An Integrative and Uniform Model for Metadata Management in Data Warehousing Environments

Thomas Stöhr
stoehr@informatik.uni-leipzig.de

Robert Müller
Institut für Informatik
Universität Leipzig
Germany
mueller@informatik.uni-leipzig.de

Erhard Rahm
rahm@informatik.uni-leipzig.de

Abstract

Due to the increasing complexity of data warehouses, a centralized and declarative management of metadata is essential for data warehouse administration, maintenance and usage. Metadata are usually divided into *technical* and *semantic* metadata. Typically, current approaches only support subsets of these metadata types, such as data movement metadata or multidimensional metadata for OLAP. In particular, the interdependencies between technical and semantic metadata have not yet been investigated sufficiently. The representation of these interdependencies form an important prerequisite for the translation of queries formulated at the business concept level to executable queries on physical data. Therefore, we suggest a uniform and integrative model for data warehouse metadata. This model uses a uniform representation approach based on the Uniform Modeling Language (UML) to integrate technical and semantic metadata and their interdependencies.

1 Introduction

Modern data warehouse environments integrate a large number of databases, file systems, tools and applications which are typically based on different data models and structural description formats. For example, on the operational side, relational models can be found together with hierarchical models and COBOL-oriented description formats for flat files. W.r.t. OLAP, reporting and navigation applications, multidimensional models co-exist with object-relational or object-oriented models. Furthermore, the interdependencies between the different subsystems can become arbitrarily complex and therefore difficult to manage, if only represented at the source code level.

This implies the necessity of a *repository* to manage metadata, i.e. information about the structure, content and interdependencies of data warehouse components. Metadata supports developers and administrators responsible for the data warehouse. Furthermore, it can significantly support business users w.r.t. warehouse navigation and querying. Although the explicit and comprehensive representation of data warehouse metadata has been identified as essential, most commercial and research approaches only provide limited solutions ignoring important types of metadata (see below). This is one of the experiences we made in a recent evaluation of metadata tools for data warehousing. The evaluation was based on a comprehensive criteria catalogue on metadata management. In addition we carried out test installations and practical functionality tests with several leading tools.

To overcome the limitations of current approaches we propose a comprehensive repository model for managing data warehouse metadata. The intended usage of the model is the following:

Basically, it shall be accessible both for administrators or programmers *and* end users for navigation purposes. Furthermore, it shall allow an improved usage of metadata for tools, which have to access multiple components of a warehouse, and therefore need information about the structure and contents of the involved components. In particular, our goal is to support ad-hoc query tools which are formulated on semantically rich layers, and which then have to be translated into executable database queries such as SQL.

A unique feature of our scheme is its support of an uniform integration of both technical and semantic metadata. The next section discusses these two types of metadata in more detail. In Section 3, we briefly discuss related work including a classification of metadata tools, standardization efforts and research approaches. Section 4 is the main

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99)
Heidelberg, Germany, 14. - 15. 6. 1999
(S. Gatziu, M. Jeusfeld, M. Staudt, Y. Vassiliou, eds.)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/

part of the paper covering our uniform and integrative model for data warehouse metadata. We close with a discussion of our approach.

2 Technical and Semantic Metadata in Data Warehouses

Our work is based on a general three-layer data warehouse architecture shown in Figure 1. The first layer contains all systems involved in the management of operational data (*Operational Layer*). The second layer consists of the data warehouse as a centralized copy of relevant operational data (*Data Warehouse Layer*). The third layer we assume includes all tools and applications used by end-users for the purposes of, for example, navigation, data analysis, and data mining. In particular, this layer contains OLAP tools operating on multidimensional datamarts.

In the following, we discuss technical and semantic metadata and sketch the benefits of an explicit representation of metadata. The distinction between technical and semantic metadata is mainly motivated by the two different types of staff members accessing a warehouse. Technical staff members such as warehouse administrators and programmers are mainly interested in metadata on a technical implementation level, and are not interested in the business semantics of warehouse data in detail (technical metadata). Business end users, who are not familiar with warehouse description formats such as database SQL-DDL-files, are interested in understanding the business semantics of warehouse data, and therefore need semantically rich representations of the structure and contents of a warehouse (Semantic metadata). From this point of view, semantic metadata on one side form a business-oriented *view* on technical metadata. On the other side, they add business-oriented semantic information to the data which is not explicitly represented by technical metadata.

2.1 Technical Metadata

Technical (or *administrative/structural*) metadata cover information about:

- the architecture and schemata w.r.t. the operational systems, the data warehouse and the OLAP databases. This includes, for example, information about table and record structures, attribute constraints, triggers, or views defined in the different databases or file systems.
- the dependencies and mappings between the operational sources, the data warehouse and the OLAP databases[1] on the physical and implementation level. This includes all data movements filters, transformations and aggregations w.r.t. flat files and physical database tables. Mappings between the physical warehouse layer and a *logical, business-oriented view* of warehouse data belong, in our terminology, to semantic metadata (see Section 2.2).
- temporal data and data about user actions (“*What has happened when in the data warehouse*?“)

Technical metadata is usually extracted from DBMS catalogues, COBOL copy books[2], data movement tools, or CASE[3] tool schema exports. In this context, a metadata repository should support the following basic requirements: First, it should support the *collection* of technical metadata (e.g. by providing scanners for database schemata). Second, it should provide a *uniform* representation approach to store the different types of technical metadata. Third, it should be able to export them in one of the standard metadata formats such as the MDIS[4] format. Fourth, it should provide a comfortable interface for administrators and developers.

Furthermore, technical metadata repositories in the data warehouse context should also be *bi-directional*. This means that the repository should not only be able to *read* metadata such as a definition specifying a mapping between an operational source and the data warehouse. It should also allow to *redefine* the mapping within the repository, and then to propagate the changes to the executing data movement tool. However, such an interoperability must be supported by both the repository and the respective tools, e.g., for data movement.

2.2 Semantic Metadata

In contrary to technical metadata, the term *semantic metadata* (or *business metadata*) is not used in a standardized manner by researchers and companies. Basically, *semantic metadata* intend to provide a *business-oriented* description of the data warehouse content. This description should be understandable for users who are not familiar with technical data descriptions or query languages such as SQL. In the following we will refer to users who are only interested in the business content as *business users*. A repository addressing semantic metadata and therefore supporting these *business users* should cover the following types of business-oriented metadata (also see Figure 1):

- ***Conceptual enterprise model***: This important functionality of a semantic-oriented repository includes the high-level representation of an enterprise data model, its business concepts and their relationships. On the base of this enterprise model, the business user not familiar with database query languages such as SQL can inform himself, which data is provided by the data warehouse.
- ***Multidimensional data model***: This important part of a

1. In the following, we use the terms *OLAP database* and *datamart* as a synonym.

2. A COBOL copy book is a file structure descriptor used by COBOL programs to interprete flat files.
3. CASE = Computer Aided Software Engineering
4. MDIS = Metadata Interface Specification of the Metadata Coalition

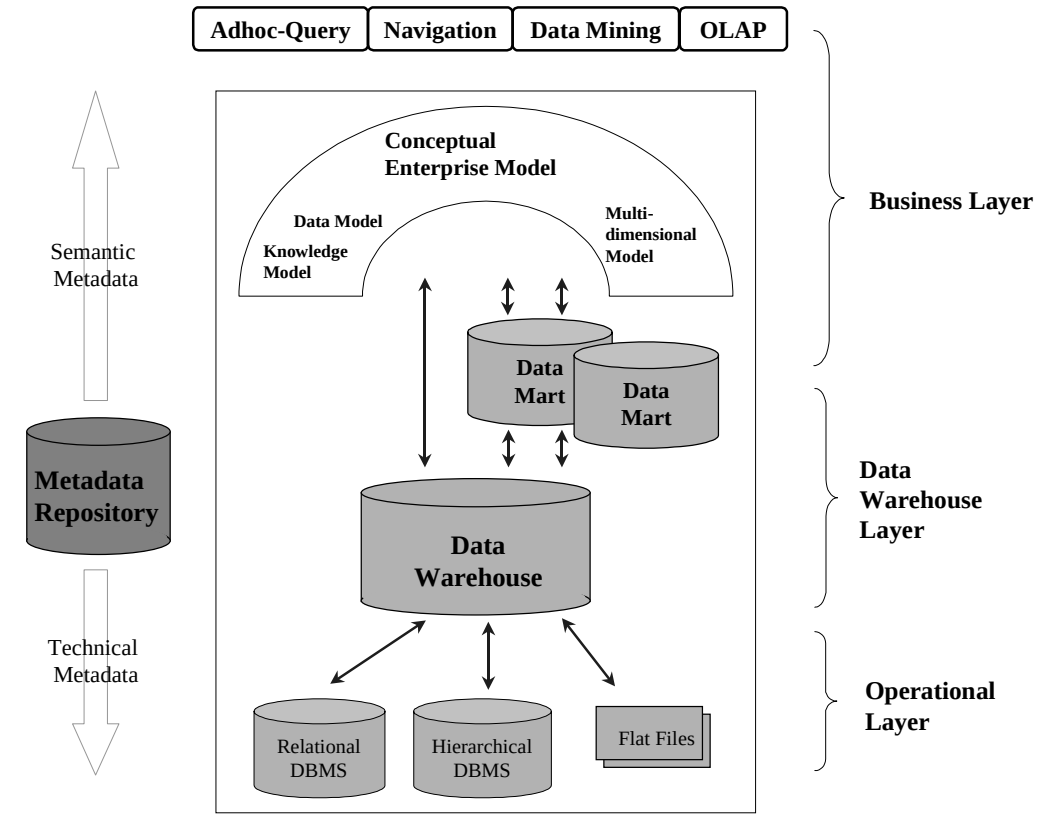

Figure 1: Metadata management in a data warehouse. The metadata repository stores and maintains information about the structure and the content of the data warehouse components. Furthermore, all dependencies between the different layers are represented in the repository.

conceptual enterprise model informs the business user about which dimensions, dimension categories, data cubes, and aggregation principles exist w.r.t. the datamarts. With the term *data cube* we denote a multidimensional organization of facts w.r.t. business concepts of the domain. In the insurance domain, for example, we may have a data cube *Claims* with the dimensions *time*, *region*, *insurance product types* and *insured-persons* and a categorization of each of these dimensions. A fact of the data cube may represent the sum of all payments w.r.t. to a time period, a particular region, a particular insurance product and an age group of insured persons.

- ***Dependencies between conceptual business models and physical data***: As the metadata types listed above only provide business-oriented *views* of the data warehouse, the dependencies between this business layer on one side and the operational and data warehouse databases on the other side also have to be represented within the repository. Let, for example, the repository contain business concepts such as *Life-Insurance*, *Insurance-Contract* and *Insurance-Claim*. Then the repository should also represent which physical base tables contain data about these business concepts. In particular, it should be represented which attributes of the base tables correspond to which attributes of the business concepts.

An explicit representation of the types of metadata listed above supports business users w.r.t. the following tasks:

- ***Navigation*** along a business-oriented view of the data collected in the data warehouse or the datamarts. For example, Figure 2 shows a navigation browser for business concepts from an insurance domain. Here the business user can inspect which contract types are offered by his enterprise.
- ***Ad-hoc Querying*** at the level of business concepts without having to know the technical details of query languages such as SQL. If the metadata repository represents the connections between the conceptual enterprise model and the physical data, the business term query can then automatically be translated to execut-

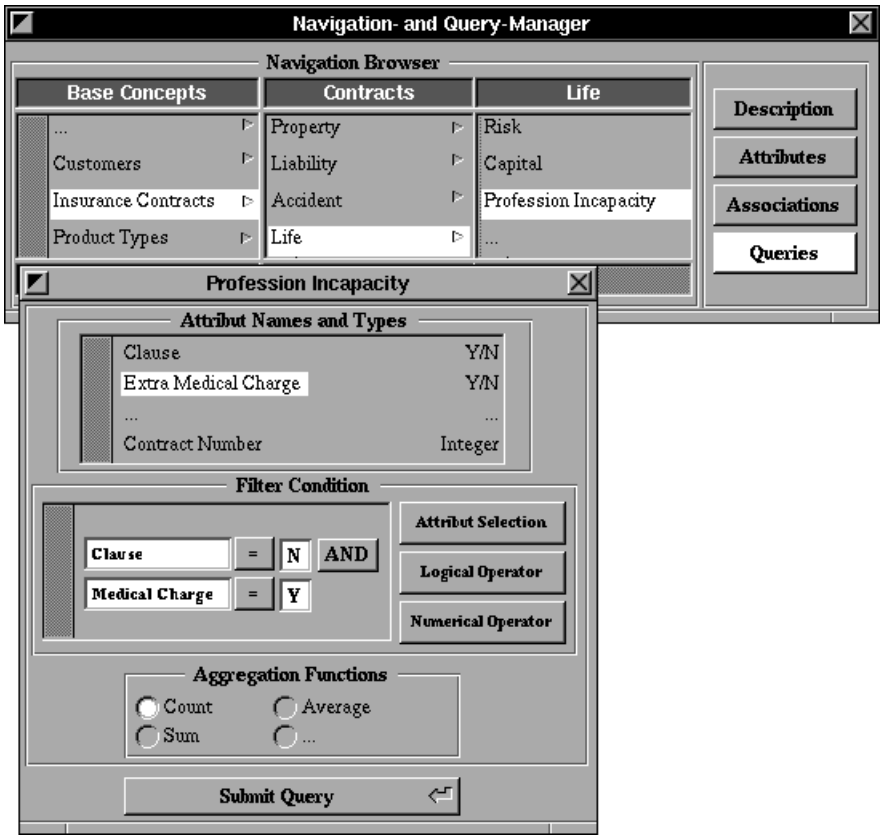

Figure 2: User interface for a business-oriented view of a data warehouse. The business concepts shown in the two windows are retrieved from a metadata repository storing semantic metadata.

able query programs accessing the data warehouse or the datamarts.
In Figure 2, for example, the user has specified an ad-hoc query asking for the number of all profession incapacity insurance contracts, where no insurance contract clause exists but an extra medical charge has to be paid.

- ***Data Mining***: As semantic metadata usually represent semantic associations and specialization hierarchies of business concepts in an explicit manner, a metadata-based hypothesis generation and result filtering can support data mining (in contrary to data mining which has to build hypotheses and filters on raw data).

3 Related Work

We briefly discuss various commercial metadata tools as well as standardization efforts for metadata exchange. In addition, we consider related research work.

3.1 Metadata-Related Tools

Tools related to the data warehouse market usually support only particular types of metadata as illustrated in Figure 3.

- One group of tools focuses on data extraction, transformation and movement (e.g. PLATINUM DECISION BASE, ARDENT DATASTAGE, ETI EXTRACT) of operational data to the data warehouse, thus offering almost no semantic capabilities. Usually, these tools provide a technical, relational view on the data warehouse. Some vendors additionally offer a warehouse creation facility. For example, INFORMATICA POWERMART supports the Data Mart Design process through the so-called multidimensional schema wizard.
- Tools for OLAP, reporting and business intelligence (e.g. MICROSTRATEGY DSS AGENT, COGNOS IMPROMPTU, BUSINESS OBJECTS) provide a multidimensional view through mapping relational tables to business-relevant dimensions and facts, to support multidimensional analysis (drill-down, roll-up etc.) on warehouse data.
- To offer a more business-oriented point of view to non-technical users, many CASE or modeling tools exist to describe the business world at a higher semantic level than relational tables do. Usually, their modeling com-

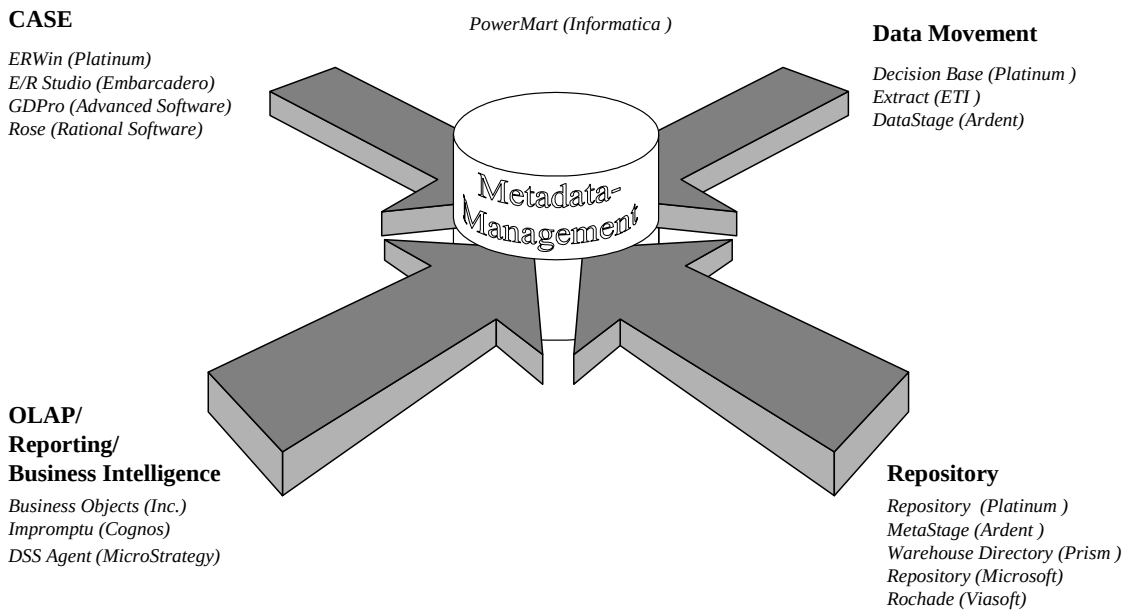

Figure 3: An overview of tools on the market, which provide metadata in data warehouse environments. POWERMART is located between the CASE and the Data Movement section, as it also provides a warehouse specification and creation facility.

plexity covers the E/R model or object-oriented approaches like UML introducing modeling capabilities like different association types or methods. Representatives of such tools include ERWIN (PLATINUM), E/R STUDIO (EMBARCADERO), GDPRO (ADVANCED SOFTWARE) and ROSE (RATIONAL SOFTWARE).

- The tools mentioned above usually store metadata in a relational or object-oriented database, which is often handled as a black box. Separate repositories are provided by MICROSOFT (REPOSITORY), PLATINUM (REPOSITORY), VIASOFT (ROCHADE) or ARDENT (METASTAGE).

However, all these different types of tools only support an isolated subset of metadata. In particular, no commercial solution exists that integrates metadata for technical data movement, multidimensional analysis and semantic modeling in a data warehouse within one tool. For example, the Repository Information Model (RIM) of ROCHADE covers a broad range of business and multidimensional as well as technical metadata. However, the representation of transformation rules on the technical layer and between technical and semantic metadata does not become clear [Wie98].

3.2 Standardization Efforts

There are several efforts to standardize metadata exchange. The objective of the Metadata Coalition is to create a standard API for metadata, the Metadata Interchange Standard (MDIS) [MC98]. This approach models schema information of different types of data stores, like relational, multidimensional, object-oriented, net or hierarchical database systems as well as file structures. MDIS is not data warehouse-specific and it is limited to schema relationships. It does not cover semantic metadata and offers only low support for data movement purposes.

MICROSOFT'S REPOSITORY [BB+99] offers an UML-oriented metadata interface and export capabilities to exchange metadata between repositories (by XIF[5]). Several vendors addressing the data warehouse sector support XIF (e.g. NCR, PLATINUM, SOFTLAB, SYBASE, UNISYS and VIASOFT).

A further approach to exchange a wide range of metadata is the XML Metadata Interchange format (XMI) of the Object Management Group (OMG). Its objective is to exchange programming data of developers working with object technology over the internet. However, it is not data warehouse-oriented.

3.3 Related Research Work

Similar to the commercial market, most research approaches are limited to metadata subsets. Bernstein et al. [BB99] describe a metadata-driven data transformation approach in the data warehouse context, which is based on the MICROSOFT REPOSITORY [BB+99]. Katic et al.

5. XIF is the proprietary XML interchange format of MICROSOFT.

[KQS+98] describe a metadata approach for data warehouse security, but do not go beyond technical metadata plus business-oriented string labels and descriptions of attribute and table names.

Golfarelli et al. [GMR98] and Wietek [Wie99] describe detailed models of multidimensional data which could serve as a base for business-oriented views of OLAP data. However, they do not explicitly address metadata for the interdependencies between multidimensional structures and business concepts spaces which are not organized in a multidimensional manner.

Within the METAFIS approach, Becker and Holten describe a elaborate semantic metadata model for conceptual data models [BH98][Hol99]. Technical metadata and especially the interdependencies between technical and semantic metadata are not addressed in this approach.

In the context of the DATA WAREHOUSE QUALITY (DWQ) project, Calvanese et al. [CGL+98] and Jeusfeld et al. [JQJ98] provide a comprehensive metadata model in the context of measuring the quality of data warehouse components. The approach covers a broad range of technical and semantic metadata. It is based on the CONCEPTBASE database [JGJ+95] and uses description logics as the underlying representation approach. Although formally very strong, the usage of logics makes it more difficult to practically apply it in specific data warehouse environments.

4 A Uniform and Integrative Model for Data Warehouse Metadata

In the following, we introduce our model of technical and semantic metadata. We call the model *uniform*, as it uses the same representation approach both for technical and semantic metadata. The representation language we use is the *Unified Modeling Language* (UML) [FS98]. We call it *integrative*, as the UML schema provides several generic classes shared both by the technical and semantic metadata model. Aspects specific for technical or semantic metadata are realized mainly by subclassing these shared classes.

We use UML class diagrams to represent our meta model. Classes are noted as grey rectangles, with the name of the class in the upper part of the rectangle (see, for example, Figure 4). Attributes are denoted in the bottom part of the class rectangle. Inheritance relationships between classes are specified via arcs without labels. For example, in Figure 4, *Aggregation* and *Non-Aggregation* are subclasses of *Transformation*. Associations between classes are noted as arcs labeled with the name of the association and the multiplicity information specifying how many instances of a class can participate within the association. Composition associations are described by a black rhombus symbol. In the example of Figure 4 it is specified, that a *Mapping* instance may use an arbitrary number of *Transformation* instances (association *uses-transformations*). Because of the composition semantics, all transformations of the mapping are deleted, if the mapping is deleted.

4.1 Shared Classes

In this subsection, we briefly describe the classes used both by the technical and semantic model (see Figure 4).

- ***Entity, Attribute*** and ***Association***: The central root class is *Entity*. Instances of this class may represent a relational table definition, a record type of a hierarchical database system, an object-oriented class, or a semantic business concept. The structure of an entity is described by a number of *Attribute* instances. Furthermore, an entity may participate within an association to itself or to one or more other entities. This is represented by an *Association* instance which connects *Entity* instances via the UML association *connects*. This UML association is attributed with the *role* of the connected *Entity* within the represented association (*source* or *target* in the case that it is directed). Furthermore, information about the multiplicity w.r.t. the entities connected by the association is represented by *lower-boundary* and *upper-boundary*. The classes *Entity*, *Attribute* and *Association* basically cover the meta model of the *Entity/Relationship* model minus the construct of weak entities. The association types *composition* and *inheritance* we need for instances of *Entity* are introduced in the semantic meta model in section 4.3, as they are not needed in the technical model.

- ***Mapping, Transformation*** and ***Aggregation***: Furthermore, our UML schema provides the generic classes *Mapping*, *Transformation*, and *Aggregation* to express structural dependencies and data dependencies. A *Mapping* instance describes structural dependencies at the level of *Entity* instances, while *Transformation* instances operate at the level of *Attribute* instances. For example, a *Mapping* instance may express that a hierarchical record is mapped to a relational table during the data movement process from an operational source to the data warehouse. A *Mapping* instance may also express that a particular business concept such as *Insurance-Claim* corresponds to two data warehouse tables which store claim event data. Such a mapping between semantic concepts and physical data can be used, for example, to translate ad-hoc queries at the semantic level to SQL queries performed on the appropriate database tables. A *Mapping* instance may use an arbitrary number of *Transformation* instances to express mapping dependencies at the attribute level. i.e. which source attributes are transformed to which target attributes, and which transformation function is used (see section 4.2 for details w.r.t. data movement metadata). Instances of the class *Aggregation*, which is a subclass of *Transformation*, represent transformations where a set of values is aggregated to a single value. Again, this class can be used both in the context of technical metadata and semantic metadata: On the one side, a set of *Aggregation* instances may express, for example, the mappings and transformations when

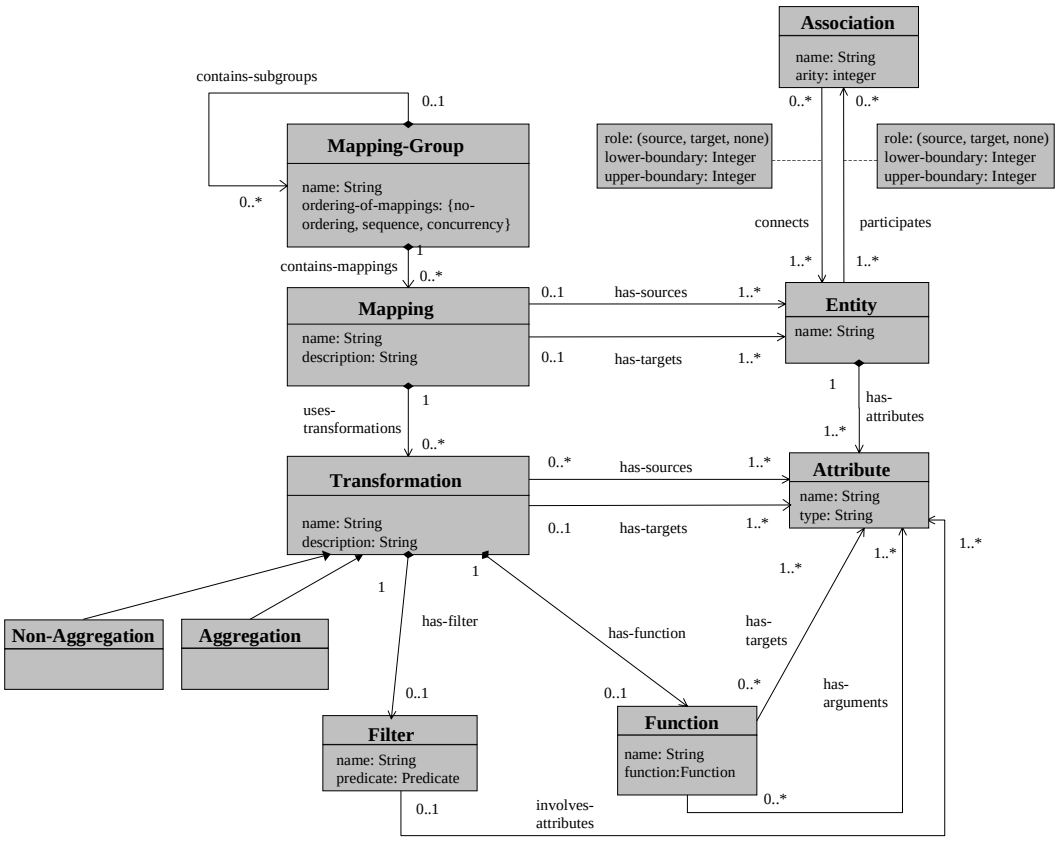

Figure 4: UML schema for classes shared by the technical and semantic metadata model. This schema covers, for example, the representation of entities, associations between entities, and mappings or transformations between entities (see text for details).

data warehouse data are aggregated to the star schema tables of a multidimensional database. On the other side, they can also be used at the business level to express, for example, how instances of the business concept *Insurance-Claim* are aggregated to the sum of all claim payments within a particular time period, region and insurance product type. *Non-Aggregation* instances cover transformations where, for example, a *„tuple-by-tuple“* transformation has to be processed. This is, for example, needed when the VAT has to be added to all tuples storing a charge value.

In the following we now describe the technical and semantic metadata model in detail. In particular, we introduce subclasses of the above generic classes specific for technical or semantic metadata.

4.2 UML-Schema for Technical Metadata

In the data warehouse context, our model covers the following two types of technical metadata:

- First, the description of the data structures in operational data stores, the data warehouse and datamarts and their intra-system dependencies. We cover functional dependencies like relational foreign key relationships, or the dependencies within a hierarchical database. Usually, all described kinds of metadata are imported from data dictionaries or COBOL copy books.
- Second, our model provides a description of inter-system dependencies. This mainly covers the complex structures of mappings between operational data sources to a data warehouse. This kind of metadata supports administrators during the design and the processing of *data movement*, when a high amount of data from heterogeneous operational systems has to be cleansed, transformed and stored into the data warehouse. It should be pointed out that data warehouse management tasks such as the periodical warehouse update shall still remain in the responsibility of specialized tools such as data movement tools. Therefore, the repository primarily serves as a centralized viewer of system inter-dependencies.

4.2.1 Schema Representation

Figure 5 shows how physical data is organized in the data

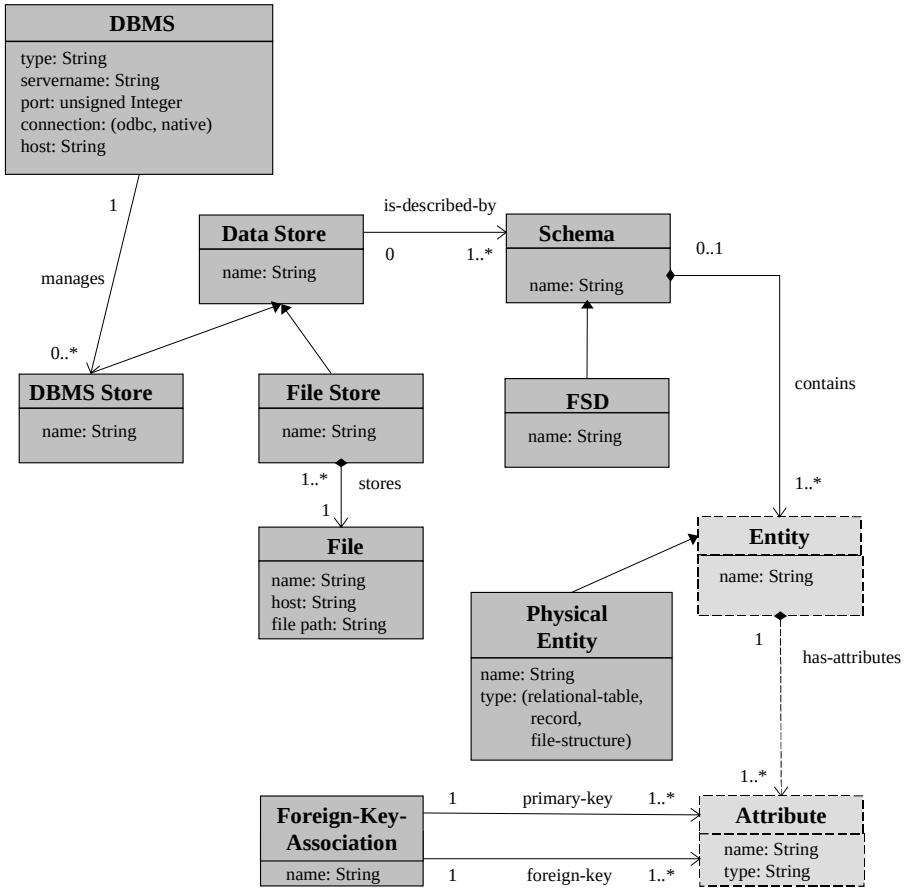

Figure 5: UML schema for physical data organization in a data warehouse (see text for details). Instances of *FSD* (for file structure description) describe the structure of flat files. A typical FSD instance, for example, would be a COBOL copy book describing the structure of a number of files storing operational data.

warehouse. Dashed lines and rectangles denote associations and classes which already have been defined in the shared model (see Figure 4) and are now re-used in a technical manner. In terms of technical metadata, instances of the shared class *Entity* represent structure units of different data system types. For example, an *Entity* instance may represent a table structure in a relational database, a record structure in a hierarchical database or a flat file description. We introduce *Physical-Entity* to represent structures which are used to materialize data.

An *Entity* instance is described by *Attribute* instances. For example, we may have an instance of *Physical-Entity* with *Physical-Entity→name = „Person“* and *Physical-Entity→type = relational-table* which describes a person table representing all persons related to an insurance company (e.g. insured, agents, policy-holders). The attributes of the table are described by associated *Attribute* instances with *Attribute→name = „first-name“* and *Attribute→name = „birth-date“* etc.

Generally, entities are organized in a logical *Schema*. In our technical context, a schema describes a set of logically associated table structures, or a set of files associated with a file structure description such as a COBOL copy book (subclass *FSD*).

A *Data-Store* is a collection of physical data which is stored by exactly one data system type, e.g. by one physical DB2 database or a set of files described by a COBOL copy book. An instance of *Data-Store* references one or more *Schema* instances, which describe the data store. However, a *Schema* instance is not necessarily associated to a *Data-Store* instance. It can also represent a collection of structures describing non-physical data.

We introduce *DBMS-Store* and *File-Store* as different subclasses of *Data-Store* to emphasize that the access to hierarchical or relational databases is different from the access to a set of files. Instances of *DBMS-Store* are maintained by a database system, represented by an instance of the *DBMS* superclass. Thus, the necessary information to access a DBMS-controlled entity is managed by a DBMS and therefore modeled at the DBMS-level (DBMS

attributes *servername, portnumber, hostname, connection* etc.). In contrast, the meta information about file storage (*host, file-path* etc.) is a file property and has to be modeled in a special *File* class.

Foreign key relationships between physical (relational) entities can be represented by instances of the class *Foreign-Key-Association.*

4.2.2 Data Dependencies and Data Movement

In this section, we present our model to describe data dependencies as they appear in the technical data movement processes. We incorporate operational systems, data warehouse base tables, multidimensionally organized datamarts and their dependencies. All needed UML classes and associations are already denoted in our shared class figure (Figure 4).

At the technical layer, we describe all data movement-related dependencies as a set of *Mapping* instances. One instance represents a "single-stage" link between a set of source *Entity* instances and a set of target *Entity* instances where every entity plays only a single role, either source or target. The aggregating class *Mapping-Group* comprises two types of mapping collections: 1) multi-stage mappings, where several *Entity* instances serve as both a source and a target within a complex mapping, and 2) a set of otherwise logically related disjunct (single- or multi-stage) mappings.

Each *Mapping* instance consists of zero, one or more *Transformation* instances, which describe the transformation for a *Mapping* instance at the *Attribute* instance level. We understand a transformation as a combination of a filter and a function, represented by the corresponding UML classes. A *Filter* instance mainly embodies a filter predicate. We allow complex predicates, described by a set of simple predicates (each containing a single comparison) which are composed by Boolean operators. A predicate additionally can incorporate function calls, e.g.

$$attr_1 = f(attr_2)\ OR\ NOT\ attr_3$$

where *f* represents an arbitrary function, which can be imported from any function library.

A *Function* instance stores a function to describe the transformation from one ore more source attributes to one ore more target attributes:

$$(t_1, ..., t_n) \leftarrow f_1(s_1, ..., s_m); ...; f_n(s_1, ..., s_m)$$

The s_i, t_i denote the source and target attributes, respectively. The function f_i describes how to calculate t_i. and can also be imported from any function library. Therefore, our functions cover the transformation complexity of C or other 3GL programming languages.

Due to the different semantics of aggregation transformations, we have already introduced two specialized transformation classes, *Aggregation* and *Non-Aggregation,* in section 4.1. *Aggregation* is characterized by aggregation functions on instances of a single *Entity* instance, e.g. a sum of an attribute value over the tuples of a relational table. A *Non-Aggregation* instance is a transformation incorporating function types not aggregating tuples or records.

An *Aggregation,* e.g. described by a function *agg-sum(attribute),* performs a sum on every (predicate-dependent) group of tuples or record values of an *Entity* instance, resulting in a single aggregate value for each group. In the context of a *Non-Aggregation* instance, *nonagg-sum(*$attribute_1$*,* $attribute_2$*)* transforms two single values of different attributes of *one source row* (for each source row) to one single value. So, summation is used with two different meanings, depending on the particular *Transformation* subclasses.

Figure 6 provides an example for a *Mapping-Group* instance named *Damage-Group* in an insurance environment. Starting from operational relational tables *Person* and *Contracts* and a flat file *Claims* we materialize the sum of money paid to accident-insured persons in case of an insurance claim, according to their certain age group (e.g. group "children", whose age is less than 16) in a fact table of a datamart. To solve this typical data movement problem, we have to execute the following three steps, each described by different *Mapping* instances:

- ***Filter-Mapping (1)***: First, we apply a filter to the *Person* table to extract the insured (in contrast to insurance holders) and store them in a temporary table *insured.* Therefore, the *Mapping* instance *Filter-Mapping* is represented by a simple *Filter-Transformation,* just storing a corresponding predicate in the *Filter* instance.

- ***Join-Mapping (2)***: As a second step, we join *Person, Contract* and *Claim* into the table *Damage* to get all information related to a claim concerning a contract which has been made by a certain person. To provide a join, the associated *predicate* attribute of the *Filter* instance contains the join condition. The *function* instance contains a list of function descriptions, one for every target attribute. In our example, the *age* value of *Damage* is generated by differentiating *current-year()* and the year of *birth-date* of a certain *insured* instance.

- ***Aggregate-Mapping (3)***: Finally, we aggregate the individual amounts of loss, grouped by applying range predicates to build certain age groups into an *Entity* instance *Damage-Aggregation*. This part is described by *Transformation* instances *A-Transformation1, A-Trans-formation2,* which are instances of the specialized Transformation class *Aggregation* mentioned above. We compute a single value for every group of the *Damage* instance by applying a set of range predicates on its *age* attribute.

By introducing *Filter, Function* and *Aggregation* classes we achieve a mapping and data movement complexity as provided by the functionality of up-to-date object-relational query languages (e.g. INFORMIX SQL, SQL3) plus user-defined functions or methods, imple-

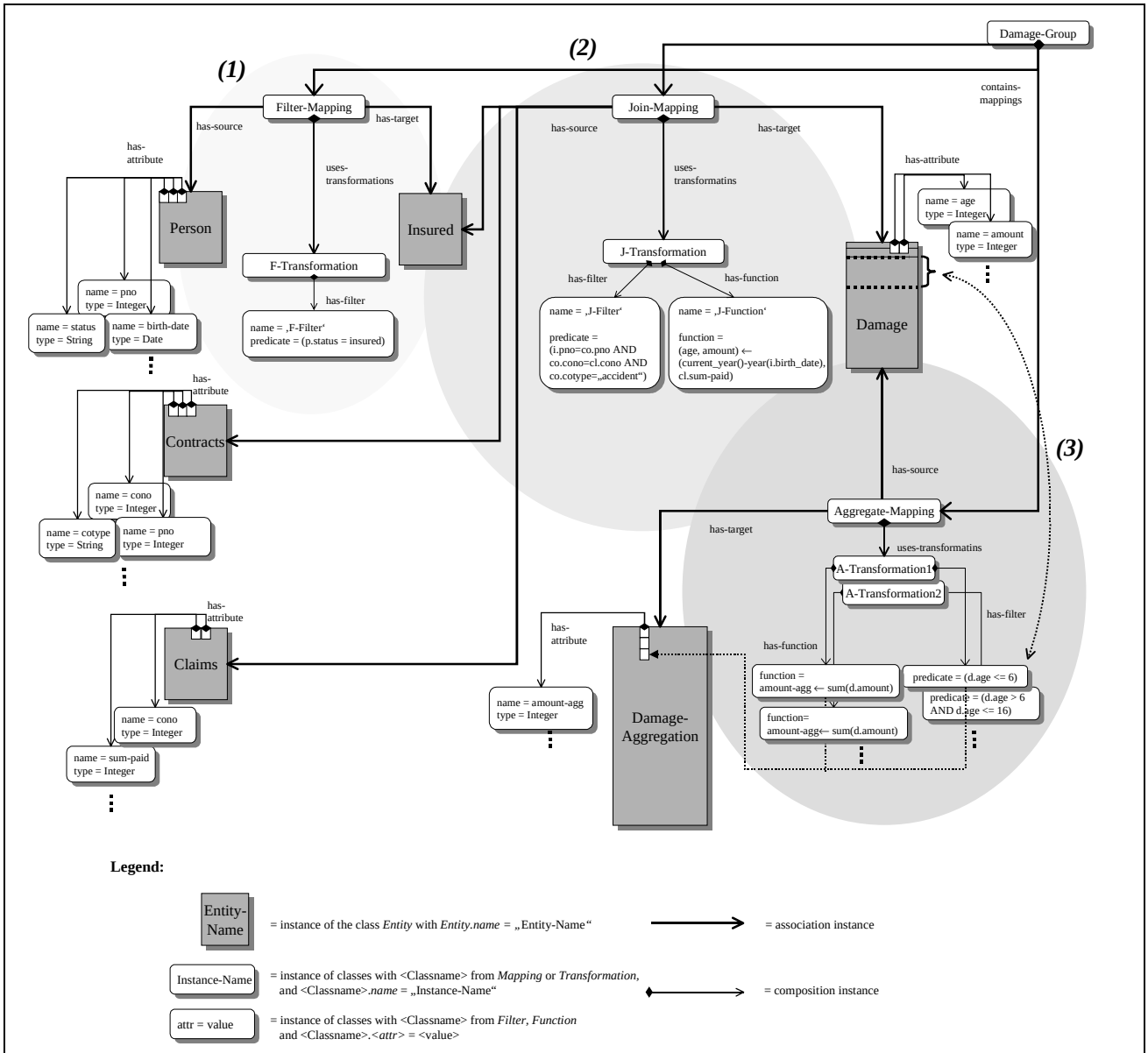

Figure 6: Instantiation of technical metadata, illustrating the representation of mappings and transformation in a data movement example (see text for details).

mented in a common 3GL language. Therefore, we gain the transformation complexity of C. To generate queries for data movement, filter predicates (incorporating functions) can be implemented as conditions in object-relational WHERE or HAVING clauses. User defined functions can either be implemented as class methods or, if externally implemented, integrated via an external library definition and then used in the SELECT or WHERE clauses.

4.3 UML-Schema for Semantic Metadata

We now describe our UML schema for business concepts and multidimensional data. This schema provides classes to represent warehouse-related metadata from the business point of view. The schema is shown in Figure 7. Furthermore, we illustrate this semantic meta data model by an example from the insurance domain (see Figure 8).

4.3.1 Business Concept Representation

The first important class of the semantic meta model in Figure is *Business-Concept*. It is a subclass of *Entity*, and therefore described by a number of *Attribute* instances. Typical *Business-Concept* instances are, for example, product types, relevant business events, or customer types.

For a more compact notation, we introduce the following convention: If we, for example, have a *Business-Concept* instance with *Business-Concept→name* = "*Insurance-Claim*", we simply say that *Insurance-Claim* is a *Business-Concept* instance, or simply a business concept. This is also reflected in the graphical notation of Figure 8, where a *Business-Concept* instance with *Business-Concept.name = "Insurance-Claim*" is noted as a shadowed rectangle with the label *Insurance-Claim*.

In the insurance domain, typical business concepts include *Property-Insurance*, *Insurance-Claim*, or *Commercial-Client*. Business concepts may be organized within multiple inheritance hierarchies, represented by the *has-subconcepts* UML association of Figure . Instantiations of this *has-subconcepts* association are noted as dashed arrows (see Figure 8). Furthermore, a business concept may form a complex concept via the *has-subparts* UML association of Figure . This association may have the semantics of an aggregation or a composition, which is described by the *Subpart-Role* class. As an example for a business concept aggregation, the insurance type *Extended-Household-Contents-Insurance (*as a sample subconcept of *Property-Insurance*) may have a *Glass-Insurance* and a *Bike-Insurance* as components. However, when the customer cancels a household insurance, we can still keep the bike insurance.

Domain-specific associations between business concepts have to be expressed as instances of the classes *Entity* and *Association* of Figure 4. For example, the association that a damage (concept *Damage*) may result in a claim (concept *Insurance-Claim*) is represented via an *Association* instance with *Association.name = „may-result-in"*. As a graphical shortcut, we use a labeled black arrow to denote such an association (see Figure 8).

4.3.2 Multidimensional Data

The right part of Figure 7 contains the UML metadata schema for multidimensional data and their associations to business concepts. Multidimensional data are represented in a *Data-Cube*, which may have an arbitrary number of *Dimension* instances such as time, region, or product type. A *Dimension* instance is split up into an arbitrary number of categories (instances of class *Dimension-Category*) which group business concept instances along the dimension. *Dimension-Category* instances may recursively be divided into subcategories grouping *Business-Concept* instances at a finer level of granularity. The three classes *Data-Cube*, *Dimension* and *Dimension-Category* are subclasses of *Entity.* Therefore, the structure of their instances can be described by an arbitrary number of *Attribute* instances.

Furthermore, a *Data-Cube* instance has a number of instances of *Fact-Attribute* (subclass of *Attribute)*, which represent business concept aggregations.

In the following, we now describe the interdependencies between business concepts on one side and the components of a data cube on the other side:

- ***Business-Concept*** ↔ ***Dimension-Category***: The mapping between *Business-Concept* instances and *Dimension-Category* instances is described by instances of the shared generic classes *Mapping* and *Filter* which have been described in Figure 4 of section 4.1.

- In the specific context of business concepts and multidimensional data, a *Mapping* instance connects exactly one *Dimension-Category* instance with an arbitrary number of *Business-Concept* instances. The *Filter* instance associated with the *Mapping* instance specifies which instances of a *Business-Concept* instance qualify themselves for the mapping, i.e. which instances logically „belong" to the particular *Dimension-Category* instance[6]. In the example of Figure 8, *Insurance-Claim* is a *Business-Concept* instance, and *3Q98* a *Dimension-Category* instance representing the 3rd quarter of the year 1998. With a *Mapping* instance between *Insurance-Claim* and *3Q98* and a *Filter* instance with predicate *in-interval(occurred-at, 7/1/1998, 9/30/1998)*, we would represent that exactly those insurance claims belong to *3Q98*, for which the value of the *occurred-at* attribute (declared in the super concept *Insurance-Event*) lays between *7/1/1998* and *9/30/1998.*

- If there is no filter associated with the mapping, this means that *all* instances of the source business concepts belong to the dimension category.

- ***Business-Concept*** ↔ ***Fact-Attribute***: In a similar manner, aggregation interdependencies between data cubes and business concepts are represented. Therefore, we use the classes *Aggregation* and *Function* of Figure 4, which represent an aggregation as follows:

- An *Aggregation* instance connects exactly one *Fact-Attribute* instance with an arbitrary number of *Attribute* instances (of *Business-Concept* instances). The *Function* instance associated with the *Aggregation* instance specifies how the attribute values are aggregated (e.g. by calculating the sum or average of the values of an attribute). A *Filter* instance is not necessary, as the *Mapping* and *Filter* instances between the *Business-Concept* instances and the *Dimension-Category* instances already specify which *Business-Concept* instances and therefore which attribute values constitute the source for the aggregation.

4.3.3 Semantics of *Business Concept* ↔ *Data Cube* Interdependencies

The semantics of mapping and aggregating business con-

6. The instances of a *Business-Concept* instance correspond to the tuples of a relational table. A particular relational table *Contracts* is an instance *i* of *Physical-Entity* with *i.name = „Contracts"*. The „instances" of this *Physical-Entity* are the tuples.

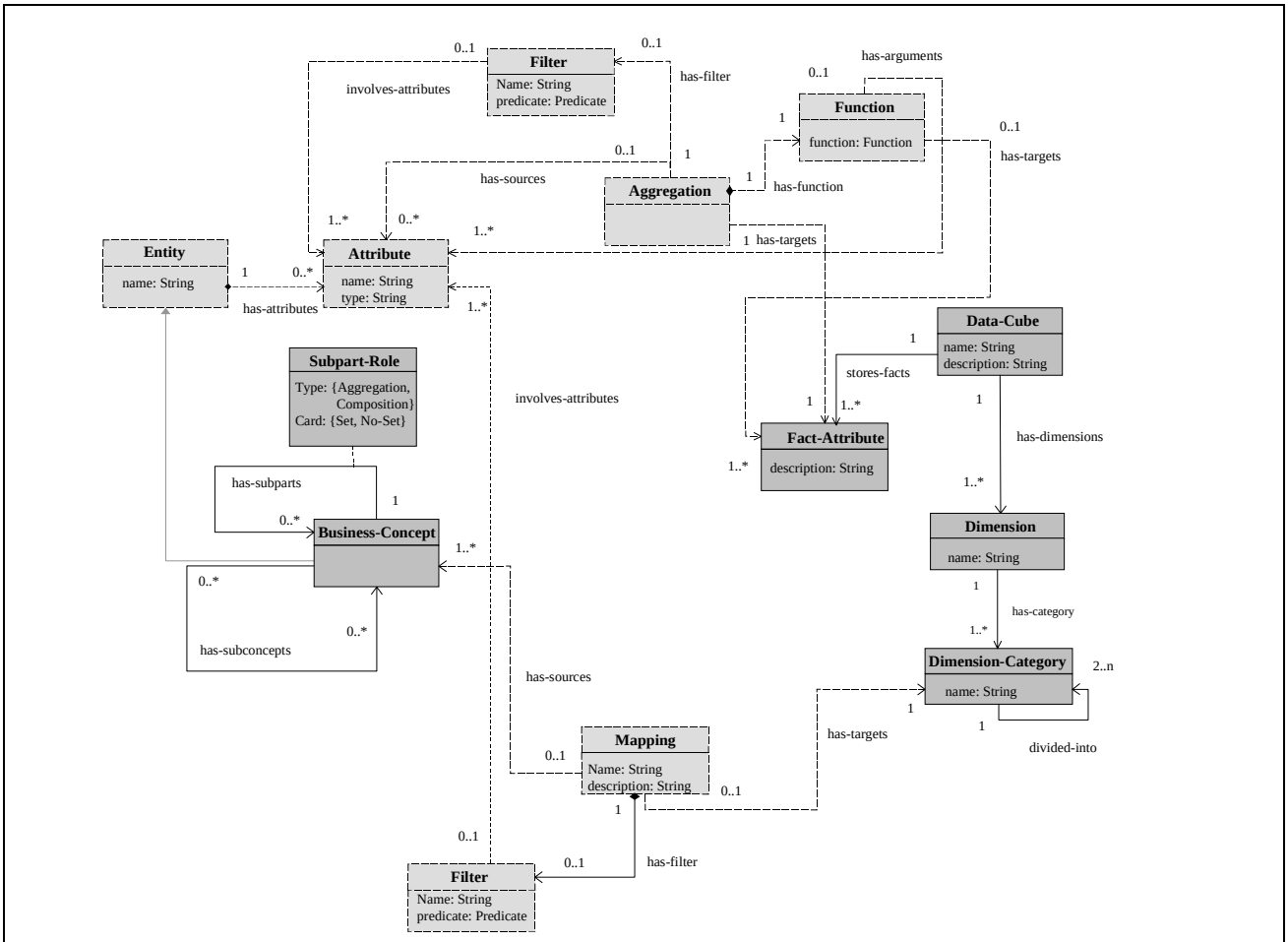

Figure 7: UML schema for concepts and multidimensional data from the business point of view (see text for details). *Data-Cube*, *Dimension* and *Dimension-Category* are subclasses of *Entity. Fact-Attribute* is a subclass of *Attribute*, and therefore inherits the *name* and *type* attribute (these UML inheritance relationships are omitted in this Figure). The dashed rectangles and arrows denote classes and associations already defined in the shared model of section 4.1.

cepts to a data cube can be best illustrated with an example obtained from Figure 8. Let us assume a business user wants to know the sum of payments for all insurance claims w.r.t. accident insurances (*Product* dimension) for the 3rd quarter of 1998 (*Time* dimension), where small children were damaged (*Insured-Person* dimension). The *Mapping* and *Aggregation* instances represent the aggregation interdependencies in the following way:

- W.r.t. the *Dimension-Category* instance *3Q98*, those instances of the *Business-Concept* instance named *Insurance-Claim*, which took place between *7/1/1998* and *9/30/1998*, are affected.
- W.r.t. the *Dimension-Category* instance for small children (of the *Insured-Person* dimension), all insurance claims where the damaged person is not older than 6 years, are affected. This is formally specified by the path expression *Insurance-Claim→because-of-damage→damaged-object→age* within the associated *Filter* instance. In case that the insurance claim does not affect a person (with the consequence that the damaged object does not provide an age attribute), the predicate evaluates to FALSE, so that the particular insurance instance claim is not considered.
- W.r.t. the *Dimension-Category* instance called *Accident-Insurances*, a filter specifying that only insurance claims related to accident insurance contracts has to be considered. This *Filter* instance with the necessary path expressions is not shown in Figure 8. It has a structure analogous to the *Dimension-Category* instance for small children above.
- Then, the set of the affected *Insurance-Claim* instances is the intersection of the three sets above. With this intersection, the sum of all *payed-sum* values can be performed according to the *Aggregation* instance.

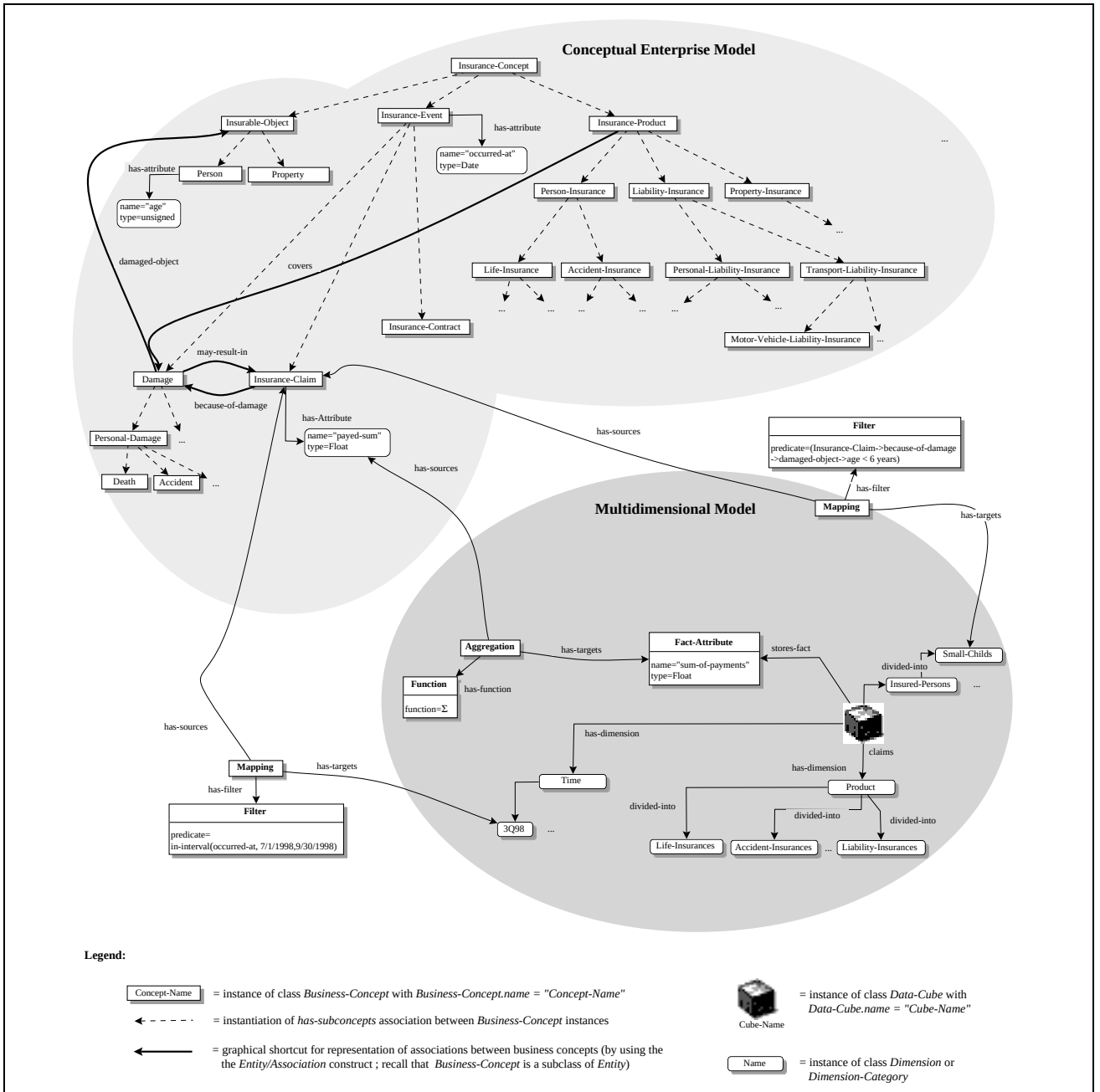

Figure 8: Instantiation of semantic metadata for an insurance domain. In the multidimensional model, a data cube for claim payment is represented. The *Mapping, Aggregation* and *Filter* instances represent the connections of the data cube components with the enterprise-related business concepts in the conceptual enterprise model (see text for details).

4.4 Interdependencies between Semantic and Technical Metadata and Ad-hoc query support

Semantic metadata only *describe* the contents of a warehouse from the business point of view, but do not support access to the physical data. Therefore, we also have to represent the dependencies between this business layer on one

side and the operational and data warehouse databases on the other side. This type of mapping information can then be used to translate ad-hoc queries on the semantic level into SQL-queries accessing the physical data of the warehouse. We sketch the usage of mappings and transformations between the technical and semantic layer in the context of ad-hoc query translation by the following example (see Figure 9):

Let as assume a business user wants to retrieve (the contract numbers of) all profession incapacity insurance contracts where an acute endocarditis[7] has been a medical problem in the biography of the insurance holder. The generation of the SQL-query accessing the relational base tables MEDICAL-INSURANCES and MEDICAL-PROBLEMS storing the related data is done in the following steps:

- First, the ad-hoc query specified at the user-interface is translated into an intermediate OQL-oriented format. OQL[8] has been chosen, as it fits well into the object-oriented data model of our business concept representation. The OQL notation of our ad-hoc example is:

  select p.contract-number
  from p in extension-of(Profession-Incapacity-Insurance-Contract)
  where exists e in p.has-subparts(Relevant-Medical-Event): e.event = „ACUTE ENDOCARDITIS“

  Herewith, *extension-of* represents the set of all instances of the business concept *Profession-Incapacity-Insurance-Contract*. The construct *p.has-subparts(Relevant-Medical-Event)* refers to all instances of the concept *Relevant-Medical-Event* which are related to the instance *p* via the *has-subparts* association.

- Second, the query generator inspects the mappings which have the two business concepts *Profession-Incapacity-Insurance-Contract* and *Relevant-Medical-Event* as targets. By doing this, it detects that the relational table MEDICAL-INSURANCES is the source for the concept *Profession-Incapacity-Insurance-Contract* (mapping „PII-Filter-Mapping“), and that the table MEDICAL-PROBLEMS corresponds to *Relevant-Medical-Event* (mapping „Event-Mapping“). Furthermore, it is detected that a foreign key relationship exists between the two tables, i.e. *cono* (for contract number) as part of the compound key *{cono, medical-problem, date-of-occurrence}* references the contract in the MEDICAL-INSURANCES table.

- Third, by using the mapping „PII-Filter-Mapping“ specifying that the *Profession-Incapacity-Insurance-Contract* represents exactly those MEDICAL-INSURANCES tuples where the insurance type is PII (for profession incapacity insurance), the query generator generates an initial query fragment:

  SELECT mi.cono
  FROM MEDICAL-INSURANCES mi
  WHERE mi.type=PII

- Fourth, by using the mapping „Event-Mapping“ and the foreign key relationship association, the query generator joins the two tables MEDICAL-INSURANCES and MEDICAL-PROBLEMS to achieve the part-of semantics between the two concepts *Profession-Incapacity-Insurance-Contract* and *Relevant-Medical-Event* (new fragments underlined):

  SELECT mi.cono
  FROM MEDICAL-INSURANCES mi, MEDICAL-PROBLEMS mp
  WHERE mi.type=PII AND
  mi.cono = mp.cono

- Fifth, the filter condition w.r.t. the medical event (acute endocarditis) completes the query:

  SELECT mi.cono
  FROM MEDICAL-INSURANCES mi, MEDICAL-PROBLEMS mp
  WHERE mi.type=PII AND
  mi.cono = mp.cono AND
  mp.medical-problem=„ACUTE ENDOCARDITIS“

This query is then processed on the relational tables.

5 Summary and Outlook

We have introduced an UML-based model for data warehouse metadata as a base for a data warehouse repository. The overall goal is to cover technical *and* semantic metadata relevant for warehouse environments. The model contains a number of generic classes such as *Entity, Association, Mapping* and *Transformation* for the representation of structures required both w.r.t. technical and semantic metadata. In the technical context, adaptations of these classes can be used to structurally describe table structures or record types of operational data bases, the data warehouse and OLAP databases. Furthermore, using the classes *Mapping* and *Transformation* we can describe the data movement processes from the operational sources to the warehouse and the OLAP databases. In the semantic context, we use specializations of these classes to represent business concepts and business-oriented views of data cubes. In this context, *Mapping* and *Transformation* instances describe the dependencies between business concepts and elements of a data cube such as dimensions, dimensions categories, and fact attributes. The *Mapping* and *Transformation* classes are also used to represent the

7. Inflammation of the heart muscle. Insurance holders who suffered from this disease in the past usually have to pay a significant extra charge for their medical insurances.
8. OQL = Object Query Language of the ODMG

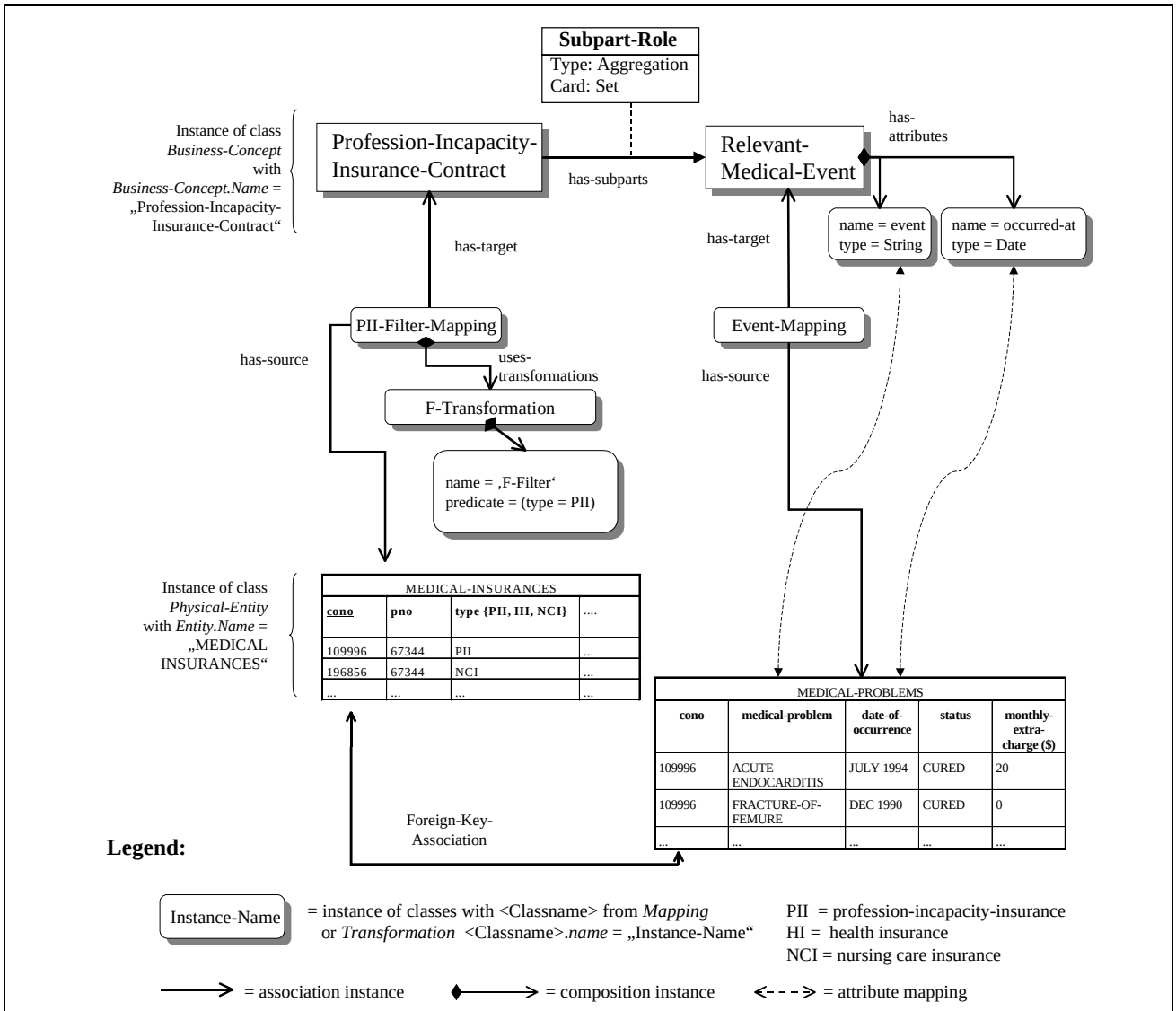

Figure 9: Mappings between the technical layer of relational tables and business concepts. These mappings can be used for translating ad-hoc queries on business terms into executable SQL queries operating on the data warehouse tables (see text for a query generation example).

dependencies between business concepts on one side and the tables and records containing the physical data for these business concepts on the other side.

By using an object-oriented modeling approach, the UML schemata can easily be extended to cover metadata types currently not addressed. For example, additional specializations of *Entity* (such as *Rule*) and *Association* could be introduced to address metadata related to an enterprise knowledge model.

We are currently implementing a repository prototype based on this model with the object-relational database management system INFORMIX. As a future goal w.r.t. technical metadata, the repository should not only store a „read“ extract of technical tool metadata, but should also support the re-definition of metadata and the propagation of re-defined metadata to the affected tools. This is useful, as the model we described in this paper provides a uniform view on different types of metadata, and abstracts from tool specific proprietary details. Redefined metadata shall then automatically be transformed to tool specific executable formats.

Furthermore, we will investigate how to comprehensively support queries at the semantic level by providing an automatic translation to query programs at the data warehouse level.

References

[BB+99] Bernstein, P.A.; Bergstraesser, T.; Carlson, J.; Pal, S.; Sanders, P.; Shutt, D.: *Microsoft Repository Version 2 and the Open Information Model*. Information Systems 24(2), 1999.

[BB99] Bernstein, P.A.; Bergstraesser, T.: *Meta-Data Support for Data Transformations Using Microsoft Repository.* IEEE Data Engineering Bulletin, March 1999, vol. 22(1): 9-16.

[BH98] Becker, J.; Holten, R.: Fachkonzeptuelle Spezifikation von Führungsinformationssystemen (Conceptual Specification of Management Information Systems. Wirtschaftsinformatik, vol. 40(6), 1998: 483-492.

[CGL+98] Calvanese, D.; De Giacomo, G.; Lenzerini, M.; Nardi, D.; Rosati, R.: *Source Integration in Data Warehouses.* DWQ Technical Report 1998.

[FS98] Fowler, M.; Scott, K.: *UML distilled: applying the standard object modeling language*. Addison Wesley, Reading, Mass., 1998.

[GMR98] Golfarelli, M.; Maio, D.; Rizzi, S.: *The Dimensional Fact Model: a Conceptual Model for Data Warehouses*. International Journal of Cooperative Information Systems, vol. 7(2&3), 1998:215-247.

[Hol99] Holten, R.: *A Framework for Information Warehouse Development Processes.* In: Becker, J.; Grob, H. L.; Müller-Funk, U.; Klein, S.; Kuchen, H.; Vossen, G.: Working Reports of Institut für Wirtschaftsinformatik Nr. 67, Münster 1999.

[JGJ+95] Jarke, M.; Gallersdörfer, R.; Jeusfeld, M.A.; Staudt, M.; Eherer, S.: *ConceptBase - a deductive object base for meta data management*. Journal of Intelligent Information Systems (Special Issue on Advances in Deductive Object-Oriented Databases), vol. 4(2), 1995: 167-192.

[JQJ98] Jeusfeld, M.A.; Quix, C.; Jarke, M.: *Design and Analysis of Quality Information for Data Warehouses*. In Proc. 17th International Conference on Conceptual Modeling (ER'98), Singapore, Nov 16-19, 1998.

[KQS+98] Katic, N.; Quirchmayr, G.; Schiefer, J.; Stolba, M.; Tjoa, A M.: *A Prototype Model for Data Warehouse Security Based on Metadata*. Proceedings DEXA 98.

[MC98] Metadata Coalition: *Metadata Interchange Specification*, Vers. 1.1, Aug. 1998 http://www.mdcinfo.com/standards/toc.html.

[Wie98] Wieken, J.-H.: *Meta-Daten für Data Marts und Data Warehouses*. In: Mucksch, H.; Behme, W. (Hrsg.): Das Data Warehouse-Konzept, Gabler, Wiesbaden, 1998: 275-315.

[Wie99] Wietek, F.: *Modelling Multidimensional Data in a Dataflow-Based Visual Data Analysis Environment*. Proceedings 11th Conference on Advanced Information Systems Engineering (CAiSE*99), Springer, 1999.