# Modeling and Executing the Data Warehouse Refreshment Process

**Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich**

Technical Report **2000.01**
January 2000

Department of Information Technology, University of Zurich
{vavouras, gatziu, dittrich}@ifi.unizh.ch

**Abstract**

Data warehouse refreshment is often viewed as a problem of maintaining materialized views over operational sources. In this paper, we show that the data warehouse refreshment process is a complex process comprising several tasks, e.g., monitoring, extracting, transforming, integrating and cleaning operational data, deriving new data, building histories and loading the data warehouse. We propose a novel approach for defining and executing the refreshment process based on specifications stored in an object-oriented metadata repository. Our approach considers the multidimensional character of OLAP data and can be used in conjunction with various operational sources and target data warehouses.

## 1 Introduction

The topic of *data warehousing* [6, 14, 17, 35, 36, 40] comprises architectures, algorithms, models, tools, organizational and management issues for integrating data from several operational systems in order to provide information for decision support, e.g., using data mining techniques or OLAP (on-line analytical processing) tools. Thus, in contrast to operational systems which contain detailed, atomic and current data accessed by OLTP (on-line transactional processing) applications, data warehousing technology aims at providing integrated, consolidated and historical data. The *data warehouse* (DWH) can be realized either as a logical (virtual) view of the data physically stored in the various operational systems, or as a separate database that stores integrated operational data (the latter being the most typical case). A *data warehouse system* (DWS) includes the data warehouse and all components responsible for building, accessing and maintaining the DWH.

Figure 1 shows a typical data warehousing environment. Data from the enterprise's (internal) operational systems (e.g., database systems, flat files, etc.) as well as external data (e.g., WWW data, demographic and statistical data, etc.) are integrated into a large, consistent data repository, the DWH (*data acquisition phase*). Specialized modeling concepts such as the multidimensional model, the star and the snowflake schema as well as storage structures like bitmap ([7], [8], [39], [41]) and join indices ([23], [21,]) are used to store extracted data in the DWH and/or in particular *data marts* (*data storage phase*). A data mart is a selected part of the data warehouse which contains simple replicas of warehouse partitions or

data that has further been summarized or derived from base warehouse data. A data mart is usually designed to meet the specific needs of a company's department or geographical region. Finally, during the *analysis phase*, users access the DWH or data marts using various tools and applications as shown in Figure 1.

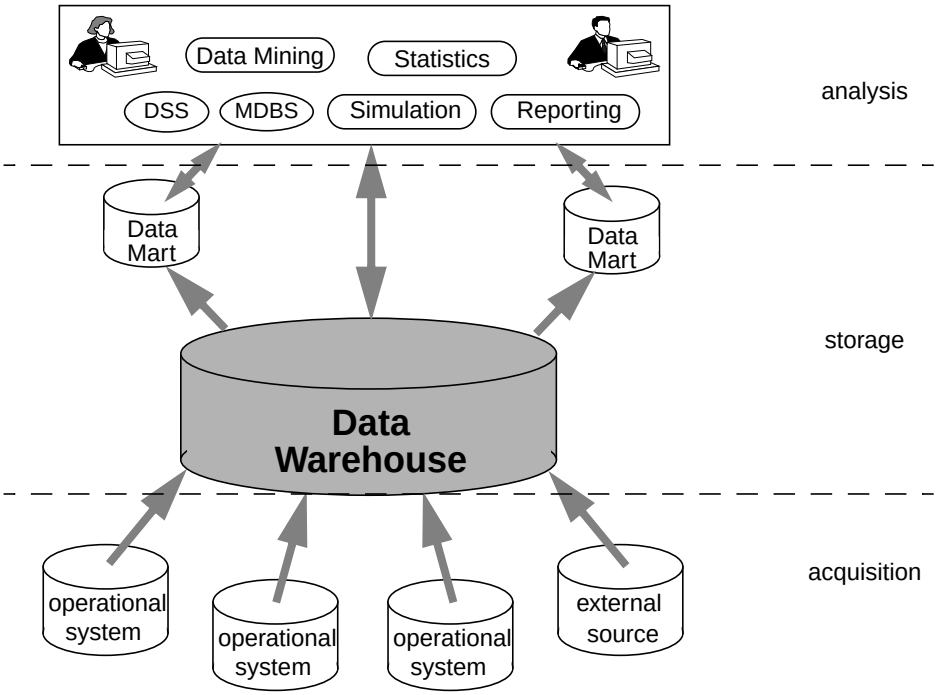


*Figure 1 Common data warehousing environment*

Implementing a concrete data warehouse solution is a complex task, comprising two major phases. In the DWS *configuration* phase, the DWS designer must determine (according to user requirements for information) the desired operational data, the appropriate operational sources, the way data will be extracted, transformed integrated and stored, and how the DWH data will be accessed during analysis. After the *initial loading* (the first load of the DWH according to the DWH configuration), during the DWS *operation* phase, warehouse data must be regularly *refreshed*, i.e., modifications of operational data since the last DWH refreshment must be propagated into the warehouse (and into data marts) such that warehouse data reflect the state of the underlying operational systems. Besides DWH refreshment, DWS operation includes further tasks like archiving and purging of DWH data or DWH monitoring.

**Data Warehouse Refreshment Issues.** In this section, we give an overview of the particular issues related to the data warehouse refreshment process and derive the requirements for an appropriate data warehouse refreshment system. The issues include 1) the tasks that must be accomplished during warehouse refreshment, 2) the process of incremental

refreshment of warehouses, and 3) temporal issues regarding the initiation of the refreshment process.

The first issue concerns the question of *which tasks* must be executed during warehouse refreshment. Operational data resides in a wide range of information systems which run on diverse platforms and have a variety of representations and formats, due to differences in data models as well as in understanding and modeling data. Dealing with the heterogeneity of the integrated sources in a data warehousing environment implies *extracting* operational data from diverse sources and *transforming* it into a common format. During the refreshment process, operational data must be *integrated* and "homogenized" according to a common, global data model. Furthermore, structural and semantic differences across operational sources must be reconciled. Data extracted from operational systems often contains errors, can be inconsistent, unreadable or incomplete. Therefore, it must be *cleaned* before being loaded into the data warehouse. Particularly, for the integration of external data, cleaning is an essential task in order to get correct data into the DWH and improve the quality of warehouse data. In most cases, additional activities are required (*completion*) before loading data into the DWH, for example

- provide a common level of detail for data from different sources and build aggregations,
- calculate derived data,
- add time-related information (usually by timestamping data with the update date).

A distinguishing characteristic of data warehouses is the temporal character of warehouse data, i.e., the management of *histories* over an extended period of time. Historical data is necessary for business trend analysis which can be expressed in terms of analysing the temporal development of real-time data. For the refreshment process, maintaining histories in the DWH means that either periodical snapshots of the corresponding operational data or relevant operational updates are propagated and stored in the warehouse, without overriding previous warehouse states.

The second issue is *how* to execute the refreshment process. The first option is to perform a complete reload of operational data, i.e., extract the last (complete) state of operational systems, execute the steps described above, and load the data into the DWH. The second one is to refresh the DWH incrementally, i.e., detect relevant updates in operational systems between two refreshment points of time, execute the above mentioned tasks only for changed data, and update the DWH accordingly. There are several reasons for refreshing a DWH incrementally in contrast to full reloads. First, DWH volumes can reach hundreds of gigabytes or even several terabytes, and will grow even more in the future. At the same time, the demand for more up-to-date data increases such that a DWH must be up-

dated more often. Under these circumstances, performing a complete reload of operational data is a very time-consuming task that becomes unacceptable. Second, since a DWH typically stores (besides basic operational data) a large amount of derived and aggregated data, detecting and propagating only updates of basic data will also significantly reduce the time for computing derived and aggregated data. Finally, there is an important coherence between incremental refreshment and history management. Detecting updates of operational data in order to refresh the warehouse incrementally enables at the same time the correct maintenance of histories in the warehouse. In contrast, discarding updates between two refreshment points of time and performing periodical, complete reloads of warehouse data leads to a (at least partial) loss of histories (this important distinction will be described in Section 4.1).

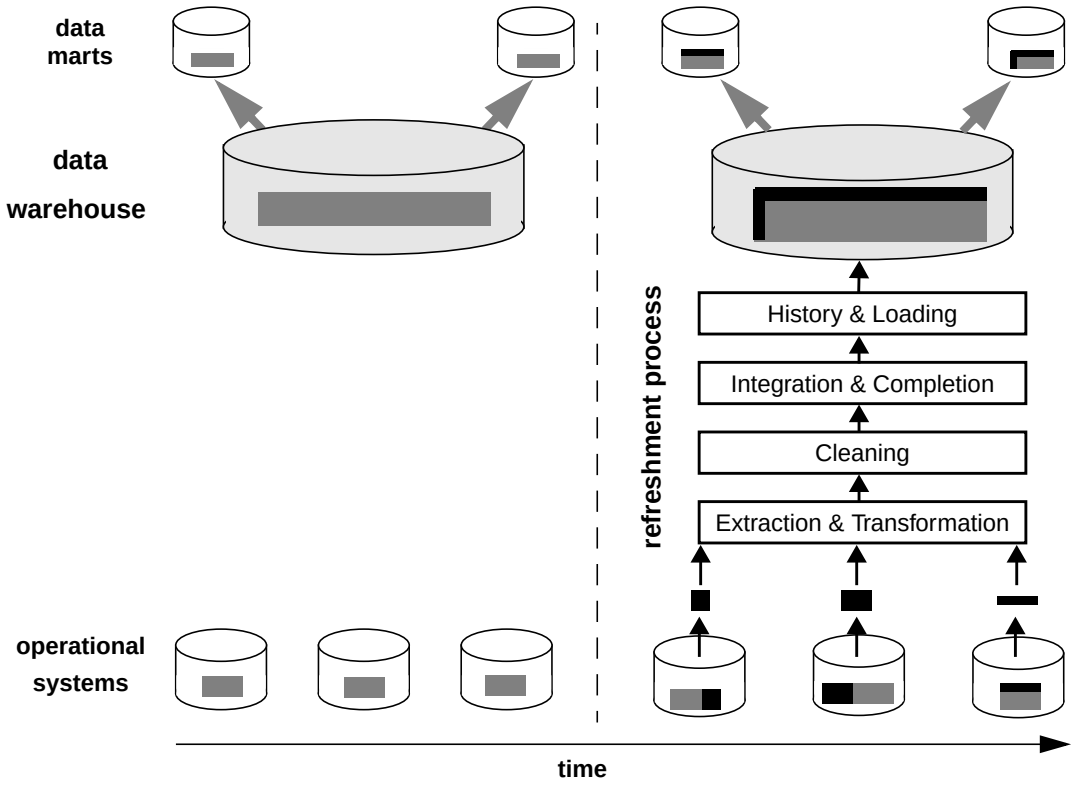


*Figure 2 Data warehouse refreshment process*

The last issue of the refreshment process concerns the question of *when* to execute the refreshment process. The DWS should provide a variety of options for defining the begin of the refreshment process, for example

- periodically, e.g., every day at 19:00, every Friday at 24:00, at the end of every month, etc.,
- depending on changes of operational data, e.g., when a certain attribute value limit in one or more operational systems has been reached,

- on explicit request of a DWH-user or the data warehouse administrator.

Figure 2 illustrates the tasks related to the refreshment process. After the initial loading of the DWH, operational updates are propagated into the DWH and data marts at certain points of time. For modified data, the particular tasks described above are executed and applied to the previous warehouse state such that the DWH represents a consistent view of the integrated operational systems.

**Related Work.** The topic of DWH refreshment so far has been investigated in the research community mainly in relation to techniques for maintaining materialized views [13, 27]. In these approaches, the DWH is considered as a set of materialized views defined over operational data. Thus, the topic of warehouse refreshment is defined as a problem of updating a set of views (the DWH) as a result of modifications of base relations (residing in operational systems). Several issues have been investigated in this context. For example, [1, 15, 20, 38, 45, 46, 47] present various algorithms for the consistent, incremental maintenance of a single view or a set of views derived from multiple data sources. [16, 25] consider the problem of view self-maintenance, i.e., maintaining views without accessing all base data. In [30], a solution for maintaining externally materialized views is presented, i.e., for the case where access to the base data and the view definition but not to the materialized view is possible. [29] provides algorithms for answering queries with grouping and aggregation using materialized views. [31] and [42] deal with the issue of warehouse configuration, i.e., selecting a set of views to materialize in a DWH. [43] presents a framework for maintaining temporal views over non-temporal source relations. In [44], the problem of how to preserve the view definition under schema-level changes of operational systems is addressed. All these approaches focus on the problem of propagating data updates from operational systems to the warehouse. However, as discussed above, update propagation using view materialization techniques is only one of the relevant tasks of the data warehouse refreshment process.

Moreover, most of these approaches presume a “relational environment”, i.e., they assume that operational sources as well as the target warehouse are relational systems, which is not always true. Operational sources can be any kind of database systems (relational, hierarchical, network, object-oriented etc.), but also ISAM files or other proprietary formats. On the other hand, although the majority of projects makes use of relational technology for implementing the data warehouse, there is an increasing demand for building data warehouses for “non-standard” applications like genomic [9], geographic or Web warehouses [3]. Likewise, it is an interesting issue to integrate data from these application domains into “traditional” warehouses, e.g., to combine customer data of a company with spatial or Web data. In both cases, refreshment approaches using materialized view tech-

niques are too limiting because they make it difficult to model and manage non-standard types such as spatial data types.

In our approach, object-oriented modeling concepts are used to describe the complex structure of operational data and the multidimensional character of warehouse data. Besides these structural (static) aspects, the dynamic aspects of the refreshment process, i.e., the execution of the particular steps of the refreshment process, can be treated in an integrated way by using methods that operate on the above mentioned data structures. The specification of the refreshment process is stored in an object-oriented metadata repository which can be extended in a simple way. The execution of the refreshment process is based on these specifications. Complex mappings and transformations as well as various ways for defining derived data are supported. History building, key management, incremental refreshment and mappings for several typical DWH design methods (e.g., star and snowflake schemas) are provided using OID's and object-oriented modeling techniques respectively.

Thus, in contrast to existing refreshment approaches, the main goal of our approach is to introduce a flexible middleware architecture that

- provides solutions to all refreshment issues discussed above,
- can be used independently of how warehouse data is stored persistently, i.e., without making any assumptions about the DBMS or other data management system used for storing the warehouse data, and
- can be used in data warehouse environments consisting of a wide variety of heterogeneous operational sources (various database systems, flat files, etc.).

The approach developed in the context of our project SIRIUS (**S**upporting the **I**ncremental **R**efreshment of **I**nformation Wareho**u**se**s**) is, to the best of our knowledge, the first one which deals with the warehouse refreshment problem without restricting the DWH to a set of materialized views over operational data. The same fundamental idea is followed by the position paper in [4] which discusses the particular steps of the refreshment process and the differences to the view maintenance problem in a similar way. The focus of their future work is the design and implementation of the refreshment process by using workflow system technology.

The paper is organized as follows. Section 2 presents a running example of a data warehouse application for a mail-order business. Section 3 presents the basic constructs for the specification of the refreshment process. Furthermore, we discuss the SIRIUS data model and the transformation steps that can be defined in SIRIUS for performing the particular tasks of the refreshment process. Section 4 presents the SIRIUS system architecture, and shows how the various components cooperate in order to refresh a DWH. Section 5 describes the most important steps during execution of the refreshment process and Section 6

summarizes the main features of our approach. The paper concludes with a presentation of our current and future work.

## 2 A Running Example

In our running example, a mail-order business aims at building a DWH to support decision making for different user groups and departments (illustrated in Figure 3).

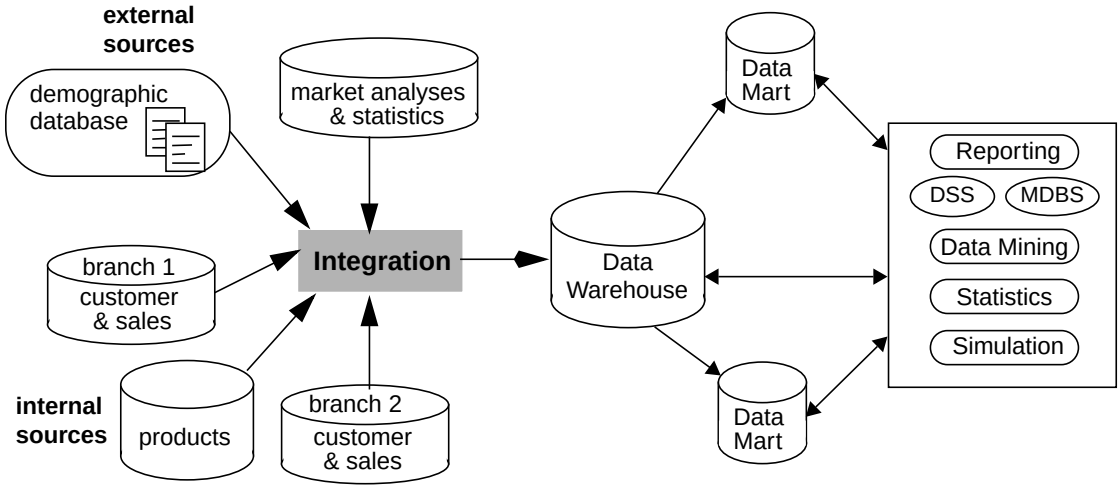


*Figure 3 The Data Warehouse System for the Mail-Order business*

The central DWH stores products, sales and customer data that is drawn from various information systems. We assume that each company branch locally stores its own sales data (e.g., sold items, quantity, price, date, etc.) and customer data (e.g., name, address, age, marital status etc.). All branches access the same product catalog which stores information like product number, description, price, marginal return etc. and is managed by the central marketing division. Besides, data delivered from the company branches, external data like demographic data (in order to classify all customers according to several characteristics) and several market analyses (e.g., about market shares and market demand) will be integrated into the DWH.

We assume that before starting with the configuration of the refreshment process, a conceptual schema of the warehouse data has been defined using one of the conceptual models for DWH design recently proposed in the literature [12, 28, 33]. Figure 4 shows a part of the conceptual *dimensional fact schema* of our running example using a graphical notation similar to the one proposed in [12]. The basic components of a dimensional fact schema are facts, dimensions and hierarchies. *Facts* represent business events (*measures*) to be analysed, e.g., sales, shipments, policy effects, phone calls, etc. *Dimensions* include the analysis criteria, i.e., the different points of "view" used for analysing facts. Our example

schema consists of the sales fact and the four dimensions product, customer, time, branch. Dimensions have usually defined on them *hierarchies* of attributes that specify aggregation levels and hence the granularity of viewing data. For example, customer->location->region->country is a hierarchy on the dimension customer that can be used for querying sales data in diverse aggregation levels by drilling down or rolling up. In Figure 4, *hierarchy attributes* like location and region are represented by empty circles, whereas *non-hierarchy attributes* (i.e., dimension attributes that do not participate in a hierarchy) like date_intro or supplier are represented by filled vertices.

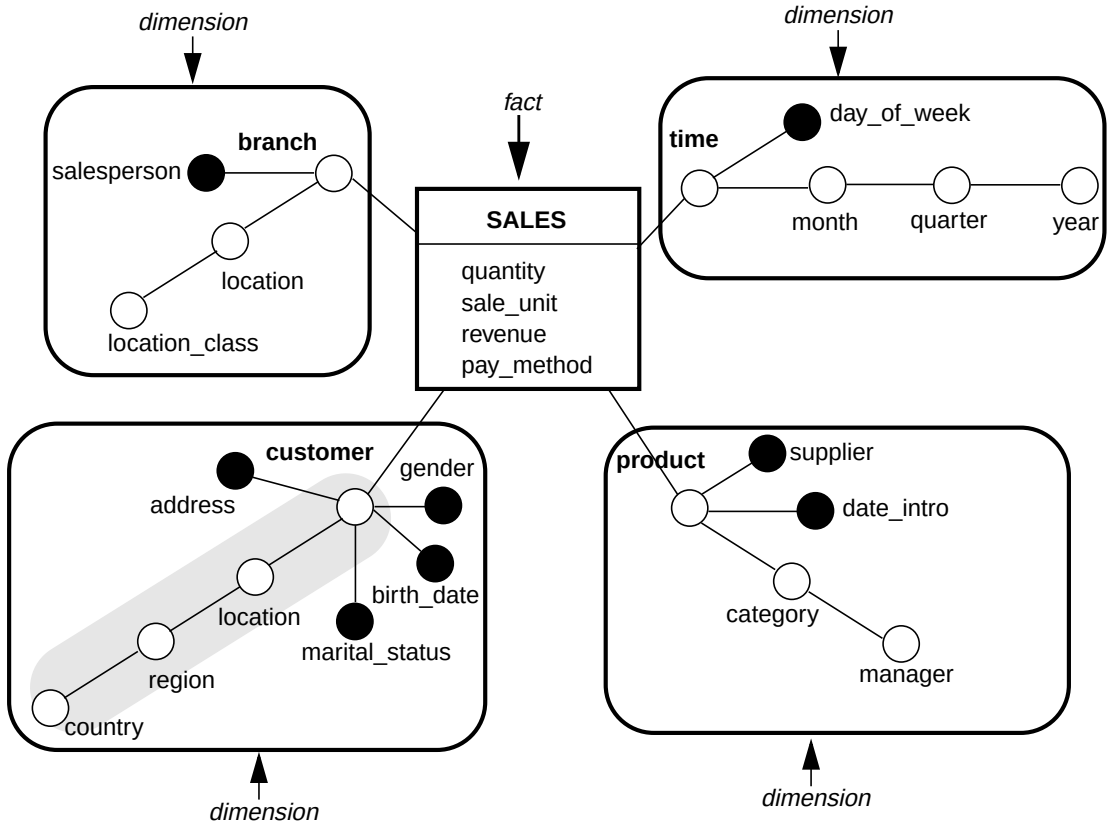


*Figure 4 Conceptual dimensional schema*

# 3 Modeling the Refreshment Process in SIRIUS

As described in the introduction, the main goal of our approach is to support the modeling and execution of the refreshment process at an intermediate layer between operational sources and the target data warehouse. On the operational side, capturing the heterogeneity of operational sources necessitates concepts for the definition of a uniform "global" representation of operational data to be integrated. In our approach, modeling the structure of operational data is performed by defining a *global schema* using the SIRIUS global data model (described in Section 3.1). The global schema is the basis for executing the refresh-

ment process, i.e., for integrating operational and external data, performing the particular refreshment tasks and loading the DWH. On the other hand, we assume that a *storage schema* is used for defining the structure of the DWH data as it is stored by the warehouse DBMS. For example, the warehouse DBMS can be a relational or a multidimensional DBMS, and the storage schema a star, snowflake or a multidimensional schema. By using a global schema and defining the appropriate mappings to operational schemas as well as the warehouse (storage) schema, the SIRIUS approach can be used in various environments and independently of how warehouse data is persistently stored.

Figure 5 gives an overview of the schema architecture proposed in SIRIUS. For each operational source a source data representation (the database schema, in case of database systems) describes the structure of the operational data. Parts of each source data representation contribute to the global schema. The various SIRIUS components which are responsible for the execution of the refreshment process (summarized as the Data Warehouse Refresh Manager, DWRM, in Figure 5) operate on top of the SIRIUS global schema.

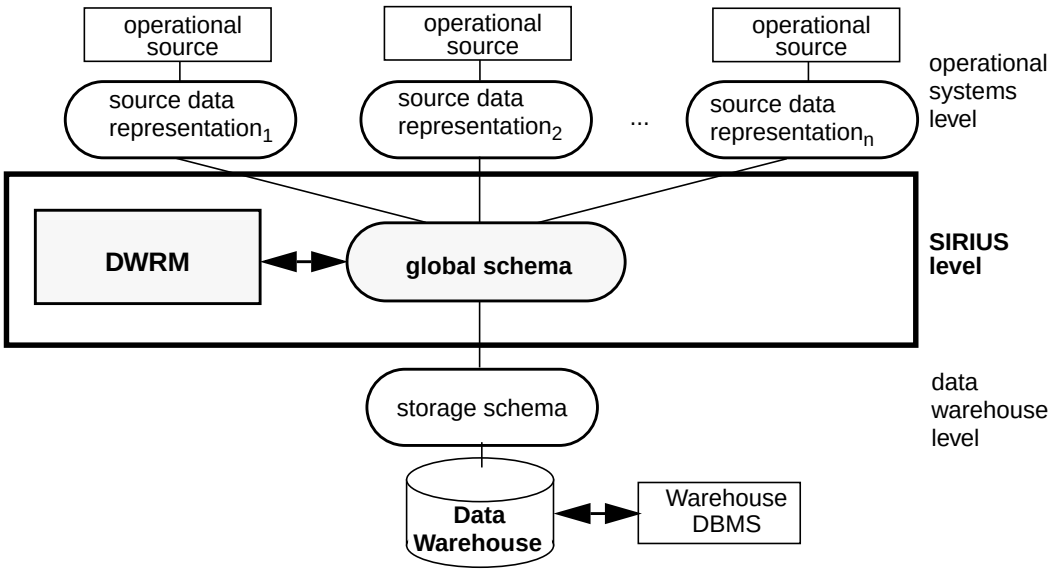


*Figure 5 Schema architecture in SIRIUS*

For the *definition of the refreshment process in the SIRIUS level*, three basic kinds of information are needed:

a. the specification of the global schema

b. a description of operational data and sources to be integrated

c. the mapping between a. and b. including all refreshment tasks.

The specification of the refreshment process in SIRIUS is based on the meta model illustrated in Figure 6 in UML notation. The central element of the meta model is the speci-

fication of the *global schema* which is the basis for executing the refreshment process. In order to embody the multidimensional character of OLAP data, the SIRIUS global schema must be specified by the DWH administrator in terms of *fact* and *dimension classes*. Fact classes contain a set of *measures* and are associated with several dimension classes. Dimensional classes consist of dimensional attributes which can have *hierarchies* (by defining appropriate derivations as described in Section 3.1) and *histories* (Section 4.4) associated with them. Besides defining hierarchies, *derivations* can be used in order to "enrich" operational data, perform aggregations, define rule-based calculations and integrate external data (e.g., by classifying customers using a demographic database).

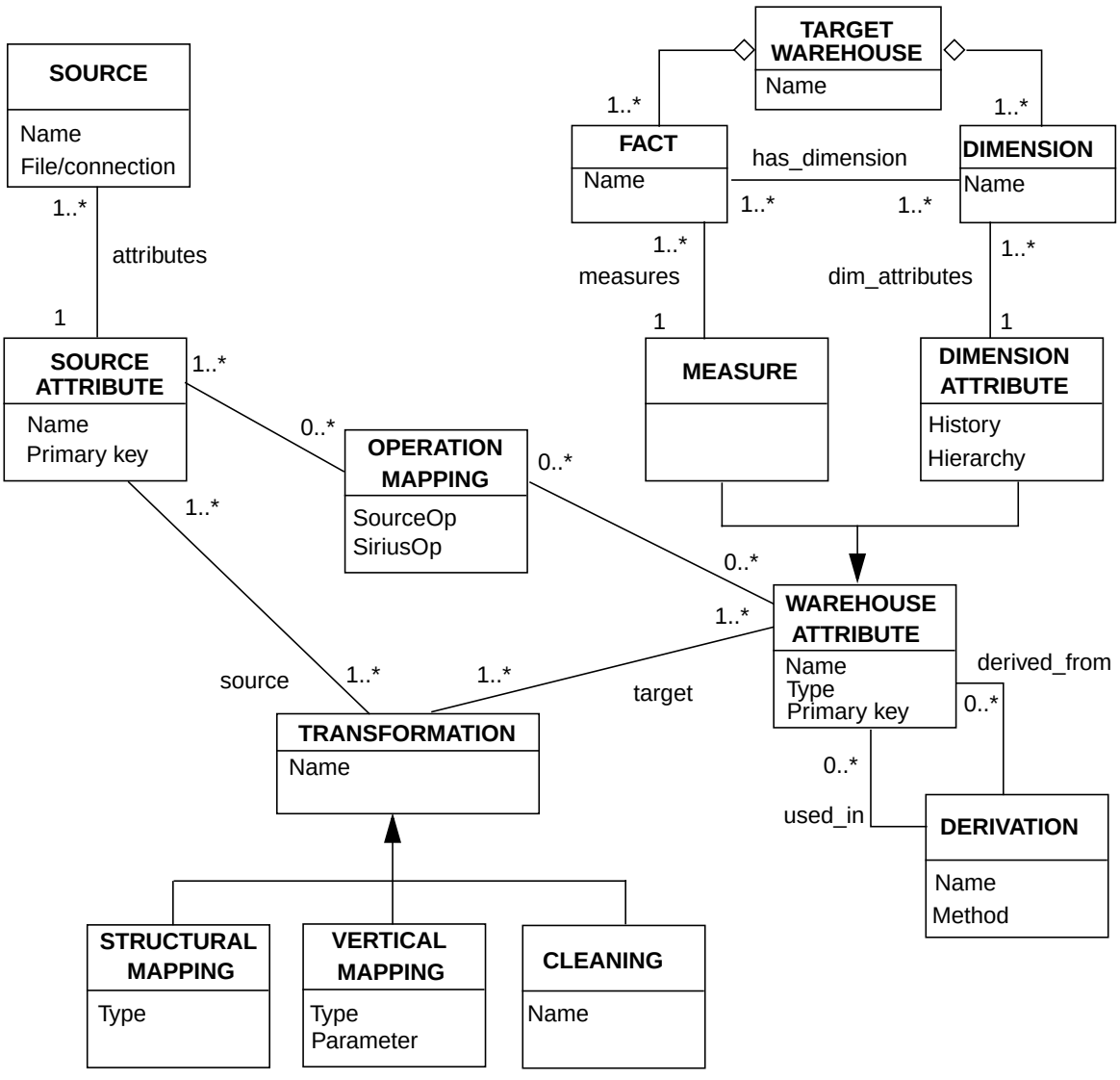


*Figure 6 Meta model for defining the refreshment process*

Furthermore, for defining various kinds of transformations between source and warehouse attributes, SIRIUS provides the following options:

- *Structural mappings* define the way attributes from a single source map to warehouse attributes. Section 3.3 describes various structural mappings provided by SIRIUS including 1:1, 1:n, n:1 and method mappings.
- In contrast to structural mappings which provide a kind of "horizontal" integration, *vertical mappings* can be used in order to define several forms of integrating groups of attributes from several sources and mapping them to warehouse attributes (i.e., structural mappings are defined at the attribute level, whereas vertical mappings are defined at the entity level). Various kinds of vertical mappings like union, difference, selection, prioritization are supported by SIRIUS (see also Section 3.4).
- Operational data *cleaning* can be performed either by using predefined SIRIUS methods (e.g., for eliminating duplicates) or by integrating specialized tools (Section 3.5).

Finally, the definition of so-called *operation mappings* between source and warehouse attributes is used for refreshing the warehouse incrementally by assigning local source operations to SIRIUS operations (Section 3.6).

Defining a concrete refreshment process means that the DWH administrator defines an instance of the meta model according to Figure 6. This instance is stored in the metadata repository of SIRIUS and is used for

- the generation of the global schema according to the SIRIUS global data model described in the next section,
- the execution of the refreshment process, and
- the documentation and administration of a concrete warehouse configuration.

## 3.1 The Global Data Model

The SIRIUS *global data model* is based on the object-oriented data model of the Object Database Management Group (ODMG) standard [5]. Similar to other projects focusing on data integration in heterogeneous environments [22, 26, 2, 19], we exploit the rich semantic expressiveness of object-oriented data models for representing the structure of operational data in the SIRIUS intermediate layer residing between operational sources and the DWH. Moreover, choosing the ODMG model as the global model of SIRIUS allows us to take advantage of further features like object identity.

The SIRIUS global model provides the basic constructs of the ODMG model like types, objects, attributes and relationships, operations, etc. In the ODMG object model, attributes may have simple or complex values (e.g., sets or references to other objects). In our approach, we additionally distinguish between *basic* and *derived* attributes [11]. Values of ba-

sic attributes are assigned directly from the values of the corresponding attributes in the various operational systems. On the other hand, derived attributes represent information that is not available (or is not explicitly stored in order to avoid redundancy) in operational systems and is added to the DWH after integrating data from the operational systems.

A further feature of our global data model is the notion of so-called *history* attributes [11]. History attributes can be used for modeling the temporal character of the data stored in the warehouse, i.e., a warehouse maintains *histories* of data over an extended period of time. Further details about the way history attributes are treated are given in Section 4.4. A part of the global schema of our running example is shown in Figure 7.

```
fact class SALES {
 attribute integer quantity;
 attribute date sales_date;
 attribute integer saleprice_unit;
 attribute string pay_method;
 derived attribute integer customer_age;
 derived attribute integer branch;
}

 dimension class PRODUCT {
 attribute integer prod_id;
 attribute string supplier;
 attribute string date_intro;
 derived attribute string date_withdrawal;
 derived attribute string prod_category;
 derived attribute string manager;
 history attribute string price_unit;
}

dimension class CUSTOMER {
 attribute integer cust_id;
 attribute string last_name;
 attribute string first_name;
 attribute date birth_date;
 attribute string marital_status;
 history attribute string street;
 history attribute string postal_code;
 history attribute string location;
 derived attribute string region;
 derived attribute string country;
 derived attribute integer income_bracket;
 derived attribute integer num_children;
 derived attribute date customer_since;
 } ;
```

*Figure 7 Global schema example*

During the configuration phase, the global schema is derived from the instance of the meta model defined as discussed in Section 3. For example, a derived attribute of the global schema is defined for each **WAREHOUSE ATTRIBUTE** that has defined a **DERIVATION** according to the meta model of Figure 6. During the execution of the refreshment process, built-in operations are used by SIRIUS for instantiating the global schema with operational updates, i.e., creating objects, retrieving and updating the attributes of an object. Transformations from operational data to warehouse data are performed using methods defined over global schema classes.

## 3.2 Derived Attributes

Producing “new information” in the DWH by defining derived attributes is one of the key issues in a data warehousing environment that can improve the quality of analysis results. In a typical data warehousing environment, several types of derived attributes can be defined and must be processed during warehouse refreshment:

- Refreshing hierarchy attributes of dimensions at the SIRIUS level is the most important application of derivations. Hierarchies like `customer->location->region->country` of the `product` dimension can be expressed in our approach by defining the appropriate derived attributes (Figure 7) and methods for calculating the values of derived attributes. During the execution of the refreshment process, hierarchy attribute values corresponding to updated basic attribute values are processed and loaded into the DWH by SIRIUS.
- Since access to warehouse data is mostly read-oriented, data warehouses often store data in different levels of detail at the same time, thus allowing users to drill down and roll up through data faster (in a relational warehouse design, this is known as a fact constellation schema). In our example, if branches deliver sales data at the level of line items, the refreshment system should further summarize data to daily, weekly and monthly sales and propagate it into the warehouse. Furthermore, if branches deliver sales data in different levels of detail (e.g., line items and daily sales), data must be brought into a common level of detail during warehouse refreshment.
- External data (e.g., from demographic and statistical databases) are often used to enrich operational data and improve the quality of analysis results. In our example, customer operational data are completed by the derived attributes `income_bracket` and `num_children` which we assume to be provided by a demographic database.
- Operational sources often use various key systems and numbering schemas. In order to preserve object identification when integrating data from different systems, a new uniform key system must often be introduced. For example, branches could use similar numbering schemas for identifying customers. Integrating customer data into the DWH requires the introduction of a new (global) primary key (e.g., `global_cust_id=cust_id&branch_id`) in order to ensure the identification of customers in the DWH. Derived attributes can thus be used to realize even complex key synchronization steps.
- Finally, derivations can be used to define a wide variety of rule-based calculations (e.g., set `customer_since` equal to the earliest date for which a sale record with the related customer can be found), complex arithmetic operations, date functions (e.g., calculate the `day_of_week`, `month`, `quarter` and `year` out of the `sales_date`), timestamping data with the refreshment time, and marking extracted operational data with a source identifier (attribute `branch` in `sales`).

In all these cases, using our object-oriented approach has a twofold benefit. First, common used derivations are implemented in the SIRIUS library and can be used during the

refreshment process specification. Second, existing derivations can be extended by users to define new kinds of derivations by inheriting and overriding existing methods, thus facilitating the maintenance and extension of the warehouse refreshment process. Thus, compared to the limited facilities provided by view-based approaches (due to restrictions of SQL), derivations provide a powerful concept for improving the value of warehouse data.

## 3.3 Structural Mappings

For each operational source and the attributes extracted from it, SIRIUS provides a *structural mapping*, i.e., attributes of the integrated sources are mapped to global attributes. A structural mapping can be defined for one attribute or a group of attributes extracted from (exactly) one operational source. SIRIUS verifies the specification of the refreshment process such that for each attribute of the global schema (except for derived attributes), a structural mapping exists (while, for example, not every global attribute has a vertical mapping defined on it). The syntax for a *structural mapping specification* has the form

```
<local attribute name, global basic attribute name, [mapping]>
```

The last (optional) part of a structural mapping specification can be used to define diverse kinds of mapping, i.e., 1:1, 1:n, n:1 and mapping methods. The default case is a 1:1 mapping between local and global attributes. In our example, if we assume that the information about the marital status of customers is modeled in the same way as in our global schema, the mapping to the global attribute `marital_status` will be a 1:1 mapping. Further (predefined) structural mappings provided by SIRIUS are 1:n (extract n global attributes from 1 local) and n:1 (merge n local attributes into 1 global). For instance, some systems store address information in a single, aggregated field. Mapping this information to the individual global attributes `last_name`, `first_name`, `address`, `postal_code`, and `location` requires splitting the extracted local attribute and building the individual global attributes (1:n mapping). In other cases, more complex mappings are needed and can be defined as a *mapping method* which is executed during warehouse refreshment. For example, product prices in different currencies (integrated from different branches) can be converted into a common currency by defining an appropriate method that performs a simple arithmetic operation. Figure 10 shows an example of structural mappings for the class `PRODUCT` of our example global schema.

## 3.4 Vertical Mappings

A vertical mapping assigns values of operational attributes from more than one source to one global attribute. SIRIUS provides several kinds of vertical mappings for the integration of operational data from various sources. The most common example is the *union* opera-

tion. In our example, detail sales data that is separately stored by each company's branch must be merged to build the company's total sales data. Moreover, the *intersection* (e.g., process only product groups sold by all branches) and *difference* operation are supported.

Since not all source can deliver operational data at the desired detail level, *filtering* (selection) of extracted data is often needed. The filtering operation allows the definition of a condition on attribute values. Only entities (at the SIRIUS layer: objects) that satisfy this condition are further processed and loaded into the warehouse. For example, we could filter sales data such that only transactions of customers living in certain regions or sales exceeding a certain value limit are loaded into the warehouse.

Another type of vertical mapping, the *prioritization* operation, is used in particular in data warehousing environments where semantically identical data (e.g., customer data) is stored in several operational systems that differ in the level of data consistency, actuality or correctness. Integrating data from these sources and identifying identical records, one would like to be able to specify the source obtaining the priority for delivering the warehouse data.

## 3.5 Data Cleaning

Data extracted from operational systems often contains errors, and must first be cleaned before loading it into the data warehouse. Data values from operational systems can be incorrect, inconsistent, incomplete or in an unreadable format. Particularly, for the integration of external data, cleaning is an essential task in order to get correct data into the data warehouse. Data cleaning includes tasks like

- reconciling semantic differences between multiple sources, due to the use of homonyms (same name for different things), synonyms (different names for same things) or different units of measurement,
- transforming and enriching data to correct values using rules related to data semantics (in contrast to the transformations related to data structures defined by means of structural and vertical mappings),
- identifying and eliminating duplicates and irrelevant data.

In our example, customer's postal codes (or further address information) must be validated using auxiliary (external) address databases. Since address information will be further used as a criterion to classify customers, and analysis of customer profiles is based on this information, it is very important to get correct values for this data. For the attribute `marital_status`, different synonyms used in the operational sources (e.g., '1 / 2', 's / m' and 'single / married') must be cleaned and treated in a uniform way. Similar, duplicates of cus-

tomer data (the same customer has purchased a product from different branches) must be eliminated.

SIRIUS provides a set of simple cleaning methods for identifying duplicates from different sources using matching of key and non-key attributes. Violation of simple, domain-specific business rules (e.g., no negative product prices or attribute value's range checking) can be implemented by defining appropriate methods on the global schema. Since data cleaning is not the main focus of the project, we only provide an interface for integrating specific cleaning tools like InfoRefiner [24], Centric [10], and the Trillium Software System (R) [32].

### 3.6 Operation Mappings

Besides differences related to the structure of operational data, operational systems also differ in the way update operations (insert, delete, update) are executed. For example, changing the price of a product could be performed in a certain operational source by updating the value of an existing record, whereas in another source a new record is inserted. In our global schema example of Figure 7, this update corresponds (in both cases) to an update operation of the attribute `price_unit`. Similarly, removing a product from the product line corresponds to the deletion of the appropriate product record (or tuple). In the SIRIUS level, since the DWH maintains histories of data, the same information is stored by introducing the attribute `date_withdrawal` of our example. In this case, the delete operation at the operational source results in an update operation of the attribute `date_withdrawal` in SIRIUS.

Thus, in order to execute the refreshment process incrementally, SIRIUS must further perform the mapping between operational and global update operations. For a given source, an *operation mapping specification* is a pair

```
<local operation name [local attribute name {, local attribute name}],
 global operation name [global attribute name {, global attribute name}]>
```

where local and global operation names have one of the values `insert`, `update` or `delete`. In case of an update operation, affected warehouse attribute names also have to be defined (see also Figure 10 for an operation mapping example).

## 4 The SIRIUS Architecture

A data warehouse system (DWS) includes the data warehouse and all components responsible for building, refreshing, accessing and maintaining the DWH. In SIRIUS, we consider the *Data Warehouse Refresh Manager* (DWRM) as the central component of a DWS which has the knowledge about the tasks that must be accomplished during the DWH refresh-

ment process. Figure 8 illustrates the DWRM and components of a DWS related to the refreshment process.

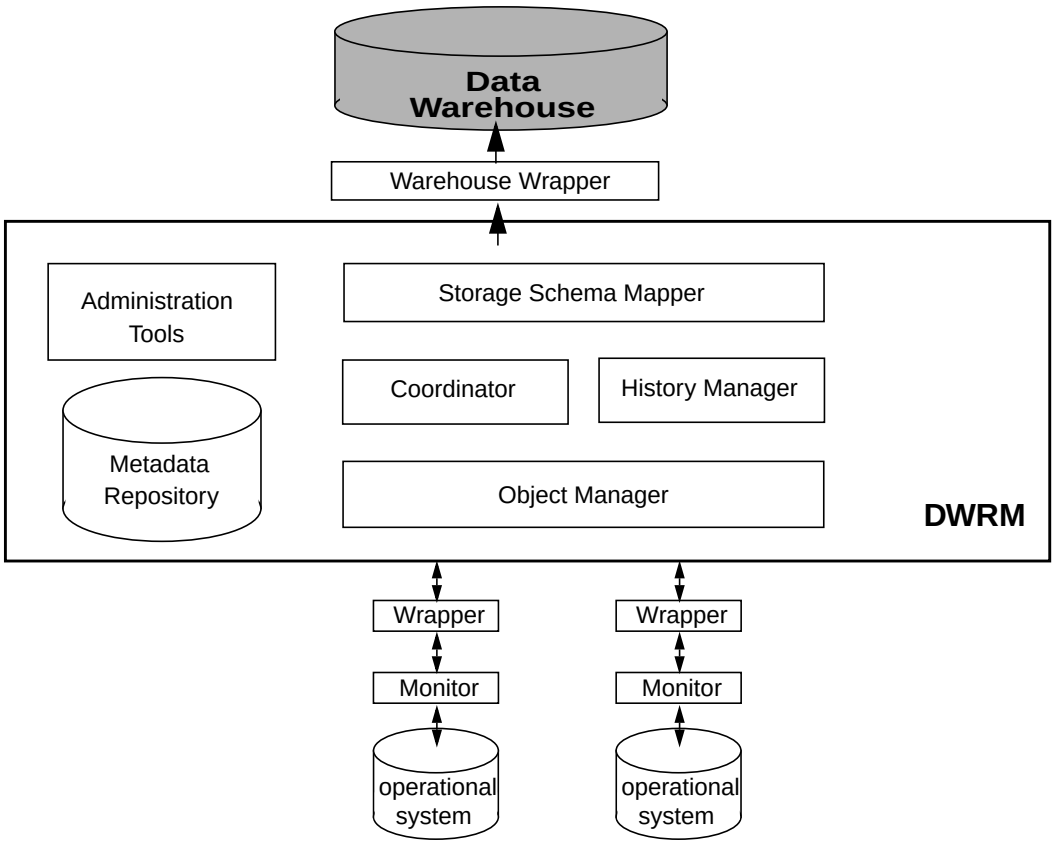


*Figure 8 Data Warehouse Refresh Manager as part of a DWS environment*

The *object manager* is responsible for populating the global schema during the execution of the refreshment process. Based on operation and structural mappings, operational updates are transformed into (transient) objects according to the global schema. Furthermore, tasks related to key management like the assignment of operational keys to warehouse keys are performed by the object manager (Section 4.2). The *storage schema mapper* performs the mapping of the global schema to storage schemas like the star or the snowflake schema (Section 4.3), whereas the *warehouse wrapper* loads the data warehouse by using the appropriate update operations of the respective warehouse DBMS. The *metadata repository* is used for the persistent storage and management of all metadata used in the refreshment process. It contains information like the description of operational systems and their contents, the particular refreshment steps required to process data from the sources into the DWH, and the documentation of executed transformation steps. The *coordinator* is responsible for initiating, controlling and monitoring the entire refreshment process. Finally, the *history manager* implements the various techniques for building histories supported by SIRIUS (Section 4.4).

The DWRM cooperates with operational sources through appropriate *monitors* and *wrappers*. Monitors detect relevant data modifications in each operational source using one of the techniques described below in Section 4.1 Wrappers translate modified data provided by the corresponding monitor into the common warehouse format, and send them to the DWRM.

In the following sections, the various SIRIUS components are presented in the order they cooperate during the execution of the refreshment process.

## 4.1 Monitors and Wrappers

One of the main goals of our approach is to provide mechanisms for refreshing a DWH incrementally, i.e., to propagate only relevant operational updates (which have occurred since the last DWH refreshment) into the warehouse. A prerequisite for refreshing a DWH incrementally is the detection and extraction of updates in operational systems. Depending on the kind of the operational system, various monitoring techniques can be applied for a concrete warehouse solution. In this section, we present and classify these techniques, and we demonstrate how they can be applied in a data warehousing environment and integrated in our approach.

Moreover, given a set of operational systems and a particular target storage schema, detected and extracted operational updates must be applied to the warehouse schema, i.e., updates must be appended to the previous warehouse state. Operation mappings (as described in Section 3.6) and a key concept based on OID's are used to assign update operations at operational sources to the corresponding warehouse data.

Finally, in contrast to an incremental refreshment approach, performing a complete reload of the warehouse implies that the individual tasks of the refreshment process are executed as soon as the beginning of the refreshment process is signalled. This results in a very long execution time and often in taking the warehouse off-line. In our incremental approach, various optimizations are possible because updates monitored between two refreshment points of time can be used to "prepare" some of the above-mentioned tasks (for example, transforming operational data into a common format) before the actual refreshment process starts.

### 4.1.1. Monitoring Updates at Operational Sources

As mentioned in Section 4.1, building complete histories and refreshing the warehouse incrementally presumes the detection of updates in the operational systems. For example, if warehouse users are interested in analysing how product price changes affect sales, the warehouse must provide all information about updates of product prices and sales. Since operational systems are not intended to store histories, there is a difference regarding how

updates are treated for this kind of data. For *event-oriented* data like product sales, each new value is explicitly stored. Delivering data for the warehouse means querying the source for all sales records. In contrast, updates on *state-oriented* data like product or customer information will normally overwrite previous values. Refreshing the warehouse correctly means that all relevant updates performed during two refreshment points of time in operational sources must be monitored and then - at refreshment time - propagated into the warehouse. In this section, we illustrate how monitors and wrappers cooperate for refreshing the DWH.

Each monitor is responsible for the detection of data (residing in the appropriate operational source) that has changed since the last DWH refreshment. Depending on the kind of operational systems, there are several techniques that can be used for this purpose:

- *Log-based* monitoring: Assuming that the source system is a database system maintaining and exposing a log, log entries with information about committed transactions that changed relevant operational data can be used to refresh the DWH incrementally. Once the beginning of the refreshment process is signalled, the monitor inspects the log file and extracts the relevant modifications that have occurred since the last execution of the refreshment process.
- *Trigger-based* monitoring: For sources that support active mechanisms [37] like triggers, appropriate events can be defined for the update operations of relevant data items. In this case, the action part of the trigger (or of an ECA-rule) writes relevant information (e.g., updated entities and attribute values, the kind of the update operation, and update time) into an auxiliary table. After refreshing the DWH, the contents of the auxiliary table can be removed.
- *Replication-based* monitoring: Using replication services of commercial DBMS is a further option to detect changes occurring in source systems. Tools like IBM's Data Propagator and Sybase's Replication Server provide mechanisms for propagating updates on base tables to outside the operational environment.
- *Application-assisted* extraction: Particularly for non-DBMS data management systems, changing existing applications or implementing new ones to notify about data changes is the only option to support the incremental DWH refreshment process. Creating snapshots of relevant data and comparing it (on a per-record basis) with the previous version that has been used for the DWH refreshment could be a solution for this problem. A further option is to change application programs to timestamp changed data. The monitor can periodically poll the source and select data with a timestamp greater than the one of the previous refreshment.

Figure 9 illustrates the usage of replication services provided by IBM's Data Propagator Relational for our example application. Data Propagator Relational provides replication services for the DB2 product family. Modification of source data are detected and propagated to consistent change data tables (CCD). Different kinds of CCD can be defined depending on user requirements. For our purpose, so-called complete noncondensed CCD tables provide all required information for maintaining complete histories of data changes including primary keys, old and new attribute values, and the operation code (for inserts, deletes and updates).

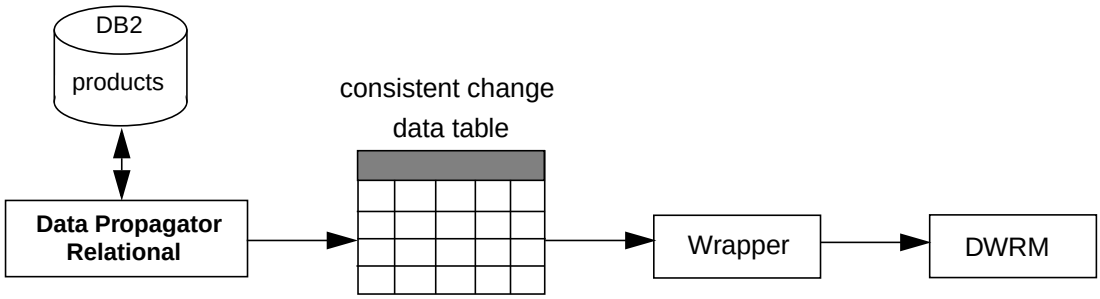


*Figure 9 Using replication services in SIRIUS*

### 4.1.2. Transforming Updates according to the Global Schema

The discussion of the monitoring techniques in Section 4.1.1. has shown that monitoring and extracting of updates in operational sources can be performed in various ways. For instance, updates detected by a monitor can be delivered in the form of tuples inserted into auxiliary tables or as simple dump files. In the next step of the refreshment process, wrappers are responsible for transforming modified data from various operational representations into a common structure that conforms to the defined global schema. Furthermore, update operations signalled by monitors must be mapped to the corresponding operations of the global (SIRIUS) layer. Results delivered by each wrapper provide all necessary information about modified data since the last refreshment.

### 4.1.3. OIF Representation

After transforming modified data using structural and operation mapping specifications, wrappers convert the results into a common format that can be further processed by the DWRM. For representing transformed objects in a uniform way, we use a variant of the OIF (Object Interchange Format) specification language for objects and their states [5]. OIF supports all object database concepts compliant to the ODMG Object Model like object identifiers, type bindings and attribute values. Each OIF object definition specifies the

type, attribute values, and relationships to other objects for the defined object. For example, the object definition

Johnson **Customer**{**last_name** “Johnson”, **first_name** “Ken”, **birth_date** “01/07/65”}

defines an instance of the class `customer` and initializes the attributes `last_name`, `first_name` and `birth_date` with the values “Johnson”, “Ken” and “01/07/65”. Since SIRIUS wrappers must provide information about modified objects, object definitions are extended by a prefix that indicates the *kind of the global update operation* (i.e., insert, update, delete). Furthermore, each object definition must also contain the corresponding local key which is then assigned to a SIRIUS object identifier. For updates of the local key, the previous value of the local key is also needed in order to transform the update correctly.

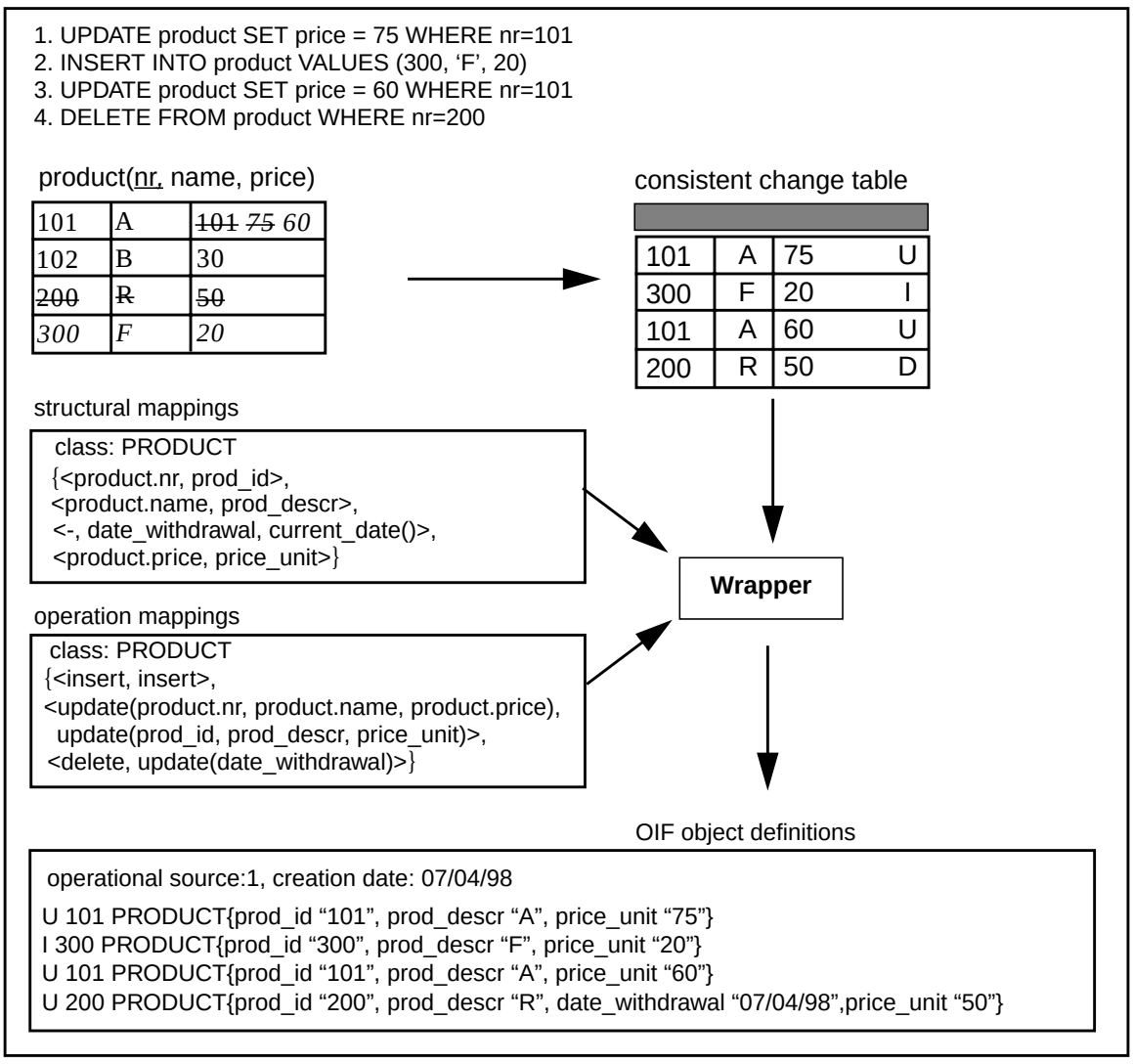


*Figure 10 An example for transforming operational updates into the global model*

Figure 10 demonstrates the shown transformation steps for a set of updates on table `product`. According to our example of Figure 9, updates are first propagated to the consis-

tent change table. Then, the wrapper generates for each tuple of this table an OIF object definition using the according structural and operation mappings.

### 4.1.4. Wrapper Operating Mode

Another important characteristic of the SIRIUS approach is the “preparation” of operational data before the next refreshment process execution is initiated, i.e., wrappers transform operational data into the warehouse format in an asynchronous mode (compared to the actual refreshment point of time). This results in a reduction of the time needed for refreshing the warehouse. The tasks of applying structural and operation mappings and converting modified data into OIF object definitions can be performed by each source wrapper in the time *between two warehouse refreshments* and independently of processing steps at other sources. As soon as the beginning of the refreshment process is signalled, only the integration of the results delivered by each wrapper must be further processed. This is a much more efficient operating mode compared to full reloads, where at the beginning of the refreshment process operational data must first be extracted and then transformed by the wrapper.

The wrapper operating mode is as follows. Wrappers access the data provided by monitors periodically or upon detection of a new update and translate the updates into OIF object definitions using the structural and operation mapping specifications as described in Section 3.6. Once the beginning of the refreshment process is signalled, the OIF object definitions are delivered to the Data Warehouse Refresh Manager. For the implementation of this “active” behavior in a simple but powerful way, we plan to use active mechanisms [37].

## 4.2 Object Manager

As in the ODMG model, each object in the SIRIUS layer has an *object identifier* which is unique and immutable during its entire lifetime [5]. Object identifiers are generated by the object manager and are an important concept for both, supporting the incremental warehouse refreshment and managing histories. Using immutable object identifiers in the SIRIUS layer enables to assign object types from operational systems to the persistent representation in the warehouse in a more natural and efficient way than using value-based identifiers (used by relational view-based warehouse approaches). Since value-based identifiers can be updated or deleted, propagating updates to the corresponding warehouse entities results in a much more complex task. In contrast, unique object identifiers allow the correct assignment of modified operational data to the corresponding warehouse data. This is a prerequisite for refreshing the warehouse incrementally.

Assuming that most operational systems support a different notion of object identity, additional information is needed in order to assign operational entities to SIRIUS objects.

For this purpose, an *object key* is assigned to each object identifier. Each object key consists of the local (operational) key provided by the operational system and a unique source key that indicates the operational system from which the modified operational data is extracted. [1] SIRIUS maintains a table that assigns OID's to the corresponding object keys. Since local primary key values can change, several object keys may be assigned to one OID (i.e., an OID can "point" to a set of object keys).

In contrast to the attribute values of the global schema, object keys and the corresponding OID's are stored persistently by the object manager in the SIRIUS level. During the execution of the refreshment process they are used as follows. After wrappers have transformed data that has been modified since the last refreshment into OIF object definition, the object manager proceeds with the creation of new SIRIUS objects and the assignment of values to basic global attributes. Depending on the information about the kind of the global update operation (provided with the OIF object definition), the object manager creates new OID's and object keys (in case of an insert) or uses the appropriate object key to match an existing OID (in case of updates or deletes of non-key attributes). Updates of primary key values result in creating a new object key version for the same OID.

## 4.3 Storage Schema Mapper

Data warehouse design methods consider the read-oriented character of warehouse data and enable efficient query processing over huge amounts of data. A special type of database schemas, called *star schema*, is often used to model the multiple dimensions of warehouse data. In this case, the database consists of a central fact relation and several dimension relations (Figure 11). The fact relation contains tuples that represent measures. Each tuple of the fact relation references multiple dimension relation tuples, each one representing a dimension of interest. Dimension relations of star schemas are not normalized in order to reduce the costs of joining the fact relation with dimension relations.

The mapping of a SIRIUS global schema to a star schema is a task performed by the storage schema mapper using a set of simple rules. Fact and dimension classes of the global schema are mapped directly to fact and dimension relations. Measures and dimension attributes build the attributes of the fact and dimension relations. For each relationship between fact and dimension classes, the storage schema mapper assigns the value of the dimension's primary key to the corresponding foreign key of the fact relation. Notice that

1. The latter may be used in various ways during warehouse refreshment, e.g., to compute derived attributes based on the origin of an attribute (e.g., the derived attribute `branch_ID` in the global schema of Figure 7), or to perform data cleaning.

mapping to star schemas and other multidimensional logical schemas is simple because SIRIUS enforces that the global schema is specified in a multidimensional way.

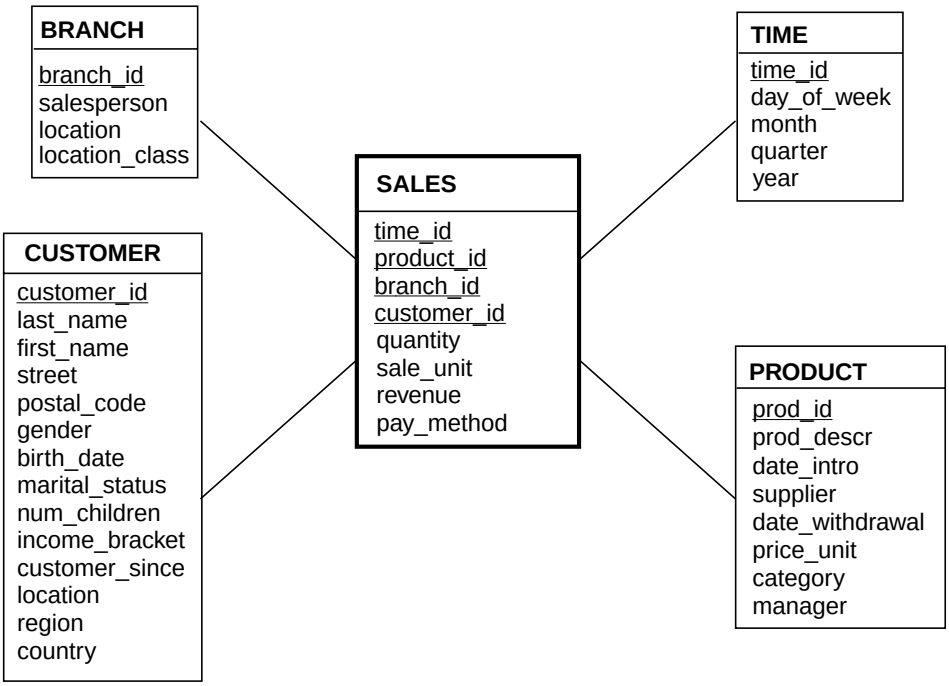


*Figure 11 Star schema of our example*

In contrast to star schemas, dimension relations of *snowflake schemas*, are in normalized form. Snowflake schemas increase the complexity and execution costs of queries. However, they reduce redundance and storage costs significantly, and are therefore often preferred (especially in very large data warehouses).

Normalization of dimensions typically results from defining new relations for hierarchy attributes. For example, based on the star schema of Figure 11, we define a new relation `region` for the hierarchy `customer->location->region->country`. Applying similar normalization steps results in the snowflake schema of Figure 12. Mapping a SIRIUS global schema to a snowflake schema is based on the specification of hierarchies (as discussed in Section 3). For each group of hierarchy attributes that build a new relation, the storage schema mapper generates the according tuples and foreign keys.

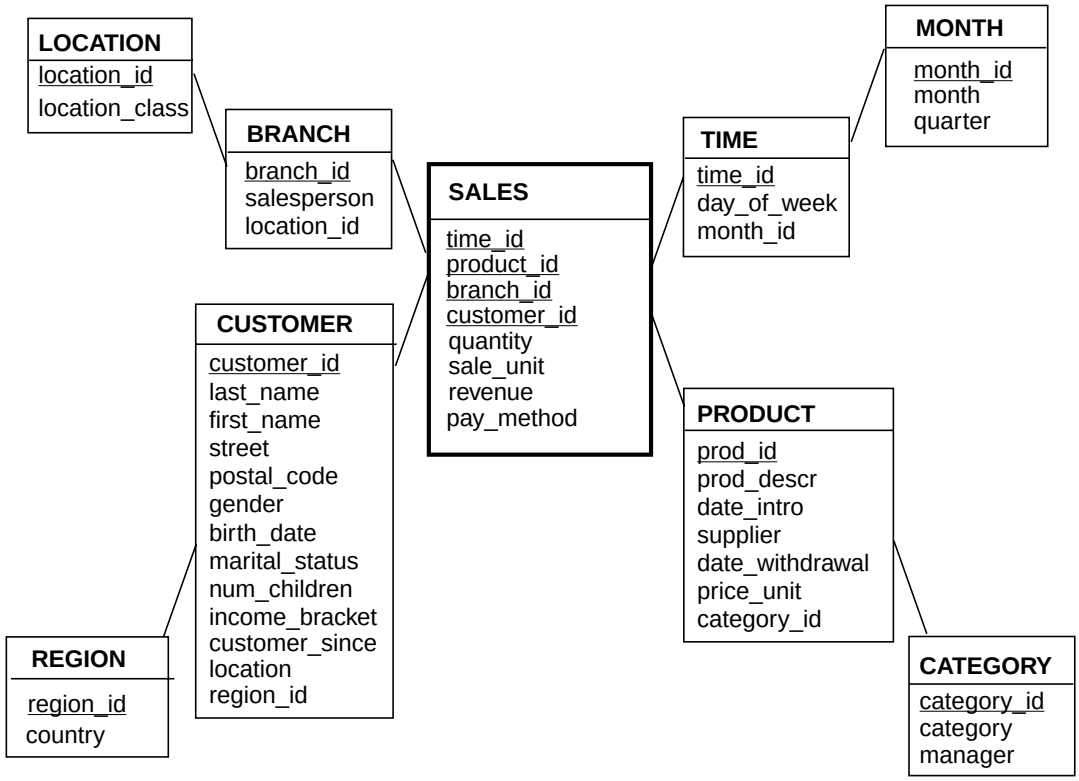


*Figure 12 Snowflake schema of our example*

## 4.4 History Manager

Maintaining histories of operational data in the DWH is one of the essential features of the data warehousing approach and one of the main reasons for building a DWH. History management improves decision support by offering the option of viewing changes of operational data over time and analysing the interdependencies among them (e.g., how did product price changes affect the company's sales in various regions). Particularly in the case where operational sources do not maintain histories of data, storing operational updates in a separate database (the DWH) is the only option to meet these kind of analysis requirements.

History management during the refreshment process is supported in SIRIUS by defining and processing history attributes. Notice that it makes only sense to define histories for state-oriented data, i.e., for attributes of dimension classes. For history attributes, SIRIUS supports various options for assigning attribute values depending on how temporal information is maintained in the DWH. Maintaining *complete* histories of warehouse data means that each update in the corresponding operational source is propagated and stored in the warehouse. For *partial* histories, all updates during two refreshment points are discarded, and only the current values - at refreshment time - are propagated into the ware-

house. At the same time, the values stored in the warehouse before refreshing it are being retained. Finally, in some cases *no* history is needed for parts of the warehouse, i.e., only the current values of the operational sources must be propagated into the warehouse to replace the old values.

Defining a history global attribute corresponds to the above mentioned case of complete histories. In this case, the attribute value consists of all modified data that have been assigned to the same SIRIUS object. For attributes that are not indicated as history attributes, only the last value assigned to a certain SIRIUS object is propagated into the warehouse. Distinguishing between partial and no histories is a task of the warehouse wrapper, i.e, according to the concrete storage schema, current values must be added to the previous values in the first case, and overwritten in the second.

# 5 Executing the Refreshment Process

The execution of the refreshment process is based on the specifications stored in the metadata repository. The coordinator is responsible for initiating and controlling the execution of the refreshment process. The beginning of the refreshment process can be signalled in various ways:

- depending on operational updates, a monitor signals that a predefined threshold has been reached and informs the coordinator,
- for periodical or user-initiated refreshments, the coordinator starts the refreshment process.

For the implementation of this “active” behavior in a simple but powerful way, we once again plan to use active mechanisms [37]. For example, the beginning of the refreshment process could be defined as a reaction on the occurrence of a primitive event (e.g., a time event or an operational update operation). Composite events can be used to start the warehouse refreshment depending on updates in several operational sources.

As described in Section 4.1.4., wrappers transform operational updates into OIF definitions between two refreshment points of time. Only in the case of limited monitoring capabilities, e.g., when periodical snapshots must be compared to extract the delta changes, the wrapper performs the OIF transformation after the beginning of the refreshment process. Then, according to operation and structural mappings, the object manager populates the global schema by creating new SIRIUS objects or assigning updated attribute values to existing ones. The order of instantiating the global schema with operational updates is important, i.e., *mappings for dimension attributes* are executed first. The reason is that after processing dimension updates, SIRIUS will propagate updates of fact attributes and check referential integrity. Thus, fact attribute updates that violate referential integrity will be

already discarded in the SIRIUS level (and not by the warehouse DBMS loader, eventually after executing several refreshment steps!). For this purpose, the primary keys of fact and dimension classes are persistently stored in SIRIUS. The goal of our approach is, after executing all refreshment steps, to "deliver" operational updates to the target warehouse and load the warehouse with a minimum of tasks (like indexing or partitioning) left to the warehouse DBMS. The resulting benefit is that the DWH needs not be taken off-line for a long time.

Structural mappings are performed according to the global schema by accessing the OIF definitions provided by wrappers (first for dimension and then for facts). In a next step, according to the specification of the particular refreshment steps the various vertical mappings and cleaning steps are executed. After all transformation steps have been completed, derived attributes can be processed. Finally, the storage schema mapper and the warehouse mapper perform the semantical and syntactical mapping from the global schema to the storage schema, respectively. The refreshment process ends with the loading of the target warehouse.

# 6 Summary

In this chapter, we give an overview of the main features provided by SIRIUS and the way our approach can be used for modeling and executing the data warehouse refreshment process. The definition of a concrete warehouse refreshment process is based on the meta model described in Section 3 and illustrated in Figure 6. The instance of this meta model for a concrete warehouse is stored in the metadata repository of SIRIUS. The various SIRIUS components (presented in Section 4) operate on top of the global schema and execute the refreshment process according to the specifications stored in the metadata repository. In summary, they support the following tasks:

- integration of various operational sources and refreshment of diverse target data warehouses by using structural and vertical mappings,
- default OID and object key management that ensures global object identity for warehouse data (generated OID's can be also used in the target DWH),
- incremental refreshment by defining operation mappings and using the SIRIUS object management facilities,
- predefined common derivations (e.g., date functions, aggregations, timestamps),
- cleaning by checking local keys and referential integrity for integrated data,
- mapping to typical DWH logical designs like star and snowflake schemas by defining the SIRIUS global schema and hierarchies,
- various kinds of history policies by using history attributes.

The basic functionality provided by SIRIUS can be extended to meet further application needs in various ways like

- definition of further structural and vertical mappings,
- implementation of advanced cleaning methods and integration in the SIRIUS repository, and
- definition of new or extension of existing derivations by implementing the appropriate methods (e.g., for new warehouse keys).

# 7 Conclusion and Status

In this paper, we presented the main features of the SIRIUS approach for modeling and executing the data warehouse refreshment process. In contrast to existing approaches which reduce the refreshment problem to the application of techniques for maintaining materialized views, SIRIUS provides mechanisms for modeling and executing several tasks of the refreshment process. Our approach allows to provide solutions for both, the static and dynamic aspects of the refreshment process in an integrated way. Integrating data from various heterogeneous operational sources, and refreshing a DWH independently of how warehouse data is persistently stored, is performed by defining various transformations and mappings on top of a global schema. Our approach considers the multidimensional character of warehouse data and allows the definition of derived data in various ways. Key management and maintenance of histories for different application requirements are supported. Various monitoring techniques for detecting relevant updates of operational data can be integrated in our approach. Furthermore, we showed how operational updates are transformed and propagated into the warehouse using structural, vertical and operation mapping specifications. Our object-oriented model and especially the notion of object identity allow the assignment of operational updates to the corresponding warehouse data in a powerful way.

We currently implement a data warehouse for the example of the mail-order business presented above. Operational sources are various database systems (e.g., Oracle and $O_2$) as well as flat files. Updates detected in these systems are loaded into different target warehouses (DB2 and Oracle). We have also implemented various monitors and wrappers according to the classification of Section 4.1.

The main focus of our future work is the extension of the history manager in order to support more complex techniques for advanced applications. Furthermore, we plan to extend the storage schema mapper by mappings for various multidimensional logical sche-

mas. Transaction support and concurrent execution of particular refreshment steps are further issues we plan to investigate in the future.

## References


1 D. Agrawal, A. El Abbadi, A. Singh, T. Yurek. Efficient View Maintenance at Data Warehouses. *Proc. of ACM SIGMOD Intl. Conf. of Management of Data*, Tucson, Arizona, May 1997.

2 E. Bertino. Integration of Heterogeneous Data Repositories by Using Object-Oriented Views. *Proc. of the 1st Intl. Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 1991.

3 S. S. Bhowmick, S. K. Madria, W. K. Ng, E.-P. Lim. Pi-Web Join in a Web Warehouse. *Proc. of the 6th Intl. Conf. on Database Systems for Advanced Applications* (DASFAA), Hsinchu, Taiwan, April 1999.

4 M. Bouzeghoub, F. Fabret. Modeling Data Warehouse Refreshment Process as a Workflow Application. *Proc. of the Intl. Workshop on Design and Management of Data Warehouses* (DMDW'99), Heidelberg, Germany, June 1999.

5 R. G.G. Cattell, D. Barry (ed). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, San Francisco, California, 1997.

6 S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record, 26:1*, March 1997.

7 C. Y. Chan, Y. E. Ioannidis. Bitmap Index Design and Evaluation. *Proc. of the ACM SIGMOD Intl. Conf. of Management of Data*, Seattle, Washington, June 1998.

8 C. Y. Chan, Y. E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, Philadephia, Pennsylvania, June 1999.

9 T. Critchlow. The DataFoundry Project: Managing Change in a Genome Warehouse. Presentation in: *IEEE EMBS Conf. on Information Technology in Biomedicin*e (ITAB'98), Washington DC, May 1998.

10 FirstLogic. *http://www.firstlogic.com*.

11 S. Gatziu, A. Vavouras, K.R. Dittrich. SIRIUS: An Approach for Data Warehouse Refreshment. Technical Report 98.07, Insitut für Informatik, Universität Zürich, July 1998.

12 M. Golfarelli, S. Rizzi. A Methodological Framework for Data Warehouse Design. *ACM First Intl. Workshop on Data Warehousing and OLAP* (DOLAP '98), Washington, D.C., USA, November 1998.

13 A. Gupta, I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin, 18(2)*, June 1995.

14 J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge. The Stanford Data Warehousing Project. In [35]

15 R. Hull, G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Apporaches. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, Montreal, Quebec, Canada, June 1996.

16 N. Huyn. Multiple-View Self-Maintenance in Data Warehousing Environments. *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, Athens, Greece, 1997.

17 W.H. Immon. *Building the Data Warehouse.* John Wiley, 1996.

18 M. Jarke, Y. Vassiliou. Data Warehouse Quality: A Review of the DWQ Project. Invited paper, *Proc. 2nd Conf. on Information Quality*, Massachusetts Institute of Technology, Cambridge, May, 1997.

19 M. Kaul, K. Dorsten, E.J. Neuhold. ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views. *Proc. of the 6th Intl. Conf. on Data Engineering*, Los Angeles, California, February 1990.

20 W. Labio, R. Yerneni , H. Garcia-Molina.Shrinking the Warehouse Update Window. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, Philadephia, Pennsylvania, June 1999.

21 C. Lee, Z.A. Chang. Utilizing Page-Level Join Index for Optimization in Parallel Join Execution. *IEEE Transactions on Knowledge and Data Engineering, 7(6)*, December 1995.

22 J. Mylopoulos, A. Gal, K. Kontogiannis, M. Stanley. A Generic Integration Architecture for Cooperative Information Systems. *Proc. of the 1st IFCIS Intl. Conf. on Cooperative Information Systems*, Brussels, Belgium, June 1996.

23 O'Neil, G. Graefe. Multi-Table Joins through Bitmapped Join Indices. *SIGMOD Record, 24(3)*, September 1995.

24 Platinum Software Corporation. *http://www.platinum.com*.

25 D. Quass, A. Gupta, I.S. Mumick, J. Widom. Making Views Self-Maintainable for Data Warehousing. *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, (PDIS '96), December 1996.

26 M.T. Roth, P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, Athens, Greece, 1997.

27 N. Roussopoulos. Materialized Views and Data Warehouses. *SIGMOD Record, 27(1), p21-26*, March 1998.

28 C. Sapia, M. Blaschka, G. Höfling, B. Dinter . Extending the E/R Model for the Multidimensional Paradigm. *Proc. Intl. Workshop on Data Warehouse and Data Mining* (DWDM '98), Singapore, November 1998.

29 D. Srivastava, S. Dar, H.V. Jagadish, A.Y. Levy. Answering Queries with Aggregation Using Views. *Proc. of the 22th Intl. Conf. on Very Large Data Bases*, Bombay, India, 1996.

30 M. Staudt, M. Jarke. Incremental Maintenance of Externally Materialized Views. *Proc. of the 22th Intl. Conf. on Very Large Data Bases*, Bombay, India, September 1996.

31 D. Theodoratos, S. Ligoudistianos , T. Sellis. Designing the Global Data Warehouse with SPJ Views. *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'99), Heidelberg, Germany, June 1999.

32 Trillium Software. *http://www.trilliumsoft.com*.

33 N. Tryfona, F. Busnorg, J.B. Christiansen. StarER: A Conceptual Model for Data Warehouse Design. *ACM Second Intl. Workshop on Data Warehousing and OLAP* (DOLAP '99), Kansas City, Missouri, USA, November 1999.

34 A. Vavouras, S. Gatziu, K.R. Dittrich. The SIRIUS Approach for Refreshing Data Warehouses Incrementally. *Proc. GI-Conf. Datenbanksysteme in Büro, Technik und Wissenschaft* (BTW), Freiburg, Germany, March 1999

35 J. Widom (ed.). *Special Issue on Materialized Views and Data Warehousing, IEEE Data Engineering Bulletin*, 18:2, June 1995.

36 J. Widom. Research Problems in Data Warehousing. *Proc. of the 4th Intl. Conf. on Information and Knowledge*, Baltimore, 1995.

37 J. Widom, S. Ceri (ed). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan-Kaufmann, 1996.

38 J. Wiener, H. Gupta, W. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom. A System Prototype for Warehouse View Maintenace. *Proc. of the ACM Workshop on Materialized Views: Techniques and Applications*, Montreal, June 1996.

39 M.-C. Wu. Query Optimization for Selections using Bitmaps. *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, Philadephia, Pennsylvania, June 1999.

40 M-C. Wu, A.P. Buchmann. Research Issues in Data Warehousing. *Datenbanksysteme in Büro, Technik und Wissenschaft: GI-Fachtagung*, Springer-Verlag, Ulm, 1997.

41 M-C. Wu, A.P. Buchmann . Encoded Bitmap Indexing for Data Warehouses. *Proc. of the 14th Intl. Conf. on Data Engineering*, Orlando, Florida, Februar 1998.

42 J. Yang, K. Karlapalem, Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, Athens, Greece, August 1997.

43 J. Yang and J. Widom. Maintaining Temporal Views Over Non-Historical Information Sources For Data Warehousing. *Proc. of the 14th Intl. Conf. on Data Engineering*, Orlando, Florida, Februar 1998.

44 X. Zhang , E.A. Rundensteiner. Data Warehouse Maintenance Under Cuncurrent Schema and Data Updates. *Proc. of the 15th Intl. Conf. on Data Engineering*, Sydney, Austrialia, March 1999.

45 Y. Zhuge, H. Garcia-Molina, J, Hammer, J. Widom. View Maintenance in a Warehousing Environment. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, San Jose, California, May 1995.

46 Y. Zhuge, H. Garcia-Molina, J.L. Wiener The Strobe Algorithms for Multi-Source Warehouse Consistency. *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, (PDIS '96), December 1996.

47 Y. Zhuge, J.L. Wiener, H. Garcia-Molina. *Multiple View Consistency for Data Warehousing.* Proc. of the 13th Intl. Conf. on Data Engineering, Birmingham U.K., April 1997.

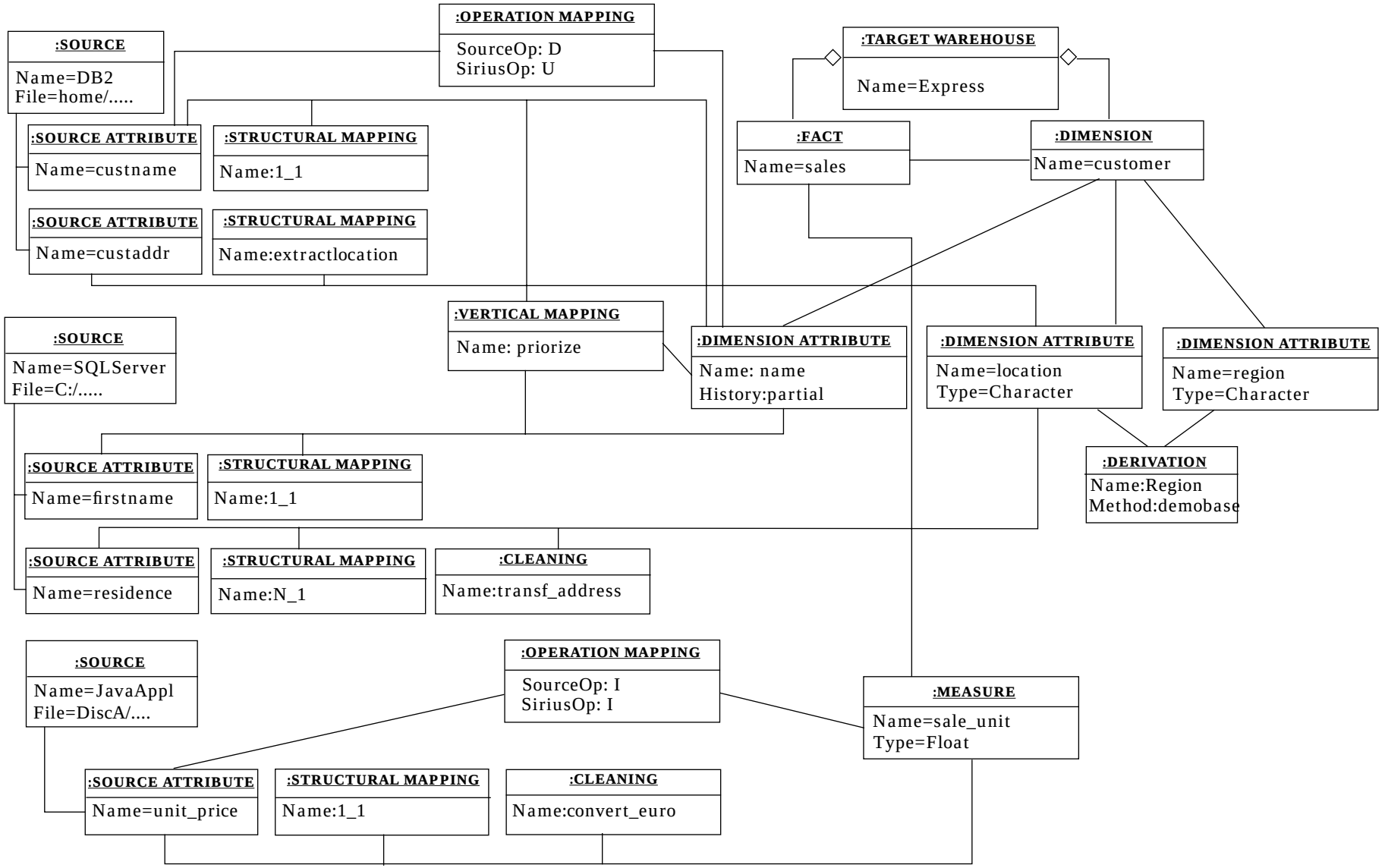


*Figure 13 Example refreshment process specification*