Metadata Management for Data Warehousing: Between Vision and Reality*

Anca Vaduva Klaus R. Dittrich

University of Zurich, Department of Information Technology
Winterthurerstr. 190, CH-8057 Zurich, Switzerland
{vaduva|dittrich}@ifi.unizh.ch

Abstract

Capturing, representing and processing metadata promises to facilitate the management, consistent use and understanding of data and thus better support the exploitation of masses of information that is available online today. Despite the increasing interest in metadata management, its purpose, requirements and problems are still not clear. This is particularly true in the area of data warehousing. The reasons are multiple. Compared to the past, today's metadata management considers a significantly larger spectrum of information (including even certain pieces of programs). Moreover, metadata are produced by various tools and reside in different sources which need to be integrated in order to ensure consistency and provide uniform access, impact analysis and tracking. Existing work has only partially covered some of these aspects, such that no comprehensive view exists so far. Therefore, this paper summarizes the most important issues of metadata management for data warehousing, including the role of metadata and solved and unsolved problems of the available solutions. The design of an appropriate information model, metadata integration and advanced user interaction facilities are crucial questions to be answered.

1 Introduction

Data warehousing is a collection of concepts and tools which aim at providing and managing a set of integrated data (the data warehouse) for business decision support within an organization. The overall purpose is to discover and explore business trends and thus achieve better and faster decisions regarding multiple aspects of business like sales and customer service, marketing, or risk assessment. Starting with the pioneering work of Inmon [11], the popularity of data warehousing grew exponentially during the last years. As a consequence, an abundance of software products for building and exploiting data warehouses overflows the market. This proliferation of tools and the spreading of data warehousing within enterprises contribute to an increased complexity of data management and processes in general. In order to master this complexity, a consistent management of related metadata is required.

*This work was partly supported by the Swiss Federal Office for Professional Education and Technology under grant KTI-3979.1 (SMART) and by Rentenanstalt/Swiss Life

Metadata in this context is generally defined as any information that may be used to support the administration and effective exploitation of data warehouses. Obviously, such information exists everywhere: either directly in data dictionaries and certain tool repositories or hidden in skripts, programs, user manuals, or, worst of all, in paper documents and people's brains. However, metadata are not worth much unless they are captured, stored and *consistently managed* in order to be uniformly accessible by users and software components. Even if the domain of metadata management has significantly evolved in the last years and many concepts have been developed, the construction of an enterprise-wide metadata management system is a demanding task. The reasons are multiple. On the one hand, there is a large spectrum of metadata to be uniformly managed, covering not only various user purposes (e.g., *technical* and *business* metadata) but having also fundamentally different scopes: *general documentation* for users versus *control information* for software programs. Particularly the second category raises problems not encountered in "classical" metadata management. On the other hand, there are a lot of commercial products that claim to manage metadata (either on purpose or as a side effect) but their elementary distinguishing features are not clear at a glance. In the following, we address these and other aspects with the aim to clarify the confusion that dominates in the area.

This paper provides an overview of metadata management for data warehousing. It discusses major aspects as *what* the pieces of information (i.e., the metadata) to be generated and captured for data warehouses are and *how* they should be stored and managed in order to achieve maximum benefits. We also discuss solved and unsolved issues and give some hints about the weaknesses of partial solutions for metadata management proposed so far[1]. Finally, we elaborate on metadata integration and present perspectives for more flexible approaches. In practice, the main problem of integration is the lack of a unique standard that commercial products comply with. The recently announced unification of the two available standards for metadata representation and interchange (OIM [16] and CWM [17]) raises hopes that the resulting standard will win a broader acceptance among vendors than the original ones.

The remainder of this paper is organized as follows: Section 2 introduces basic notions necessary to understand the paper. Section 3 analyzes the concepts and requirements metadata management is based on. Section 4 characterizes the state of the art and its problems. In Section 5 approaches for future metadata management are discused. Section 6 concludes the paper.

2 Preliminaries

In the following, we first focus on the architecture of a data warehouse system; for deeper consideration of the topic, an extensive literature is available [11, 7, 14, 15]. Then, we discuss what information is considered as metadata. Because of its wide use, a compact, precise definition of metadata can hardly be provided. Thus, we explain the notion by means of examples and a coarse classification.

2.1 Data Warehouse Systems

The kernel of a data warehouse system is the *data warehouse*. It serves as an (enterprise-wide) collection of integrated data, often stored in a relational database system. In particular, the

[1]This paper is not intended to be a snapshot of market offerings. Comprehensive overviews that include more detailed descriptions of products and standards have been given in [20] and, with a broader discussion of metadata in general, in [19].

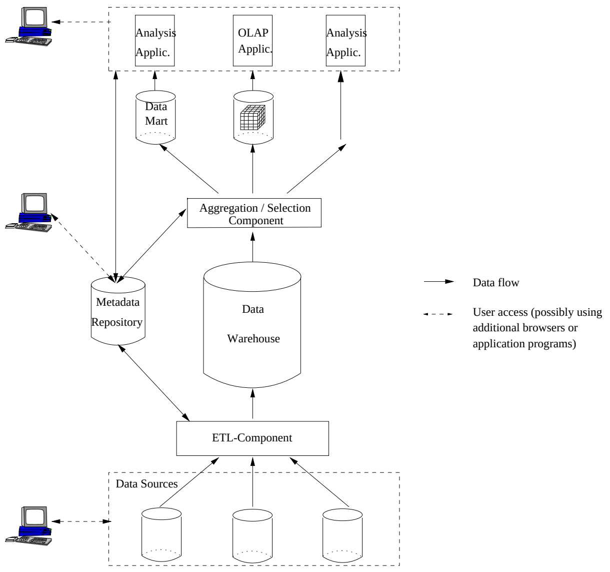


Figure 1: A typical data warehousing system architecture

data warehouse incorporates data collected from many *data sources* with various data models and heterogeneous structure, e.g., database systems, flat files, indexed files, proprietary systems, etc. A typical data warehouse system architecture is depicted in Figure 1.

Data marts are data stores (usually subordinated to data warehouses) that are built with the aim to fulfill specific application requirements of a certain group of users, e.g., of a department or a geographical region. A data warehouse and its data marts are populated by performing an initial loading and then regularly updated during the periodical execution of a *refreshment* process. The *ETL* (*E*xtraction, *T*ransformation and *L*oading or data acquisition) component is responsible for populating the data warehouse: it extracts data from sources, transforms and finally stores them in the data warehouse. Data transformation comprises various forms: reconciliation of syntactic and semantic differences between operational sources, consolidation, and mapping from local data models to the global one. Similarly, the *aggregation and selection component* prepares data to be fed into analysis applications or to be stored into data marts.

Data warehouses are built for analysis purposes. Analysis applications are of various complexity ranging from reports relying on simple calculations to complex data mining methods. However, the most popular analysis means for data warehousing are *OLAP applications* which enable users to examine data within a multidimensional model allowing to instantaneously retrieve and summarize data. The multidimensional model provides measures (i.e., business facts to be analyzed like sales or shipments) and dimensions for these measures (i.e., the context in which the measures have values, e.g, products, customers, time, region). Dimensions have associated hierarchies that specify aggregation levels for viewing the data. In order to serve OLAP- applications, data warehouses or data marts are implemented either by special multidimensional databases (MOLAP) or relational databases (ROLAP).

The *metadata repository* ideally stores and manages all the metadata necessary to improve work with the data warehouse. It is the integrating component of the architecture and plays a key role in every phase of data warehousing.

2.2 What is Metadata?

Metadata (data about data) is commonly understood as any information needed to develop, maintain and exploit a data warehouse system. It is usually distinguished according to its use into: *business metadata*, mainly needed by end-users, and *technical metadata*, produced and used by database administrators or by software components of the data warehouse system. The category of business metadata contains end-user-specific documentation, dictionaries, thesauri, and domain-specific ontological knowledge, business concepts and terminology, details about predefined queries, and user reports. In contrast, technical metadata includes schema definitions and configuration specifications, physical storage information, access rights, executable specifications like data transformation and plausibility rules, and runtime information like log files and performance results.

Another distinction regards *descriptive* versus *transformational* metadata. The former category includes information related to the structure of data sources, data warehouses, and data marts. In this context, we distinguish between metadata concerning the entire schema (e.g., schema description, statistical values like, e.g., the number of entries in the database) and metadata associated with parts of the schema. Examples include quality attributes that specify the credibility of single attributes (e.g., birthdate).

In contrast, transformational metadata is information associated with data processing: information regarding the loading and refreshment process, the analysis process, and inherently the administration of the data warehouse system. Examples are rules specified for data extraction, transformation, and aggregation and their execution schedules; rules are usually defined by means of executable specification languages.

A special category constitutes metadata concerning the organisation of the enterprise which includes administrative data, information related to staff, as e.g., user rights to access the data warehouse, data sources, and data marts.

Specific information about certain aspects of data warehousing like ETL und OLAP is stored as metadata as well. For example, cubes, dimensions, and hierarchies needed for building multidimensional views are explicitly available as metadata.

In analogy to database design, metadata may be classified according to the three levels of abstraction (conceptual, logical and physical). Examples are descriptions of conceptual, logical and physical schemas, e.g. ER-schemas or relational schemas.

3 Metadata Management for Data Warehousing

Metadata are stored and maintained in repositories. These are structured storage and retrieval systems, typically built on top of a conventional database management system. A repository is not simply a storage component but also embodies functionalities necessary to handle the stored metadata. It is (logically) independent of the data warehouse, even if physically the same database management system (DBMS) platform is possibly used. Note that in reality, more than one repository is used to manage metadata available in a data warehouse system. However, for simplification, we consider in Section 3 the existence of a single repository for managing all kind of metadata and postpone the discussion of metadata management architectures with many repositories to Section 4.2.

In the following, we first discuss the role played by metadata management in Section 3.1 and then briefly address the requirements for metadata management in Section 3.2 and 3.3.

3.1 Role of Metadata Management

Metadata is captured, generated and managed in a repository in order to be used in two ways

- as consistent *documentation* about the structure, the development process and the use of a data warehouse system. It may comprise both technical and business metadata and is needed by users (i.e., end-users, system administrators, and application developers) to achieve their tasks. Examples of information expected from a repository are:
  - What means column *payroll_a*?
  - What means the term *insured person*, depending on context? Where do I find information about *insured person*s?
  - How many data records are processed during data aquisition?
  - What is the difference between two versions of a certain software module?
  - How granular is the history, i.e., what is the shortest time interval betweeen entries in the history?
  - What happens if data source *ACCOUNTING_DATA* is disconnected from the system?
  - Which are the sources and tables the warehouse attribute *payroll_a* originates from? Which fields of which sources are used to compute the warehouse attribute *payroll_a*?
- as *control information* for certain tools. Tools store either static information (like structure definitions, configuration specifications, etc.) or part of the logic of their programs as metadata. In other words, control information is stored in repositories, outside programs and applications. At runtime, this kind of metadata is read, (possibly) interpreted and dynamically bound into software execution. If new requirements arise, metadata may be easily changed without affecting the programs sharing it and without requiring re-compilation of these programs.

In either way, the generation and management of metadata serves two purposes: to minimize the efforts for development, maintenance and administration of a data warehouse and to improve the effectivity of extracting information from it. The first objective mainly concerns

- *improvement of the flexibility of the system and reuse of existing software modules.* Since reusable abstractions and configurations are explicitly stored as metadata outside application programs, the system may be extended and adapted without difficulty. Besides the reuse of "code fragments", design decisions adopted for existing applications may also be stored as metadata and thus reused for analysis and design of new applications.
- *automation of various administration processes.* Since the repository is (ideally) shared by all tools or software components involved in data warehouse processes, they may pass control of execution to each other by means of metadata (*metadata-driven* processes). Information about process execution (access logs, number of records added to the warehouse etc.) may be stored in the repository as well for easy access by the administrator.
- *impact analysis.* Often, administrators need to evaluate the impact of potential changes in the data warehouse system before they are actually executed. For example, changes in the schema of sources may have consequences for transformation rules (e.g., resulting

in type mismatches, violations of referential integrity) and inherently for the structure of the data warehouse or the data marts. Obviously, a prerequisite is to "link" information in the repository: data sources are linked to transformation rules which are linked to certain tables in the warehouse. In this way, one may automatically detect which changes of the sources may affect the warehouse.

The second objective refers to the effective extraction of information from data:

- *improving interaction with the data warehouse system during analysis.* Interaction may be performed either by means of simple queries and reporting applications or by using complex analysis applications. Metadata provides information about the meaning of data, terminology and business concepts used within the enterprise and their relationship to the data. Thus, metadata improves query, retrieval and answer quality. It allows for precise, well-directed queries and helps to understand the application domain and its representation in the data warehouse in order to adequately apply and interpret results.

- *improving data quality.* Data quality includes dimensions like consistency (whether the representation of data is uniform and no duplicates, no data with overlapping and confusing definitions exist), completeness (whether data is missing), accuracy (the conformity of the stored value with the actual value, including precision and confidence of data), timeliness (whether the recorded value is up-to-date). Quality assurance rules have to be defined, stored as metadata and checked each time the data warehouse is refreshed. In addition, high data quality requires the support of data tracking. Metadata provides information about the creation time and the author of data, the source of data (data provenance), the meaning of data at the time it was captured (data heritage), and the path followed from source to the current site (data lineage) [6]. In this way, users may reconstruct the path followed by data during the transformation process and verify the accuracy and credibility of returned information.

- *enforcing a unique terminology and communication language within the enterprise.* Since the metadata repository is available as a unique documentation source for users, it provides a consistent means for people to communicate, understand, and interpret information provided by the data warehouse system, eliminates ambiguity and helps to guarantee consistency of information within the enterprise.

In summary, metadata promises to solve a lot of problems for improving work with a data warehouse. However, partially managing metadata for a specific purpose in a specific sub-domain (like data acquisition, or OLAP) renders only reduced advantages. Benefits like consistency, data tracking and impact analysis are provided only if the entire *integrated, consistent and uniform management* of all metadata is supported.

3.2 General Features of Metadata Management Systems

A number of approaches exist that deal with general functional features of metadata management. Inherently, insights gained so far apply for data warehousing as well. Among typical general features which have been considered especially in the work of Bernstein [1, 3], we pick the three most important ones (change management, interoperability and user access) and discuss them below. The specific needs of a particular application domain (i.e., data warehousing in our case) is only reflected in the repository schema which will be discussed in Section 3.3.

Change Management. Change management handles of changes inside and outside the repository. A *notification mechanism* is necessary to propagate changes to tools that have registered their interest in being notified. Also, users who have previously "subscribed" are informed. Furthermore, the repository has to provide *version and configuration management* and *impact analysis* mechanisms.

Interoperability and Tool Access. The interaction of software components and tools with the repository requires appropriate mechanisms, in particular:

- a comprehensive application programming interface (API) for metadata read and write access by other software components,
- interfaces ensuring the interoperability with other repositories.
- a flexible core (meta)model that allows to easily extend a given set of types concerning additional tools or new data sources.

User Access. The metadata repository has to offer an appropriate, user-friendly (and thus highly graphical) interface and suitable mechanisms for interacting with human beeings. *Browsing* mechanisms are necessary to navigate within the metadata collection along the links between the individual metadata elements. Navigation is "driven" by the underlying repository schema. The structure (schema) of the repository has to allow *querying* according to specific conditions and *filtering*, i.e., the selection of relevant information based on information retrieval techniques. *User views* are used to restrict access to information according to both user interests and user access rights. Furthermore, user access tools have to support editing and designing mechanisms for manually entering metadata elements and relationships between them.

3.3 Metadata Structures

A metadata repository has to provide a schema fitting its utilisation purposes: first of all, this structure has to reflect the diversity of information required by users and tools (e.g., business and technical metadata, descriptive and transformational, conceptual and logical, etc). Then the repository schema should be easily extendable in order to support extension of the system for additional demands, e.g., information types, sources and tools that access it. At the conceptual level, the structure of the repository is described by a *metamodel* or an *information model.* In order to better understand the role of a metamodel, we make a short digression to the theory of modelling.

3.3.1 Metalevels

The notion of "meta" is closely related to the abstraction levels of modeling. For the modeling of complex information systems, at least 4 modeling levels are required (see Figure 2). Each level is the "meta" level for the next lower level, that means it comprises the modeling constructs used to define the information on the level below. To start with, on the lowest level, level 0, there are the actual data items (e.g., the customer data). The levels above contain the metainformation: level 1 contains metadata (e.g., the database schema), level 2 specifies the schema used to store the metadata (the so-called metamodel, information model or metadata schema). Typically, data models belong here. Level 2 also includes metamodels

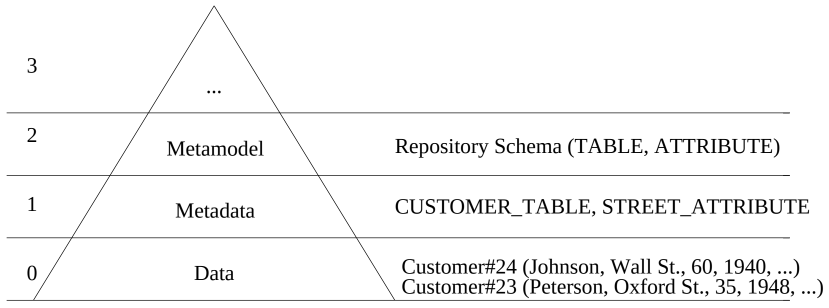

3
...
2
Metamodel
Repository Schema (TABLE, ATTRIBUTE)
1
Metadata
CUSTOMER_TABLE, STREET_ATTRIBUTE
0
Data
Customer#24 (Johnson, Wall St., 60, 1940, ...)
Customer#23 (Peterson, Oxford St., 35, 1948, ...)

Figure 2: Levels of Metamodeling

of common modeling languages like UML (Unified Modelling Language) which, for example, may be directly used to define a database schema. Level 3 contains the metametamodel that unifies the different metamodels of the level 2. Note that an instantiation relationship exists between levels (i.e., on the level below there are instances of the elements above) while on the same level the (meta)model may be refined and extended with specialization relationship between elements.

Among the extensive efforts of various organizations to establish a multi level system architecture, we mention OMG[2] which aims with its approach at enabling the integration of tools and applications running in heterogeneous, distributed environments accross their life cycle. Further work on meta-modelling can be found in [2, 4, 5, 12].

Metadata management is concerned with levels 1 and 2 of the pyramid of Figure 2. The aim is to have a uniform metamodel (or information model) that serves the following purposes: users need the repository as a consistent documentation; software components may use metadata of the repository as control information in order to achieve their tasks. In the following, we give two simple examples of sub-metamodels: one for documentation purposes and one for tool controlling. They may be seen as parts of a larger metamodel.

3.3.2 Metamodel for System Documentation

A common example for metamodels aiming at documentation is a metamodel representing information regarding relational database structures on three schema design levels: conceptual, logical and physical. We use UML to depict metamodel classes, their associations on the same level and - most important of all - between levels (see Figure 3). The conceptual level manages information about ER-schemas, i.e., ENTIT(Y)ies, their RELATIONSHIPs and their PROPERT(Y)ies. The logical level manages RELATIONs, their FOREIGN_KEY relationships and ATTRIBUTEs. The physical level represents platform dependent information, in particular the ORACLE implementation of the relational schema (ORA/TABLE contains one or more ORA/COLUMN, and zero, one or many foreign keys ORA/FK; in turn ORA/FK is associated to one table, aso.) Since levels are related to each other (e.g., an ENTITY is represented as a RELATION in the relational model and as a TABLE in the ORACLE data model), navigation between the three levels is possible.

Instances of metamodel classes are the metadata objects (not illustrated in the figure), e.g., entity CUSTOMER_ENT, table CUSTOMER_TAB, attribute NAME, etc. In turn, if we consider a meta-metalevel with two classes CLASS and META_ATTRIBUTE, the following instantiation relationships exist: ENTITY, RELATION, ORA/TABLE, RELATIONSHIP, FOREIGN-KEY and ORA/FK are all instances of CLASS while PROPERTY, ATTRIBUTE,

[2]http://www.omg.com/

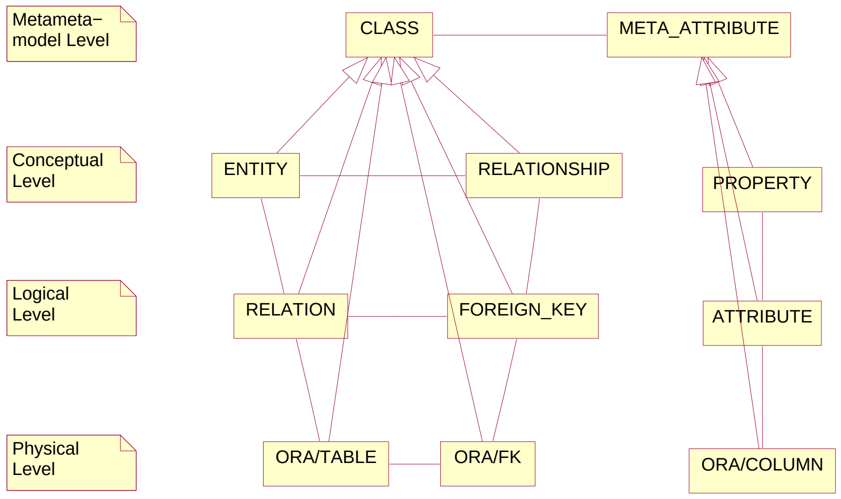


Figure 3: Descriptive metadata for conceptual, logical and physical schema

ORA/COLUMN are all instances of META_ATTRIBUTE.

Summarizing, the most important property of a metamodel for documentation is the existence of relationships among classes and implicitly among their instances in order to provide navigation possibilities within the structure. In this way, users have access to related information starting with an initial entry point.

3.3.3 Metamodel for Automatic Generation of Structures

Systems driven by metadata are more flexible than hard-coded systems since they may be easily configured when new requirements arise. We take as example a metamodel representing metadata that serves as controlling information for a tool to automatically build multidimensional OLAP structures. These structures may be dynamically built instead of being hard-coded in scripts or applications.

Consider the metamodel part depicted in Figure 4. There are four classes, *Dimension*, *Hierarchie*, *Fact* and *Cube*. A dimension may have one or more hierarchies, a cube contains at least one dimension and depends on a fact. Instances of these classes, i.e., the metadata, are stored in the repository as class extensions. At runtime, when a cube has to be defined, the method *DefineCube()* is invoked with corresponding (four) parameters. Their values are chosen among the instances of class *Dimension* (e.g., object *Time*, *Region* and *Product*) and *Fact* (e.g., *Sales Revenue*). The method *DefineCube()* may invoke some SQL commands applied for example to a ROLAP structure in order to calculate (select and aggregate) the corresponding cube. The flexibility of this design pattern is given by the fact that the method *DefineCube()* may be executed for any instances of *Dimension* and *Fact* and may therefore build any required OLAP cube.

Summarizing, we presented in a simplified form two kinds of metamodels: for documentation purpose and for metadata-driven tools. The main distinguishing feature is that the first metamodel may be easily extended without affecting its main purpose - navigation within the documentation. In contrast, metamodels for metadata-driven tools, i.e., where metadata serves as controlling information, are not designed to be extensible. They fulfill their (narrow) purpose (in our case building OLAP structures) but to this end the behavior (e.g., *Define Cube()*) has to be “wired” to the given structure (e.g., *Dimension*, *Hierarchie*, *Fact*).

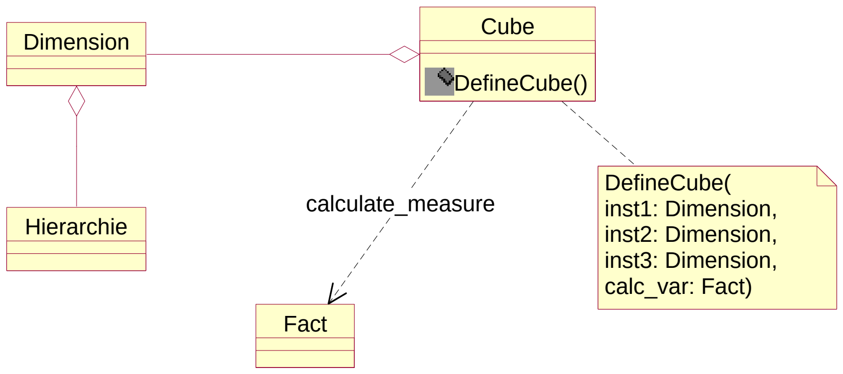


Figure 4: OLAP metamodel classes

4 State of the Art in Metadata Management for Data Warehousing

In this chapter, we give a brief overview of the state of the art with a focus on discussing weaknesses of today's information models provided by existing tools (Section 4.1) and by existing solutions for metadata integration (Section 4.2).

4.1 Tools for Metadata Management

In accordance with the use of metadata disscused in Section 3.1 and the metamodels presented in Section 3.3, two categories of tools that manage metadata can be distinguished:

- *metadata-driven* tools use metadata as a means to an end. They store and use metadata as *controling information* for achieving their specific task. These are the new generation of tool packages for data warehousing, e.g., for building the data warehouse (like PowerMart[3], Ardent[4]) or for using it (like Cognos[5], Business Objects[6]). Research prototypes following the same principle exist for data aquisition [21] and data preprocessing for data mining [13].

- *general-purpose metadata management tools* aim at enterprise-wide metadata management mainly for *system documentation*. Their objective is the extensive provision of information for all user categories and the sharing of metadata between software components within a company. According to a Gartner Group survey[9], Computer Associates[7] is classified as the "leader" in the general-purpose metadata management market segment while three vendors (Allen Systems Group[8], Unisys[9] and Microsoft[10]) are classified as "visionaries".

In the following, we discuss particularities of both approaches considering mainly their metamodels and user access interfaces.

[3]http://www.informatica.com/
[4]Ardent Software was recently acquired by Informix, http://www.ardentsoftware.com
[5]http://www.cognos.com
[6]http://www.businessobjects.com
[7]acquired recently Platinum Technology, http://www.cai.com
[8]acquired recently Viasoft with its Rochade product, http://www.asg.com
[9]http://www.marketplace.unisys.com/urep/
[10]http://msdn.microsoft.com/repository/

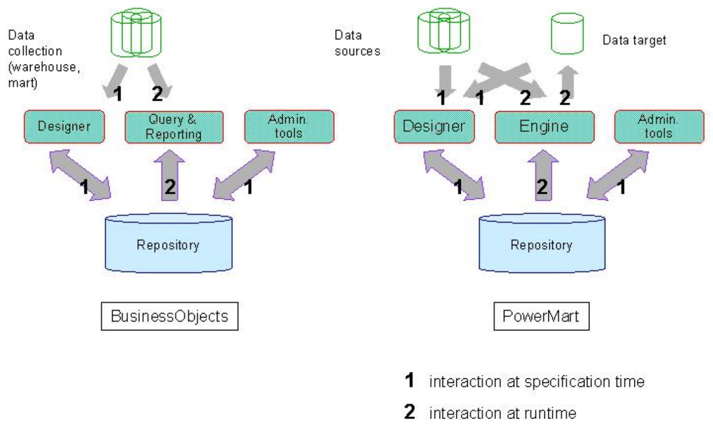


Figure 5: Metadata-driven tools

4.1.1 Metadata-Driven Tools

Metadata-driven software packages have as kernel a repository with various tools accessing it. A typical architecture is illustrated on the basis of two commercial products in Figure 5. Metadata is specified by means of adequate tools (e.g., *Designer*, *Administration tool*) and stored into the repository. Data structure definitions have to be extracted and stored as metadata as well (see the interactions at specification time). At runtime, an appropriate tool has to read the data from a given data collection, process it and, depending on the purpose, either store it into a target database (e.g., PowerMart) or return an analysis result (e.g., Business Objects). For a better understanding, we consider in detail the two metadata-driven tools of Figure 5.

PowerMart aims at implementing data acquisition, i.e., the extraction, cleaning, integration, transformation and loading of the data into the warehouse. The process is split into many steps (*mappings*) which are then executed in a certain order. Mappings are specified by means of a component called *Designer* and stored in a repository. In Figure 6, the graphical interface of *Designer* is illustrated. Each mapping has one or more sources (e.g., flat files) and one or more targets (e.g., ORACLE tables). Sources and targets are schema definitions imported either from corresponding data sources, the data warehouse or data marts or from temporary tables. The data "flows" between sources and targets through *transformations* which perform certain operations on it. Transformations are depicted in Figure 6 as the icons that make the link between the source on the left and the target table on the right of the window. When specifying mappings, the developer has to choose among 14 transformations (including e.g., aggregation, join, expression), configure and link them in accordance with the processing algorithm. When the mapping definitions and the order of their execution are stored, they are"broken" into code parts specified in an SQL-like language and dispersed over many tables in a relational database (usually ORACLE). At runtime, these code parts are read, interpreted, linked and executed by the *Engine* and the results are stored in the target database.

A major characteristic of the PowerMart language is the set-oriented processing of data. This feature may sometimes facilitate the processing, but may also require to rethink the (usually procedural, instance-oriented) algorithm in a different programming paradigm[11].

[11]For example, iteration has to be hand-coded because no iteration construct is directly available.

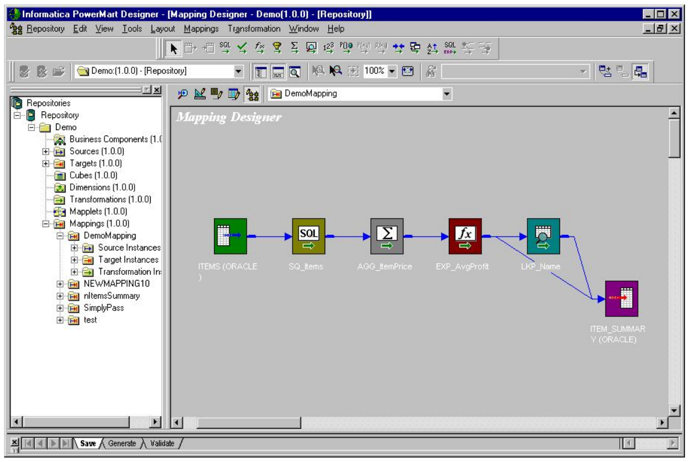


Figure 6: PowerMart Designer

Business Objects is a software package for querying, reporting and applying simple data mining with a typical metadata-driven architecture (see Figure 5). Besides access rights and generated documents, the repository is used as storage for *universes*. A universe contains mappings between business terms commonly used by end-users (like “product”, “sales revenue”, “last year”) and their representation on the implementation level, e.g., corresponding tables, SQL queries or stored procedure calls. The end-user manipulates known terms available on an intuitive drag & drop interface and generates queries which are internally translated into queries understandable by the data source (e.g., SQL if the data source is a relational database system). The universe represents the metadata. It allows users to query database systems (e.g., the data warehouse) without any database technology knowledge. The prerequisite is to have the universe specified by means of the *Designer* tool before use.

Discussion. As expected, metadata-driven tools cover only a limited diversity of metadata; only metadata necessary to fulfill the purpose of the tool is stored. Furthermore, since the operation of these tools requires their software components to read metadata of a certain structure and format, they are “bound” to the given repository structure which is inherently not designed for extension or update purposes.

The advantage of explicitly storing semantic aspects outside skripts and programs is certainly their *reuse* during development. In addition, it is claimed that *maintenance* is easier with metadata-driven tools since metadata may be changed independently of the clients accessing it. However, the problem of metadata-driven tools is to offer right abstractions, i.e., the right granularity of metadata to enable reusability and improve maintenance of the system. If the granularity is low like in the case of transformations in PowerMart, the developer still has to use a low-level implementation language, lacking higher abstractions[12]. For complex algorithms, elegant reusable implementation is as challenging as with any other general purpose implementation language. Thus, maintenance is still a demanding task.

Since transformation rules are stored in the natural form “source -> target”, data can be (kind of) *tracked* from the source to the target (e.g., data warehouse, data mart or OLAP view) and impact analysis may thus be automatically provided. However, note that data tracking is only possible to some extent. Only dependencies between attributes and fields of the original sources and targets are available. The processing track or the calculation formula

[12]As an example, a “higher abstraction” would be a reusable component for history strategy, which offers many alternatives for implementing histories of any data element (e.g., record, attribute, table)

cannot be derived. A developer must have a look at the transformation rules (e.g., mappings) by means of the *Designer* to possibly understand how an attribute of the target has been calculated.

As far as tools like Business Objects are concerned, their metadata is used for a narrow scope, the structure is less complex and thus the tool's purpose is easier to achieve. The reason is that Business Objects uses descriptive metadata which are easier to handle than the transformational metadata (mappings, transformations) of PowerMart.

4.1.2 General-Purpose Metadata Management Tools

General-purpose metadata management mainly consists of a repository, a browser that allows user access and interfaces for import/export data from other tools or repositories. In the following, we focus on repository structures, i.e., metamodels and user access mechanisms.

Metamodels. Since the main purpose is the documentation of structures, systems and applications, a *flexible metamodel* is provided with a granularity that allows to store the entire spectrum of metadata required and the effortless extension for new user demands and new tools that have to access the repository (e.g., CASE tools, versioning tools, impact analysis tools).

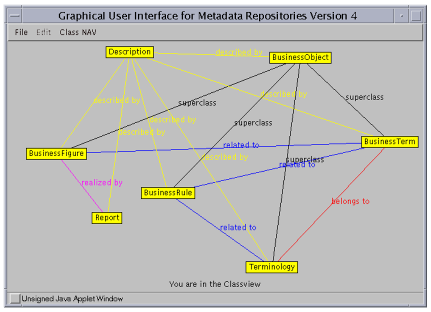
Regarding the representation on different abstraction levels (e.g., conceptual, logical physical illustrated in Figure 3), descriptive metadata for data structures do not raise significant problems. In contrast, transformational metadata are more difficult to handle. The complexity of the task is not suprising since it is closely related to software engineering objectives people are still working on: to break a problem into smaller tasks and solve and describe these in a sequence of steps (analysis, design and implementation), such that maintenance and reuse are facilitated. Even if transformational metadata in data warehousing concerns data processing rules only (e.g., the mappings of PowerMart), their documentation and representation on higher abstraction levels than the implementation level and the link to business rules governing the enterprise is a challenge for the repository administrator; no generic solution exists yet.

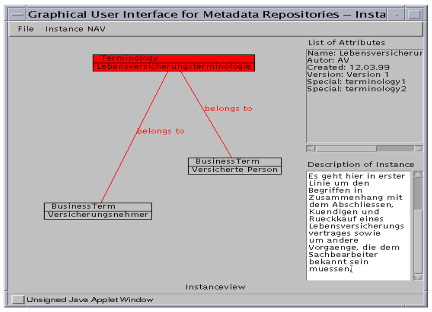
User Access. User access tools have to enable the navigation within the metamodel and display any metadata, independent of the given application domain. Thus, user access tools are bound either to the meta-metamodel (level 3 of the metalevel pyramide in Figure 2) or to given core metamodel(s). If, for example, a graphical representation is hard-coded for some given classes, any user-defined specialisation of a class is similarly displayed as its superclass during navigation within the repository structure.

A solution for ensuring extensibility and flexibility of the metamodel is the use of "self description". That means, both metadata schema and instances are managed by the same (hard-coded) metamodel. Figure 7 depicts first experiments [18] with browsing interfaces for both metadata classes (i.e., the metamodel) and its instances (i.e., the actual metadata). One can easily switch between the two views; navigation mechanisms within a view are similar since classes and instances are uniformly managed through the use of "self description".

User views have to restrict access to parts of the metamodel according to specified roles (e.g., power user, administrator, developer).

General-purpose metadata management systems cannot be successful as standalone information systems. They require the integration with other tools which both automatically update the information in the repository and use metadata for development and maintenance purposes.




Graphical User Interface for Metadata Repositories – Insta
File
Instance NAV
Terminology
Lebensversicherungsterminologie
belongs to
BusinessTerm
Versicherte Person
BusinessTerm
Versicherungsnehmer
Instanceview
Unsigned Java Applet Window

List of Attributes

Name: Lebensversicherur
Autor: AV
Created: 12.03.99
Version: Version 1
Special: terminology1
Special: terminology2

Description of Instance

Es geht hier in erster Linie um den Begriffen in Zusammenhang mit dem Abschliessen, Kuendigen und Rueckkauf eines Lebensversicherungs vertrages sowie um andere Vorgaenge, die dem Sachbearbeiter bekannt sein muessen.

Figure 7: ClassView (left) and InstanceView (right)

4.2 Metadata Interoperability and Integration

The simplest solution for metadata management would be a single, centralized repository for managing all kinds of metadata within an enterprise. In this case, metadata is uniformly and consistently managed, and accessed by all possible consumers (i.e., users, applications). However, the use of different tools, software components and repositories with divergent data models, following diverse representation formats, makes centralization close to impossible. The usual solution adopted in enterprises is *decentralized metadata management*, based on ensuring interoperability between repositories.

Interoperability. Bi-directional tool-specific interfaces are used when repositories have to exchange metadata (e.g., Business Objects has to import the warehouse schema definitions stored in PowerMart). This requires to agree on a common metadata representation in a given tool environment or to use a predefined one-to-one interchange format between commercial products, assuming they provide any. In other words, a common interchange representation format has to be adopted and used when data is imported or exported between repositories and/or tools. Apart from the problems raised by agreeing on a unique exchange format, the main disadvantage of decentralized metadata management is simply the lack of system-wide metadata integration. Repositories are storing and importing only some kinds of metadata, necessary to achieve their special purposes. They cannot establish and exploit links between metadata of various domains, they cannot perform system-wide impact analysis since only a reduced amount of metadata is available in a certain repository. For example, data tracking for figures in a certain OLAP report is based on questions like "which fields of which data sources have been used for calculating these figures". The answer can be given only in an integrated metadata management system where all metadata (including those originating from data acquisition, data warehouse and OLAP) are consistently and uniformly managed and thus all links between related metadata elements are available as well.

Integration. First of all, integration presumes the existence of a unique metamodel that combines and links all the types of metadata with their various purposes (including e.g., system documentation and tool controlling). All tools and repositories to be integrated have to find a place for their metatada in this metamodel. Finding or selecting such a unique metamodel is not a trivial task. The most often adopted solution for integrating metadata is the use of general-purpose metadata management tools. The metamodel provided with the tool is used to manage metadata uniformly. Import/export interfaces are used to import all metadata available in metadata-driven tools or other repositories into the given general-purpose tool.

Standards. For both, interoperability and integration, the main problem is actually to agree on a common metadata representation and exchange format to be used for exchange interfaces and as a core metamodel for integration. Coalitions of manufacturers or all-from-one-hand argumentation of certain vendors are only partial solutions. The freedom of combining products from any manufacturer to create a customized system is only given if a standard exists and all products comply with it. One step in this direction has been made with the recent announcement to merge Open Information Model (OIM) [16] and Common Warehouse Metamodel (CWM) [17], the two standards for metadata representation and exchange proposed by Meta Data Coalition and OMG (a comparision between the two standards is given in [22]). Even if their metamodels cover extensive information of both general areas (like analysis, design, relational schemas) and data warehousing specific areas (like OLAP and data transformation), they also show some deficiencies. They fall short in providing different conceptual levels of metadata as e.g., suggested by Zachman [23] and abstractions for transformational metadata that would really enforce reusability and facilitate maintenance. Moreover, the metamodels do not handle some specific aspects which have been identified as demanding for data warehousing: security aspects, temporal evolution of structures and mechanisms for ensuring data quality (e.g., plausibility systems). But the main open question remains whether the CWM release resulted from the unification of the two will win acceptance in practice as the unique representation and exchange modality for metadata in data warehousing.

5 Perspectives for Metadata Integration

Since metadata integration is typically required for querying only, metadata could still be locally updated but it should be available to the whole enterprise. There are two integration alternatives one can take into account: either a (kind of) federation with an apriori defined global schema, or a more flexible way of querying where no integrated global schema is required.

5.1 Federation

Assuming the new CWM release will be adopted by all commercial products to be employed within an enterprise, the implementation of a *federated metadata management system* will be the most convenient solution. A typical federation architecture is depicted in figure 8. Metadata is captured and stored in *repositories* which have their own data models and store metadata according to a so-called *local schema*. They are managed by the *metadata manager* in accordance with a *global schema* covering all necessary metadata. The global schema provides the integrated view over the repositories. Obviously, the adopted standard or a self-defined refinement of it should be taken for representing the global schema. For each of the repositories a *wrapper* is needed for mapping local repository schemas to the global schema. In this way, metadata is still locally stored and repositories may be individually maintained with separate tools (like the standalone tool in figure 8), but the federation layer (i.e., the metadata manager and the wrappers) performs the virtual integration such that a trade-off between the advantages of centralization and those of local control is achieved.

Various *metadata producer and consumer tools* are using the metadata management system. Tools access information of repositories only through the metadata manager by means

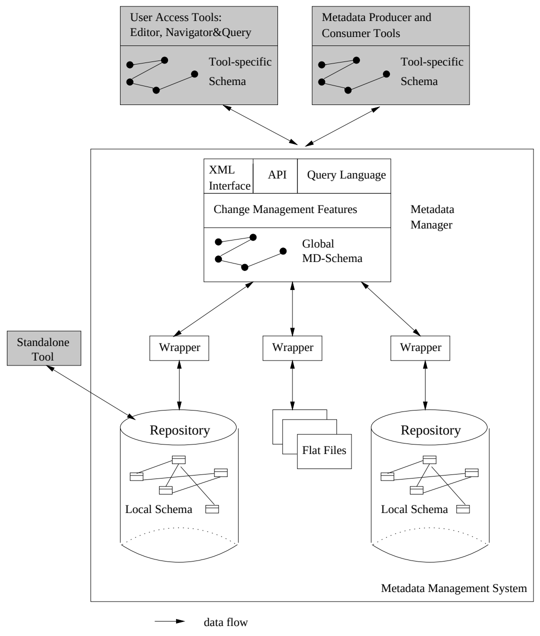


Figure 8: Metadata Management System and Tools Using it

of one of the alternatives offered by its interface layer: the API, the query language or the XML-interface. The interface layer passes operations to the metadata manager which in turn passes the operation to the corresponding repository storing the metadata of interest.

Note that often users and tools accessing the federation are interested in some kind of metadata only and thus this and only this metadata needs to be mapped into the global schema. Metadata of low level granularity needed for example to control standalone tools may be relevant only for these tools and not for the whole federation.

5.2 Single Access Point for Heterogeneous and Distributed Repositories

Metadata sources may be highly heterogeneous regarding their structure. Besides the known structured metadata collections like data dictionaries, tool repositories (e.g., PowerMart and BusinessObjects), other databases, Excel files, etc., there is a huge amount of unstructured or semistructured metadata collections in Word documents, pdf-files, e-mails, HTML and XML files, which should be managed as well. Since today's metadata management systems rely on traditional database technology, they require the management of metadata in structured repositories (usually databases). If metadata is available in semi-structured or unstructured sources, it has to be first imported into the structured repository for being uniformly managed later. Also the federation presented above is based on a rigid global schema defined beforehand. However, things should change in order to avoid the expensive costs of importing (or mapping) metadata to the structured repository and make use of the possibly more effective query capabilities for semi-structured and unstructured data of the original information sources. The approach of SINGAPORE (*Sing*le *A*ccess *Po*int for Heterogeneous and Distributed *Re*positories) [8] combines database and information retrieval technology and provides a unified interface to effectively query the metadata sources available. It is more flexible than a federation since it does not rely on a global schema but instead uses integration through the query language. However, this solution is not for free; homogenization of data is still required, and the exactness of query results often cannot be guaranteed.

Future challenges include the embedding of metadata management into enterprise-wide knowledge management systems. Such systems require the appliance of advanced approaches from various fields of computer science in order to manage and extract knowledge from the huge data and information assets available in enterprises.

6 Conclusion

Developing a successful metadata management system within an enterprise is not a trivial task. Besides technical issues like integrating and building the right structure for managing metadata, there are other problems which should not be underestimated: the costs of the unavoidable initial manual documentation (including connections between metadata elements) and the everlasting problems of maintaining the system. The expensive efforts to build a metadata management system are in vain if the metadata available is not actual and correct; users lose their trust and the metadata management system becomes useless.

This paper gave an overview of the topic of metadata management for data warehousing covering both, requirements and the state of the art. We gave examples of good solutions for employing metadata to support rather narrow tasks like querying OLAP views by using common business terms. But we also revealed problems of existing tools for metadata management aiming at broader tasks like generally supporting the implementation of ETL process. These problems are inherently closely related to still open software engineering tasks like finding of appropriate abstractions for improving reusability and maintenance of implemented software. In the same context, the ideal representation of transformational metadata on higher levels up to the business rules and their links to the lower implementation levels is still an open issue.

Regarding metadata integration, efforts are underway to establish standards for a global schema to manage all metadata in the data warehouse system and to provide the infrastructure for metadata exchange between tools. However, the future will show whether the new release of the CWM standard will win broad acceptance among vendors in a way that enables the enterprise-wide metadata integration. According to a recent Gartner Group note [10], it will take some time until CWM will be eventually considered.

In summary, a lot of promising work exists in the area of metadata management for data warehousing but its successful, large-scale use still needs more research to be done.

References

[1] P.A. Bernstein. Repositories and object oriented databases. *SIGMOD Record*, 27(1):88 –96, March 1998.

[2] P.A. Bernstein. Panel: Is generic metadata management feasible? In *Proc. of the 26th Intl. Conf. on Very Large Databases (VLDB)*, Cairo, Egipt, September 2000.

[3] P.A. Bernstein and U. Dayal. An overview of repository technology. In *20th International Conference on Very Large Data Bases (VLDB '94)*, pages 705 –713, Santiago de Chile, Chile, 1994.

[4] P.A. Bernstein, A.Y. Levy, and R.A. Pottinger. A vision for management of complex models. Technical report MSR-TR-2000-53, Microsoft Research, Microsoft Corporation, June 2000. http://www.research.microsoft.com/pubs.

[5] P.A. Bernstein and E. Rahm. Data warehousing scenarios for model management. In *Proceedings of the 19th Intl. Conf. on Entity-Relationship Modelling.* Springer, October 2000.

[6] M.H. Brackett. *The Data Warehouse Challenge.* Wiley, 1996.

[7] B. Devlin. *Data Warehouse: from Architecture to Implementation.* Addison Wesley, 1997.

[8] R. Domenig and K.R. Dittrich. A query based approach for integrating heterogeneous data sources. In *Proceedings of the Ninth Intl. Conf. on Information and Knowledge Management (CIKM)*, Washington DC, USA, November 2000.

[9] Gartner Group. *IT Metadata Repository Magic Quadrant Update 2001.* Research Note 25 August 2000, M-11-8336.

[10] Gartner Group. *OMG's Common Warehouse Metamodel Specification.* Research Note 28 July 2000, E-11-4175.

[11] W.H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, 1996.

[12] M. Jarke and D. Shasha, editors. *Information Systems; Special Issue on Meta-Modelling and Methodology Engineering*, volume 24. Pergamon, April 1999.

[13] J.U. Kietz, R. Zücker, and A. Vaduva. MINING MART: Combining case-based-reasoning and multistrategy learning into a framework for reusing KDD-applications. In *Proc. of the 5th Intl. Workshop on Multistrategy Learning (MSL 2000)*, Guimaraes, Portugal, June 2000.

[14] R. Kimball, L. Reeves, M. Ross, and W Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses.* Wiley, 1998.

[15] D. Meyer and C. Cannon. *Building a Better Data Warehouse.* Prentice Hall, 1998.

[16] Microsoft. *Open Information Model.* http://msdn.microsoft.com/repository/OIM.

[17] Object Management Group (OMG). *Common Warehouse Metamodel (CWM) Specification*, 2000. OMG Document ad/00-01-01, ad/00-01-02, ad/00-01-03, ad/00-01-11, also see http://www.cwmforum.org/.

[18] M. Osley. Development of a browser for accessing data warehousing metadata. Diploma thesis, University of Zurich, Dept. of Information Technology, June 2000.

[19] M. Staudt, A. Vaduva, and T. Vetterli. Metadata management and data warehousing. Technical Report 21, Swiss Life, Information Systems Research, July 1999. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-99/ifi-99.04.pdf.gz.

[20] M. Staudt, A. Vaduva, and T. Vetterli. The role of metadata for data warehousing. Technical Report 99.06., University of Zurich, Dept. of Information Technology, September 1999. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-99/ifi-99.06.ps.gz.

[21] A. Vavouras, S. Gatziu, and K.R. Dittrich. Modeling and executing the data warehouse refreshment process. Technical report 2000.01, University of Zurich, Dept. of Information Technology, January 2000. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-2000/ifi-2000.01.pdf.

[22] T. Vetterli, A. Vaduva, and M. Staudt. Metadata standards for data warehousing: Open Information Model vs. Common Warehouse Metamodel. *ACM Sigmod Record*, 29(3), September 2000.

[23] John A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.