Bhide, 2008] Mohania, M. K. and Bhide, M. (2008). New trends in information integration. In Kim, W. and Choi, H.-J., editors, *ICUIMC*, pages 74–81. ACM.

[Navathe and Elmasri, 2004] Navathe, S. B. and Elmasri, R. A. (2004). *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 4^{th} edition.

[Niemi et al., 2002] Niemi, T., Niinimäki, M., Nummenmaa, J., and Thanisch, P. (2002). Constructing an OLAP cube from distributed XML data. In [Theodoratos, 2002], pages 22–27.

[Nuseibeh and Easterbrook, 2000] Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA. ACM Press.

[(OMG), 2003a] Object Management Group (OMG), (2003a). *Common Warehouse Metamodel Specification v1.1.* http://www.omg.org/spec/CWM/1.1/PDF/.

[(OMG), 2003b] Object Management Group (OMG), (2003b). *Model Driven Architecture Specification Guide v1.0.1.* http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[(OMG), 2009] Object Management Group (OMG), (2009). *Unified Modelling Language Specification v2.2.* http://www.omg.org/spec/UML/2.2/PDF/.

[Özsu and Valduriez, 1999] Özsu, M. T. and Valduriez, P. (1999). *Principles of Distributed Database Systems, Second Edition.* Prentice-Hall.

[Papakonstantinou et al., 1995] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. (1995). Object exchange across heterogeneous information sources. In Yu, P. S. and Chen, A. L. P., editors, *ICDE*, pages 251–260. IEEE Computer Society.

[Pardillo et al., 2008] Pardillo, J., Mazón, J.-N., and Trujillo, J. (2008). Model-driven metadata for OLAP cubes from the conceptual modelling of data warehouses. In [Song et al., 2008], pages 13–22.

[Pardillo and Trujillo, 2008] Pardillo, J. and Trujillo, J. (2008). Integrated model-driven development of goal-oriented data warehouses and data marts. In Li, Q., Spaccapietra, S., Yu, E. S. K., and Olivé, A., editors, *ER*, volume 5231 of *Lecture Notes in Computer Science*, pages 426–439. Springer.

[Peckham, 1997] Peckham, J., editor (1997). *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*. ACM Press.

[Pedersen et al., 2002a] Pedersen, D., Riis, K., and Pedersen, T. B. (2002a). A powerful and SQL-compatible data model and query language for OLAP. In Zhou, X., editor, *Australasian Database Conference*, volume 5 of *CRPIT*. Australian Computer Society.

[Pedersen et al., 2002b] Pedersen, D., Riis, K., and Pedersen, T. B. (2002b). Query optimization for OLAP-XML federations. In [Theodoratos, 2002], pages 57–64.

[Pedersen et al., 2002c] Pedersen, D., Riis, K., and Pedersen, T. B. (2002c). XML-extended OLAP querying. In *SSDBM*, pages 195–206. IEEE Computer Society.

[Pendse and Creeth, 1995] Pendse, N. and Creeth, R. (1995). The FASMI test. *The OLAP Report*.

[Poole and Mellor, 2001] Poole, J. and Mellor, D. (2001). *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*. John Wiley & Sons, Inc., New York, NY, USA.

[Poole, 2003] Poole, J. M. (2003). *Common Warehouse Metamodel Developer's Guide*. John Wiley & Sons, Inc., New York, NY, USA.

[Pottinger and Halevy, 2001] Pottinger, R. and Halevy, A. Y. (2001). MiniCon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198.

[Poulovassilis and McBrien, 1998] Poulovassilis, A. and McBrien, P. (1998). A general formal framework for schema transformation. *Data Knowl. Eng.*, 28(1):47–71.

[Prakash et al., 2004] Prakash, N., Singh, Y., and Gosain, A. (2004). Informational scenarios for data warehouse requirements elicitation. In [Atzeni et al., 2004], pages 205–216.

[Prat et al., 2006] Prat, N., Akoka, J., and Comyn-Wattiau, I. (2006). A UML-based data warehouse design method. *Decision Support Systems*, 42(3):1449–1473.

[Quass et al., 1996] Quass, D., Gupta, A., Mumick, I. S., and Widom, J. (1996). Making views self-maintainable for data warehousing. In *PDIS*, pages 158–169. IEEE Computer Society.

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350.

[Reilles, 2007] Reilles, A. (2007). Canonical abstract syntax trees. *Electronic Notes in Theoretical Computer Science*, 176(4):165 – 179. Proceedings of the 6th International Workshop on Rewriting Logic and its Applications (WRLA 2006).

[Rood et al., 1999] Rood, C. M., Gucht, D. V., and Wyss, F. I. (1999). MD-SQL: A language for meta-data queries over relational databases. Technical Report TR528, Indiana University Computer Science.

[Rossgatterer, 2008] Rossgatterer, T. (2008). Query processing in a federated data warehouse system (in german). Master's thesis, University of Linz.

[Schmitt and Saake, 2005] Schmitt, I. and Saake, G. (2005). A comprehensive database schema integration method based on the theory of formal concepts. *Acta Inf.*, 41(7-8):475–524.

[Schöning, 2001] Schöning, H. (2001). Tamino - a DBMS designed for XML. In *ICDE*, pages 149–154. IEEE Computer Society.

[Schöning, 2003] Schöning, H. (2003). Tamino - a database system combining text retrieval and XML. In Blanken, H. M., Grabs, T., Schek, H.-J., Schenkel, R., and Weikum, G., editors, *Intelligent Search on XML Data*, volume 2818 of *Lecture Notes in Computer Science*, pages 77–89. Springer.

[Schwarz et al., 1999] Schwarz, K., Schmitt, I., Türker, C., Höding, M., Hildebrandt, E., Balko, S., Conrad, S., and Saake, G. (1999). Design support for database federations. In [Akoka et al., 1999], pages 445–459.

[Seligman et al., 2002] Seligman, L. J., Rosenthal, A., Lehner, P. E., and Smith, A. (2002). Data integration: Where does the time go? *IEEE Data Eng. Bull.*, 25(3):3–10.

[Sheth and Larson, 1990] Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236.

[Singh et al., 1997] Singh, M. P., Cannata, P., Huhns, M. N., Jacobs, N., Ksiezyk, T., Ong, K., Sheth, A. P., Tomlinson, C., and Woelk, D. (1997). The carnot heterogeneous database project: Implemented applications. *Distributed and Parallel Databases*, 5(2):207–225.

[Song et al., 2008] Song, I.-Y., Eder, J., and Nguyen, T. M., editors (2008). *Data Warehousing and Knowledge Discovery, 10th International Conference, DaWaK 2008, Turin, Italy, September 2-5, 2008, Proceedings*, volume 5182 of *Lecture Notes in Computer Science*. Springer.

[Srivastava et al., 1996] Srivastava, D., Dar, S., Jagadish, H. V., and Levy, A. Y. (1996). Answering queries with aggregation using views. In [Vijayaraman et al., 1996], pages 318–329.

[Telang et al., 2008a] Telang, A., Chakravarthy, S., and Huang, Y. (2008a). Information integration across heterogeneous sources: Where do we stand and how to proceed? In [Das et al., 2008], pages 186–197.

[Telang et al., 2008b] Telang, A., Chakravarthy, S., and Li, C. (2008b). Querying for information integration: How to go from an imprecise intent to a precise query? In [Das et al., 2008], pages 245–248.

[The OLAP Council, 2009] The OLAP Council (2009). http://www.olapcouncil.org/research/resrchly.htm.

[Theodoratos, 2002] Theodoratos, D., editor (2002). *DOLAP 2002, ACM Fifth International Workshop on Data Warehousing and OLAP, November 8, 2002, McLean, VA, Proceedings.* ACM.

[Thomsen, 2002] Thomsen, E. (2002). *OLAP Solutions – Building Multidimensional Information Systems.* John Wiley & Sons, 2^{nd} edition.

[Torlone, 2008] Torlone, R. (2008). Two approaches to the integration of heterogeneous data warehouses. *Distributed and Parallel Databases*, 23(1):69–97.

[Torlone and Panella, 2005] Torlone, R. and Panella, I. (2005). Design and development of a tool for integrating heterogeneous data warehouses. In Tjoa, A. M. and Trujillo, J., editors, *DaWaK*, volume 3589 of *Lecture Notes in Computer Science*, pages 105–114. Springer.

[Tseng and Chen, 2005] Tseng, F. S. and Chen, C.-W. (2005). Integrating heterogeneous data warehouses using xml technologies. *Journal of Information Science*, 31(3):209–229.

[Tzitzikas et al., 2005] Tzitzikas, Y., Spyratos, N., and Constantopoulos, P. (2005). Mediators over taxonomy-based information sources. *VLDB J.*, 14(1):112–136.

[Van den Bussche et al., 1996] Van den Bussche, J., Van Gucht, D., and Vossen, G. (1996). Reflective programming in the relational algebra. *J. Comput. Syst. Sci.*, 52(3):537–549.

[Vassiliadis and Sellis, 1999] Vassiliadis, P. and Sellis, T. K. (1999). A survey of logical models for OLAP databases. *SIGMOD Record*, 28(4):64–69.

[Vijayaraman et al., 1996] Vijayaraman, T. M., Buchmann, A. P., Mohan, C., and Sarda, N. L., editors (1996). *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India.* Morgan Kaufmann.

[Warmer and Kleppe, 1999] Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language OCL: Precise Modelling with UML.* Addison Wesley, Reading, MA, USA.

[Watson et al., 2001] Watson, H. J., Annino, D. A., Wixom, B., Avery, K. L., and Rutherford, M. (2001). Current practices in data warehousing. *IS Management*, 18(1):1–9.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49.

[Wirth, 1988] Wirth, N. (1988). *Programming in Modula-2.* Springer, 4^{th} edition.

[Wyss and Robertson, 2005] Wyss, C. M. and Robertson, E. L. (2005). Relational languages for metadata integration. *ACM Trans. Database Syst.*, 30(2):624–660.

[Xu and Embley, 2004] Xu, L. and Embley, D. W. (2004). Combining the best of global-as-view and local-as-view for data integration. In Doroshenko, A. E., Halpin, T. A., Liddle, S. W., and Mayr, H. C., editors, *ISTA*, volume 48 of *LNI*, pages 123–136. GI.

[Xu and Embley, 2006] Xu, L. and Embley, D. W. (2006). A composite approach to automating direct and indirect schema mappings. *Inf. Syst.*, 31(8):697–732.

[Yan et al., 2001] Yan, L.-L., Miller, R. J., Haas, L. M., and Fagin, R. (2001). Data-driven understanding and refinement of schema mappings. In *SIGMOD Conference*, pages 485–496.

[Yang and Larson, 1987] Yang, H. Z. and Larson, P.-Å. (1987). Query transformation for PSJ-queries. In Stocker, P. M., Kent, W., and Hammersley, P., editors, *VLDB*, pages 245–254. Morgan Kaufmann.

[Zhao and Ram, 2007] Zhao, H. and Ram, S. (2007). Combining schema and instance information for integrating heterogeneous data sources. *Data Knowl. Eng.*, 61(2):281–303.

[Zhou et al., 2007] Zhou, J., Larson, P.-Å., and Elmongui, H. G. (2007). Lazy maintenance of materialized views. In Koch, C., Gehrke, J., Garofalakis, M. N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C. Y., Ganti, V., Kanne, C.-C., Klas, W., and Neuhold, E. J., editors, *VLDB*, pages 231–242. ACM.

[Zubcoff and Trujillo, 2007] Zubcoff, J. J. and Trujillo, J. (2007). A UML 2.0 profile to design association rule mining models in the multidimensional conceptual modeling of data warehouses. *Data Knowl. Eng.*, 63(1):44–62.

Appendix

List of Figures

List of Tables

List of Definitions

List of Examples

Appendix A

Syntax Specification of SQL-MDi

This Chapter specifies the syntax of the SQL-MDi query language introduced in Chapter 8 of this thesis as formal, context-free grammar. The production rules of the grammar are formulated in the well known EBNF, the *Extended Backus–Naur Form* originally developed by Niklaus Wirth. The Chapter specifies two versions of the SQL-MDi language grammar. While version 1.1 of SQL-MDi has been implemented in the latest release of the FedDW Query Tool, version 1.2 contains some minor syntactical improvements.

As mentioned in Chapter 8, the basic structure of an SQL-MDi statement—together with the SQL OLAP query—consists of the following clauses:

```
{DEFINE [GLOBAL] CUBE   <cube-declarations>}
{MERGE DIMENSIONS       <merge-dim-subclauses>}
 MERGE CUBES            <merge-cube-subclauses>

 SELECT     <dimension attributes>, <aggregated measures>
 FROM       <fact tables>, <dimension tables>
 WHERE      <selection criteria>
 GROUP BY   <dimension attributes>
[HAVING    <group selection criteria>]
```

While Chapter 8 explains the syntax and semantics of all clauses and sub-clauses available in SQL-MDi in detail, it omits the formal specification of the language grammar for brevity. The implementation of a language parser, however, requires a formal grammar that defines unambiguously the allowed symbols of a language, and possible "sentences" (sequences) of the symbols. Therefore, this Chapter provides the syntax of SQL-MDi—originally introduced in [Berger and Schrefl, 2006]—as formal grammar.

In what follows, we formally specify the clauses of SQL-MDi as context-free grammar, formulated in the *EBNF (Extended Backus–Naur Form)* originally developed by [Wirth, 1988]. The EBNF grammar formalizes the clauses and sub-clauses of the SQL-MDi language exhaustively. For better clarity of the production rules, we enclose every non-terminal symbol between angle brackets. Terminal symbols, in turn, are all enclosed between quotation marks.

It is important to note that two different versions of SQL-MDi syntax have been used in the FedDW project, and in this thesis as well. Version 1.1 of SQL-MDi (Section A.1) has been implemented in the latest release of the FedDW Query Tool, that is discussed in Chapter 9. In turn, version 1.2 of SQL-MDi improved upon some minor syntactical details. The newer version 1.2 of the SQL-MDi grammar corresponds to the language syntax that has been introduced and explained in depth in Chapter 8 of this thesis.

A.1 SQL-MDi Syntax Version 1.1

The following Section specifies the EBNF rules of SQL-MDi version 1.1, which is implemented in the FedDW Query Tool [Brunneder, 2008].

```
<sql-mdi-query>  ::=  <define-clause>   <merge-dim-clauses>   <merge-cubes-clauses>.
```

Figure A.1: Top-level structure of an SQL-MDi statement.

In Figure A.4, the non-terminal expression *SQL DDL statement* refers to a CREATE TABLE statement as specified in the SQL standard [(ISO), 1992].

<define-clause>	::=	“DEFINE” <cube-specs>.
<cube-specs>	::=	<cube-spec> {[“,”] <cube-spec>} <global-cube-spec>.
<global-cube-spec>	::=	“GLOBAL CUBE” <node> “::” <cube-name> “AS” <global-cube-alias>.
<cube-spec>	::=	“CUBE” <node> “::” <cube-name> “AS” <cube-alias> <source-schema-defs> <source-instance-defs>.
<source-schema-defs>	::=	[“(”] <measure-imports> [“,”] <dim-imports> [[“,”] <pivot-schema>] [“)”].
<measure-imports>	::=	“MEASURE” <measure-import> {[“,”] <measure-import>}.
<measure-import>	::=	<cube-alias> “.” <measure-attr-name> [“->” <new-measure-attr-name>].
<dim-imports>	::=	<dim-import> {[“,”] <dim-import>}.
<dim-import>	::=	“DIM” <cube-alias> “.” <dim-attr-name> [“->” <new-dim-attr-name>] [<level-mappings-subclause>].
<level-mappings-subclause>	::=	[“(”] “MAP LEVELS” <cube-alias> “.” <dim-attr-name> <mapped-levels-list> [“)”].
<mapped-levels-list>	::=	[“(”] “[” <level-name> [“->” <new-level-name>] {[“,”] <level-name> [“->” <new-level-name>]}.
<pivot-schema>	::=	“PIVOT” (<pivot-split-measure-attr> | <pivot-merge-measure-attrs>).
<pivot-split-measure-attr>	::=	“MEASURE” <cube-alias> “.” <measure-attr-name> “BASED ON” <cube-alias> “.” <dim-attr-name> [<rename-context-dim-insts>].
<rename-context-dim-insts>	::=	<rename-context-dim-inst> {[“,”] <rename-context-dim-inst>}.
<rename-context-dim-inst>	::=	[“(”] “RENAME CONTEXT DIM” <cube-alias> “.” <context-dim-attr-name> [“’”] <old-value> [“’”] “>>” [“’”] <new-value> [“’”] [“)”].
<pivot-merge-measure-attrs>	::=	“MEASURES” <measure-attr-list> “INTO” <cube-alias> “.” <measure-attr-name> “USING” <cube-alias> “.” <context-dim-attr>.
<measure-attr-list>	::=	<cube-alias> “.” <measure-attr-name> {[“,”] <cube-alias> “.” <measure-attr-name>}.
<source-instance-defs>	::=	{<rollup-instr>} <measure-conversion-instrs>.
<rollup-instr>	::=	[“(”] “ROLLUP” <cube-alias> “.” <dim-attr-name> “TO LEVEL” <cube-alias> “.” <dim-attr-name> “[”<level-name>“]” [“)”].
<measure-conversion-instrs>	::=	[“(”] “CONVERT MEASURES APPLY” <measure-conversion-instr> {[“,”] <measure-conversion-instr>}.
<measure-conversion-instr>	::=	<function-name> “FOR” <cube-alias> “.” <measure-attr-name> (“DEFAULT” | “WHERE” <conditions>).
<conditions>	::=	<condition> {[“,”] <condition>}.
<condition>	::=	[“AND” | “OR”] <cube-alias> “.” [<dim-name> “.”] <dim-attr-name> <operator> (<cube-alias> “.” [<dim-name> “.”] <dim-attr-name> | [“’”] <value> [“’”]).

Figure A.2: DEFINE clause of SQL-MDi – sub-clauses.

```
<merge-dim-clauses>    ::= <merge-dim> {[“,”] <merge-dim>}.
<merge-dim>            ::= [“(”] “MERGE DIMENSIONS” <dim-list> “INTO” <dim-
                           def> [<set-operation>] {<member-rel-subclause>}
                           {<rename-instance-instrs>} {<attribute-mappings>}
                           [<attribute-conversion-instrs>] [“)”].
<dim-list>             ::= <dim-def> {[“,”] <dim-def>}.
<dim-def>              ::= (<global-cube-alias> | <cube-alias>) “.” <dim-name>
                           “AS” <dim-alias>.
<member-rel-
subclause>             ::= [“(”] “RELATE” <levels-list> <join-conditions> “USING
                           HIERARCHY OF” <dim-alias> [“)”].
<levels-list>          ::= <dim-alias> “.” <level-name> {[“,”] <dim-alias> “.”
                           <level-name>}.
<join-conditions>      ::= “WHERE” <join-condition> {“AND” <join-
                           condition>}.
<join-condition>       ::= <dim-alias> “.” <primary-key> “=” <dim-alias> “.”
                           <primary-key>.
<rename-instance-
instrs>                ::= [“(”] “RENAME” <dim-alias> “.” <dim-attr-name>
                           (<mapping-table-spec> | <rename-instance-instr> {[“,”]
                           <rename-instance-instr>}) [“)”].
<mapping-table-
spec>                  ::= “USING MAPPINGTABLE” <node> “::” <table-name>
                           “TO” <dim-alias> “.” <dim-attr-name>.
<rename-instance-
instr>                 ::= “>>” [“’”] <value> [“’”] [“WHERE” <conditions>].
<attribute-mappings>   ::= [“(”] “MATCH ATTRIBUTES” <attr-def> “IS” <attr-
                           list> [“)”].
<attr-list>            ::= <attr-def> {[“,”] <attr-def>}.
<attr-def>             ::= <dim-alias> “.” <attr-name>.
<attribute-conversion-
instrs>                ::= [“(”] “CONVERT ATTRIBUTES APPLY” <attribute-
                           conversion-instr> {[“,”] <attribute-conversion-instr>}
                           [“)”].
<attribute-conversion-
instr>                 ::= <function-name> “FOR” <dim-alias> “.” <dim-attr-
                           name> (“DEFAULT” | “WHERE” <conditions>).
```

Figure A.3: MERGE DIMENSIONS clause of SQL-MDi – sub-clauses.

<merge-cubes-clauses>	::=	“MERGE CUBES” <cube-alias-list> “INTO” <global-cube-alias> [<set-operation>] “ON” <dim-attr-list> ([<preference-subclauses>] [<aggregation-subclauses>] | [<context-dim-subclause>]) <global-mapping-subclause>.
<cube-alias-list>	::=	<cube-alias> {[“,”] <cube-alias>}.
<dim-attr-list>	::=	<local-dim-attr-id> {[“,”] <local-dim-attr-id>}.
<preference-subclauses>	::=	[“(”] <preference-subclause> {[“,”] <preference-subclause>} [“)”].
<preference-subclause>	::=	“PREFER” <cube-alias> “.” <measure-attr-name> (“DEFAULT” | “WHERE” <conditions>).
<context-dim-subclause>	::=	[“(”] “TRACKING SOURCE AS DIMENSION” <dim-name> “(” <schema-spec> “)” (“IS” [“′”] <value> [“′”] “WHERE” <source-condition> {[“,”] “IS” [“′”] <value> [“′”] “WHERE” <source-condition>} | “DEFAULT”) [“)”].
<schema-spec>	::=	*SQL DDL statement*.
<source-condition>	::=	“SOURCE()” “=” [“′”] <value> [“′”].
<aggregation-subclauses>	::=	[“(”] <aggregation-subclause> {[“,”] <aggregation-subclause>} [“)”].
<aggregation-subclause>	::=	“AGGREGATE MEASURE” [<global-cube-alias>] “.” <measure-attr-name> “IS” <aggregation-function> “OF” <local-measure-attr-id> [“WHERE” <conditions>].
<global-mapping-subclause>	::=	[“(”] <measure-mappings> [“,”] <dim-mappings> [“)”].
<measure-mappings>	::=	<measure-mapping> {[“,”] <measure-mapping>}.
<measure-mapping>	::=	“MEASURE” <global-cube-alias> “.” <measure-attr-name> [“->” <new-measure-attr-name>].
<dim-mappings>	::=	<dim-mapping> {[“,”] <dim-mapping>}.
<dim-mapping>	::=	“DIM” <global-cube-alias> “.” <dim-attr-name> [“->” <new-dim-attr-name>].

Figure A.4: MERGE CUBES clause of SQL-MDi – sub-clauses.

```
<node>                          ::=  <identifier>.
<cube-name>                     ::=  <identifier>.
<global-cube-name>              ::=  <identifier>.
<cube-alias>                    ::=  <identifier>.
<global-cube-alias>             ::=  <identifier>.
<dim-name>                      ::=  <identifier>.
<level-name>                    ::=  <identifier>.
<new-level-name>                ::=  <identifier>.
<dim-alias>                     ::=  <identifier>.
<dim-attr-name>                 ::=  <identifier>.
<new-dim-attr-name>             ::=  <identifier>.
<attr-name>                     ::=  <identifier>.
<new-attr-name>                 ::=  <identifier>.
<context-dim-attr>              ::=  <identifier>.
<context-dim-attr-name>         ::=  <identifier>.
<measure-attr-name>             ::=  <identifier>.
<new-measure-attr-name>         ::=  <identifier>.
<primary-key>                   ::=  <identifier>.
<local-measure-attr-name>       ::=  <identifier>.
<local-measure-attr-id>         ::=  <identifier>.
<local-dim-attr-id>             ::=  <identifier>.
<table-name>                    ::=  <identifier>.
```

Figure A.5: Identifiers in SQL-MDi statements.

```
<identifier>          ::=  {<letter> | <digit> | <special-sign>}.
<value>               ::=  {<letter> | <digit>}.
<old-value>           ::=  {<letter> | <digit>}.
<new-value>           ::=  {<letter> | <digit>}.


<letter>              ::=  “_” | “,” | “a”–“z” | “A”–“Z”.
<digit>               ::=  “0”–“9”.
<special-sign>        ::=  “/” | “$” | “#”.
<function>            ::=  <identifier> “(” {(<identifier> | <integer>} “)”.
<aggregation-
function>             ::=  ( “SUM” | “COUNT” | “AVG” | “MIN” | “MAX”).
<set-operation>       ::=  (“UNION” | “JOIN” | “MINUS” | “FULL OUTER JOIN”
                           | “LEFT OUTER JOIN” | “RIGHT OUTER JOIN”).
<operator>            ::=  (“<>” | “<” | “>” | “<=” | “>=”).
```

Figure A.6: Miscellaneous non-terminal symbols and terminal symbols of SQL-MDi statements.

A.2 Changes in SQL-MDi Syntax Version 1.2

Based on the testing experiences with the FedDW Query Tool prototype, we refined several clauses of the SQL-MDi grammar. While the updated Version 1.2 of the syntax is not yet implemented in the latest release of the Query Tool prototype, we explained the newest syntax in depth within the many examples presented in Chapter 8 of this thesis. In particular, the update from 1.1 to 1.2 of the SQL-MDi syntax defined the following improvements:

- Uses the '->' rename operator for matching non-dimensional attributes among dimension import schemas within the MERGE DIMENSIONS clause. This change simplified the syntax by eliminating the MATCH ATTRIBUTES sub-clause (updated rule: <attribute-mappings>, see Figure A.3). Besides, the renaming operation with the '->' operator is consistent with other renaming operations of schema elements used within SQL-MDi. Furthermore, and most importantly, to some degree the MATCH ATTRIBUTES sub-clause contradicted FedDW's hub integration paradigm (cf. Chapter 7) by requiring pairwise mappings between import schemas.
- Changed the names of both PIVOT operations (updated rules: <pivot-split-measure-attr> and <pivot-merge-measure-attrs>, see Figure A.2). In version 1.2, the keywords SPLIT and MERGE were added, respectively, to improve clarity of the resulting code. Version 1.1 of the pivot operations created sub-clauses that were too similar and thus easy to confuse. Moreover, rule <rename-context-dim-insts> was eliminated to simplify the syntax.
- Extended the ROLLUP clause of DEFINE CUBE with a sub-clause specifying the desired aggregation function (updated rule: <rollup-instr>, see Figure A.2). This additional sub-clause in SQL-MDi version 1.2 allows the user to control how the bags of cells in the imported cube are merged when decreasing the grain in one of the dimensions. Recall that this operation is needed to repair heterogeneous base level domains among cubes.
- Refined the resolution of non-dimensional value conflicts by improving the clarity of the RENAME clause of MERGE DIMENSIONS (updated rule: <rename-instance-instr>, see Figure A.3).
- Moved the <set-operation> token within the MERGE DIMENSIONS and MERGE CUBES clauses directly behind these keywords in order to improve legibility of the resulting SQL-MDi statement. In case that several of these clauses appear in a row—which happens almost certainly in every statement—the position of the <set-operation> token in version 1.1 resulted in statements that were awkward to read and counter-intuitive. For instance, in the code fragment MERGE DIMENSIONS dw1.a, dw2.a UNION MERGE DIMENSIONS dw1.b, dw2.b UNION the UNION keywords are misleading. In Version 1.2, the same code fragment is specified as follows: MERGE DIMENSIONS UNION dw1.a, dw2.a MERGE DIMENSIONS UNION dw1.b, dw2.b. Updated rules: <merge-dim> (see Figure A.3) respectively <merge-cubes-clauses> (see Figure A.4).

The following Figure A.7 shows the updated EBNF rules of the SQL-MDi grammar, Version 1.2.

<attribute-mappings>	::=	[“(”] **“ATTRIBUTE”** <**dim-alias**> **“.”** <**attr-name**> **“->”** <**new-attr-name**> [“)”].
<pivot-split-measure-attr>	::=	“**SPLIT** MEASURE” <cube-alias> “.” <measure-attr-name> “BASED ON” <cube-alias> “.” <dim-attr-name> [<rename-context-dim-insts>].
<pivot-merge-measure-attrs>	::=	“**MERGE** MEASURES” <measure-attr-list> “INTO” <cube-alias> “.” <**new**-measure-attr-name> “USING” <cube-alias> “.” <context-dim-attr>.
<rollup-instr>	::=	[“(”] “ROLLUP” <cube-alias> “.” <dim-attr-name> “TO LEVEL” <cube-alias> “.” <dim-attr-name> “[”<level-name>“]” [“)”] **“WITH”** <**aggregation-function**> **“FOR”** <**cube-alias**> **“.”** <**measure-attr-name**>.
<rename-instance-instr>	::=	**[“’”]** <**old-value**> **[“’”]** **“>>”** **[“’”]** <**new-value**> **[“’”]** [“WHERE” <conditions>].
<merge-dim>	::=	[“(”] “MERGE DIMENSIONS” **[<set-operation>]** <dim-list> “INTO” <dim-def> {<member-rel-subclause>} {<rename-instance-instrs>} {<attribute-mappings>} [<attribute-conversion-instrs>] [“)”].
<merge-cubes-clauses>	::=	“MERGE CUBES” **[<set-operation>]** <cube-alias-list> “INTO” <global-cube-alias> “ON” <dim-attr-list> ([<preference-subclauses>] [<aggregation-subclauses>] | [<context-dim-subclause>]) <global-mapping-subclause>.

Figure A.7: Updated production rules in SQL-MDi syntax Version 1.2 (changes given in bold font).

Curriculum Vitae

Stefan Berger

Research Assistant

Department of Business Informatics – Data & Knowledge Engineering
Altenberger Str. 69
A-4040 Linz, AUSTRIA

✉ berger@dke.uni-linz.ac.at
http://www.dke.jku.at/staff/sberger.html

📱 +43 650 68 30 100
☎ +43 70 2468-9518
FAX +43 70 2468-9471

Personal Information

Date of Birth	Jan 17, 1979
Place of Birth	Braunau am Inn, AUSTRIA
Home Address	Oberreikersdorf 47, A-4963 St. Peter am Hart, AUSTRIA
Nationality	Austrian
Marital Status	Engaged

Research Interests

- Data Warehousing (DW integration, design tools)
- Conceptual modelling (UML, E/R, etc.)
- Distributed OLAP
- Schema integration and data integration
- Ontologies, domain modelling
- XML Technologies

Academic Education

2004–2009 **Dr. rer. soc. oec. in Business Informatics**, *Department of Business Informatics – Data & Knowledge Engineering*, Linz, AUSTRIA.
Home: http://www.dke.jku.at

1999–2004 **Mag. rer. soc. oec. in Business Informatics**, *Johannes Kepler University of Linz*, Linz, AUSTRIA.
Home: http://www.jku.at; Curriculum: http://www.win.jku.at/diplomstudium-win-2002.html

Doctoral thesis

Title *FedDW: a Model-Driven Approach for Querying Federations of Autonomous Data Marts*

Supervisors o. Univ.-Prof. DI Dr. Michael Schrefl, a. Univ.-Prof. Dr. Josef Küng

Description The FedDW approach introduced in this thesis analyzes model-driven design of Data Mart federations. FedDW provides a global "mediated", multi-dimensional schema across the analytical data stores of several autonomous and heterogeneous Data Marts. Thus, FedDW allows strategic analysts to run Business Intelligence applications over larger repositories of data across organizational boundaries, enabling better founded business decisions.

The advantages of FedDW are manifold. First, FedDW integrates multi-dimensional data at the logical schema level while the underlying Data Marts remain autonomous. Second, the privacy of confidential or sensitive data is ensured by FedDW's conceptual architecture. Every participating organization is entitled to decide which business Data Mart(s) to disclose within the federation. Third, FedDW is system independent because it represents all multi-dimensional schemas, data and the mappings in an internal "canonical" data model. Fourth, FedDW uses source-to-target mappings from autonomous Data Marts to the federated layer. Thus, the global schema remains stable despite possible changes of local Data Mart schemas, and the federation is easier to extend.

Master thesis

Title *Zerlegung von digitalisierten Lehrbüchern für den Aufbau eines Contentpools (in German)* [zipped PDF: www.dke.jku.at/research/publications/MTE0401.zip]

Supervisors o. Univ.-Prof. DI Dr. Christian Stary, Dr. Andreas Auinger

Description Telelearning environments (i.e. software supporting learning in virtual class rooms) are getting more and more important nowadays. Users of those systems may expect electronic material supporting the learning process to be provided by the teacher, for instance. However, all material provided for learning purposes is completely static so far. What in most cases is missing within telelearning environments is the possibility, be it for the teacher or the student, to create learning materials dynamically. This diploma thesis contains a proposal for how to satisfy these demands based on telelearning state-of-the-art.

Based upon the technical concept of XML Topic Maps, the author presents a way how to create a semantics-based repository of so called "knowledge atoms" that can be created by decomposition of and metadata creation on slices of electronic documents (books, papers, scientific magazines, and so on). A knowledge atom repository, as mentioned above, will provide teachers and learners with the possibility of creating individual learning materials, tailored for their personal needs.

Publications & Talks

Peer Reviewed Conference Proceedings

2009 Stefan Berger, Michael Schrefl: *FedDW: A Tool for Querying Federations of Data Warehouses – Architecture, Use Case and Implementation.* Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009).

2008 Stefan Berger, Michael Schrefl: *From Federated Databases to a Federated Data Warehouse System.* Proceedings of the 41st Hawaii International Conference on System Sciences (HICSS 2008) – Organizational Systems and Technology Track, pp. 394. [DOI: http://doi.ieeecomputersociety.org/10.1109/HICSS.2008.178]

2006 Stefan Berger, Michael Schrefl: *Analysing Multi-dimensional Data Across Autonomous Data Warehouses.* In: A Min Tjoa, Juan C. Trujillo (eds.): Proceedings of the 8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2006), pp. 120–133. [DOI: http://dx.doi.org/10.1007/11823728_12]

Book Chapters

2009 Stefan Berger, Michael Schrefl: *Federated Data Warehouses.* To appear in: Nguyen Manh Tho (editor): Complex Data Warehousing and Knowledge Discovery for Advanced Retrieval Development: Innovative Methods and Applications, IGI Publishing, Hershey, PA, July 2009. ISBN: 978-1-60566-748-5. http://www.igi-global.com/reference/details.asp?ID=34437&v=preface

Academic Appointments

2004–now **Research and Teaching Assistant**, *University of Linz, Department of Business Informatics – Data & Knowledge Engineering*, Linz, AUSTRIA.
http://www.dke.jku.at

2002–2004 **Project Assistant – "Scholion WB+"**, *University of Linz, Department of Business Informatics – Communications Engineering*, Linz, AUSTRIA.
https://scholion.jku.at/

Teaching Experience

2007–2009	Data Warehousing, lectures (3 ECTS) and laboratory exercises (3 ECTS)
2006–2007	Data Warehousing & Data Mining, practical course (6 ECTS)
2005–2006	Data & Knowledge Engineering, practical course (6 ECTS)
2005–2006	Data & Knowledge Engineering, laboratory exercises (3 ECTS)

Development Experience

2007-2009 FedDW Global Schema Architect (project leader). See master thesis:
Maislinger, L. (2009): Eclipse-based Visual Tool for Data Mart Integration (in German). http://www.dke.jku.at/research/publications/abstracts/MT0901e.html

2006-2008 FedDW Query Tool (project leader). See following two master theses:
Rossgatterer, T. (2008): Development of a Federated Data Warehouse Query Processor (in German). http://www.dke.jku.at/research/publications/abstracts/MT0809e.html
Brunneder, W. (2008): Parsing and Transforming SQL-MDi Queries (in German). http://www.dke.jku.at/research/publications/abstracts/MT0802e.html

2002-2004 "Scholion WB+" Web-based e-learning environment (developer). University of Linz, Department of Business Informatics – Communications Engineering.

Languages

English	**Fluently**	
German	**Fluently**	*Mother Tongue*
Spanish, French	**Intermediate**	*University/School education, skill level B1/B2*
Russian	**Basics**	*University Education to skill level B1*

Computer skills

Scientific Writing	LaTeX, Microsoft Office 2007
Databases	Oracle, Microsoft SQL Server, MySQL
Query Languages	SQL, MDX
Programming Languages	Java, PL/SQL, Datalog
Web and Markup	XML, PHP, HTML, XHTML, XSD, XMLT
Administration	Apache Server, Tomcat Servlet Engine, Oracle and SQL Server databases
Java Technologies	Servlets, XML Processing, Swing, JDBC, XML Topic Maps
IDEs	Eclipse, Borland Together Architect, Borland JBuilder, IntelliJ IDEA

Services to the Profession

2006–now	Member of "Studienkommission Wirtschaftsinformatik" (curricular commission for Business Informatics) at Johannes Kepler University of Linz http://www.win.jku.at/studienkommission.html
2005–now	Occasional reviewer of DaWaK, DEXA, and WCBP Conferences

Interests

Mountains	Hiking, mountain climbing (in summer), skiing (in winter)
Chess	Club player, member of WSV/ATSV Ranshofen
Other sports	Volleyball, inline skating, table tennis, jogging
Photography	Digital photography is one of my latest hobbies

References

The following persons are familiar with my qualifications and personality:

o. Univ.-Prof. DI Dr. Michael Schrefl
Doctoral thesis supervisor
Dept. of Business Informatics – Data & Knowledge Engineering, University of Linz
Altenberger Str. 69, A-4040 Linz, AUSTRIA
☎ +43 70 2468-9480
FAX +43 70 2468-9471
✉ schrefl@dke.uni-linz.ac.at

o. Univ.-Prof. DI Dr. Christian Stary
Master thesis supervisor
Dept. of Business Informatics – Communications Engineering, University of Linz
Freistädter Str. 315, A-4040 Linz, AUSTRIA
☎ +43 70 2468-7102
✉ christian.stary@jku.at

Last Updated

May 30, 2009.