Incremental Refreshment of Data Warehouses : The SIRIUS Approach

Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich

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

Abstract

Data warehouse refreshment incorporates issues in how to capture and propagate updates occurring in operational systems into the warehouse in order to keep the data warehouse up-to-date. In this paper, we propose an approach for the incremental refreshment of data warehouses which considers the special nature of warehouse data, e.g., its temporal and heterogeneous dimension. Our approach can be used for the integration of data from a wide variety of heterogeneous operational sources and independently of a particular database system used for storing the warehouse data.

1 Introduction

The topic of *data warehousing* [4, 7, 10, 11, 23, 24, 27] 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. 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.

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 and integrated, and how the DWH data will be accessed during analysis. After the initial load (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 such that warehouse data reflects 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.

In the context of our project SIRIUS (**S**upporting **I**ncremental **R**efreshment of **I**nformation Wareho**u**se**s**) we investigate the process of refreshing a DWH. Our approach provides concepts for refreshing data warehouses independently of how warehouse data are persistently stored, i.e., we make no assumptions about the database system used for storing the warehouse data [22]. The SIRIUS approach can be used in data warehouse environments consisting of a wide variety of heterogeneous operational sources (various database systems, flat files, etc.). Furthermore, it provides mechanisms regarding the special nature of the warehouse data like derived and historical data.

One of the main goals of our approach is to provide mechanisms for refreshing a DWH incrementally, i.e., to propagate only relevant updates in operational systems (which have occurred since the last DWH refreshment) into the warehouse. There are several reasons for refreshing a DWH incrementally in contrast to full reloads. First, today's 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 the DWH must be updated 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 lot 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, detecting all updates of operational data in order to refresh the warehouse incrementally at the same time enables to maintain histories in the warehouse correctly. In contrast, discarding updates between two refreshment points of time and reloading warehouse data leads to the loss of histories.

The topic of data warehouse refreshment so far has been investigated in the research community mainly in relation with techniques for maintaining materialized views [1, 6, 8, 9, 15, 17, 18, 19, 20, 26, 28, 29, 30, 31]. 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). To the best of our knowledge, SIRIUS is the first approach which deals with the refreshment problem without restricting the DWH to a set of relational views over operational data. This allows us to widely explore the refreshment problem considering also issues like building histories or performing cleaning of the operational data.

The paper is organized as follows. Section 2 discusses issues and problems relevant to the incremental refreshment problem. Section 3 presents a running example of a data warehouse application for a mail-order business. In Section 4, we give an overview of the SIRIUS architecture. Section 5 presents the basic constructs of the SIRIUS data model, and Section 6 discusses various monitoring techniques for detecting operational updates. Section 7 and 8 describe how operational updates are transformed and further processed before loading the DWH. The paper concludes with Section 9 which presents our current and future work.

2 Incremental Refreshment Issues

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 transforming operational updates into a common format and reconciling structural and semantic differences. In particular, before loading relevant updates of operational data into the warehouse, updated data must be integrated and “homogenized” according to a uniform, global data model. Besides aspects related to heterogeneity of integrated sources, refreshing a DWH incrementally implies the following issues:

- A prerequisite for refreshing a DWH incrementally is the detection and extraction of updates in operational systems. Depending on the kind of the integrated operational systems, various monitoring techniques can be applied for a concrete warehouse solution. In this paper, we present and classify these techniques, and we demonstrate how they can be applied in a data warehousing environment and integrated in our approach.
- Given a set of operational systems and a particular target warehouse schema, detected and extracted operational updates in a next step must be applied to the warehouse schema, i.e., updates must be appended to the previous warehouse state. As a result, refreshing the DWH necessitates concepts for assigning update operations at operational sources to the corresponding warehouse data.
- Performing a full reload of the warehouse implies that the individual tasks (like extraction, transformation, cleaning, merging, sorting of operational data) 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 usually 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.
- Finally, special mechanisms are needed in order to consider particular properties of warehouse data, e.g., maintenance of histories or building aggregations.

3 A Running Example

In our running example, a mail-order business aims at building a data warehouse to support decision making for different user groups and departments (illustrated in Figure 1).

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.

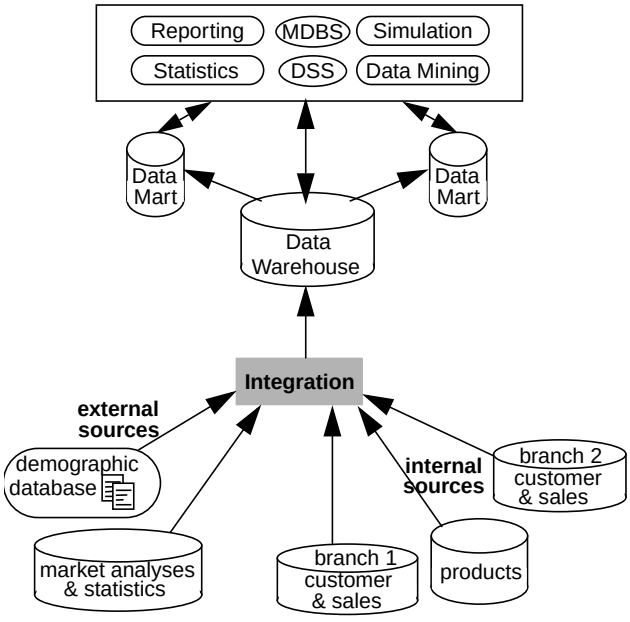

Figure 1 The Data Warehouse System for the Mail-Order business

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 refreshment process. Figure 2 illustrates the DWRM and components of a DWS related to the refreshment process.

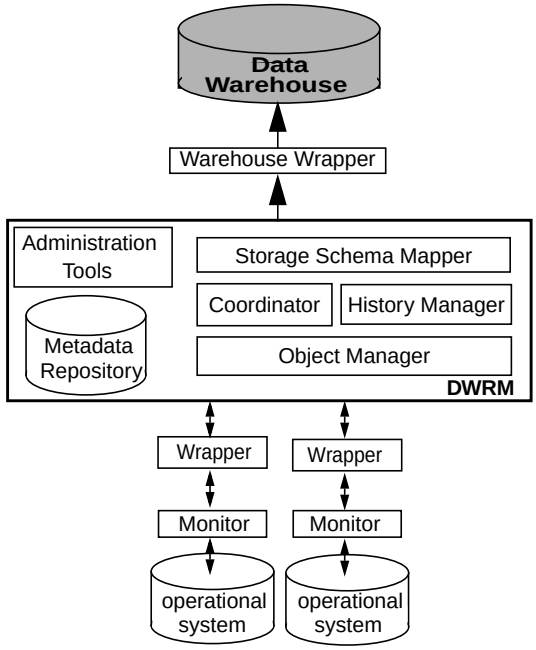

Figure 2 Data Warehouse Refresh Manager as part of a DWS environment

The *object manager* is responsible for populating the SIRIUS global schema (Section 5) during the execution of the refreshment process. Based on operation and structural mappings (Section 7), 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. The *storage schema mapper* performs the mapping of the global schema to storage schemas like the star or the snowflake schema, 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

(for a detailed description of the SIRIUS components see [22].

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 6. Wrappers translate modified data provided by the corresponding monitor into the common warehouse format (Section 7), and send them to the DWRM.

5 The Global Data Model

As mentioned in Section 2, capturing the heterogeneity of operational sources necessitates concepts for the definition of a uniform "global" view on integrated operational data. In our approach, the structure of the integrated operational data is described by a *global schema* using the SIRIUS global data model. The global schema is used for the uniform representation of the operational updates at an intermediate layer between operational sources and the target data warehouse. It is the basis for executing further steps of the refreshment process, e.g., integrating and cleaning data, building histories, processing derived data, and loading the DWH. On the other side, we assume that a *storage schema* is used for defining the structure of the DWH data as it is stored in the DBMS used for the persistence of the DWH and as it is visible to the DWH users. 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.

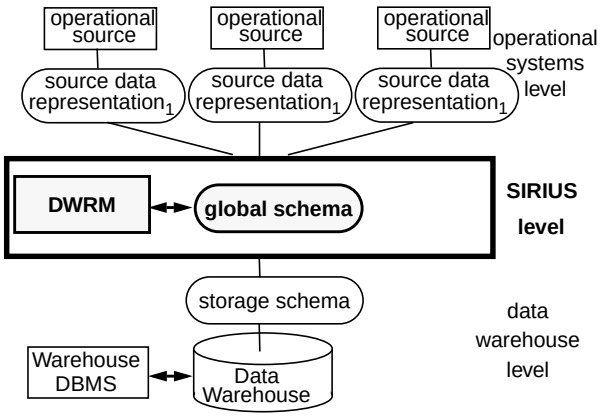

Figure 3 Schema architecture in SIRIUS

Figure 3 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 DWRM components which are responsible for the execution of the refreshment process operate on top of the SIRIUS global schema.

The SIRIUS *global data model* is based on the object-oriented data model of the Object Database Management Group (ODMG) standard [3]. Similar to other projects focusing on data integration in heterogeneous environments [13, 16, 2, 12], we exploit the rich semantic expressiveness of object-oriented data models for representing the structure of integrated operational data in the SIRIUS layer. 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. During the configuration phase, object types (at the SIRIUS level) related to existing object types of the operational sources are defined. During the execution of the refreshment process, built-in operations are used from the DWRM components for creating objects, retrieving and updating the attributes of an object.

In the ODMG object model, attributes may have simple or complex values (i.e., sets or references to other objects). In our approach, we additionally distinguish between *basic* and *derived* attributes [22]. Values of basic attributes are assigned directly from the values of the corresponding attributes in the 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 source systems. Derived attributes can be defined based on values of basic attributes either by performing arithmetical operations or using rules (implemented using methods).

A further feature of our global data model is the notion of so-called *history* attributes. 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 8. A part of the global schema of our running

example is shown in Figure 4.

```
interface SALES {
  attribute integer quantity;
  attribute date sales_date;
  attribute integer saleprice_unit;
  derived attribute integer customer_age;
  derived attribute integer branch_ID;
 }
interface PRODUCT {
  attribute integer prod_nr;
  attribute string prod_descr;
  attribute string prod_category;
  attribute string date_intro;
  attribute string date_withdrawal;
  attribute string margin_unit;
  history attribute string price_unit;
 }

interface CUSTOMER {
  attribute string last_name;
  attribute string first_name;
  attribute date birth_date;
  attribute string marital_status;
  history attribute string address;
  history attribute string postal_code;
  history attribute string location;
  derived attribute string loc_class;
  derived attribute date customer_since;
  derived attribute date branch_ID;
 } ;
```

Figure 4 Global schema example

As in the ODMG model, each SIRIUS object has an *object identifier* (OID) which is unique and immutable during its entire lifetime [2]. 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 SIRIUS 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 appropriate wrapper and a unique source key that indicates the operation system from which the modified operational data is extracted. The latter may be used in various ways during the refreshment steps, e.g., to compute derived attributes based on the origin of an attribute (e.g., the derived attribute `branch_ID`) or to perform data cleaning. 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.

6 Monitoring Updates at Operational Sources

As mentioned in Section 2, 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 information 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 integrated 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 [25] 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 5 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).

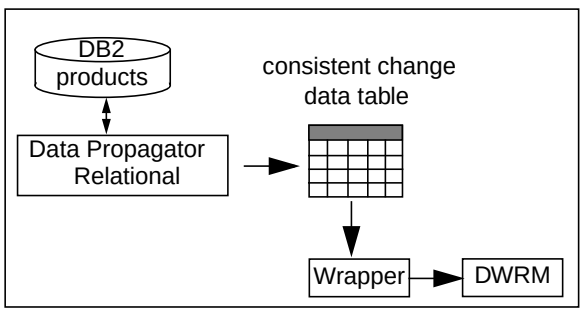

Figure 5 Using replication services in SIRIUS

7 Transforming Updates according to the Global Schema

The discussion of the monitoring techniques in Section 6 has shown that monitoring and extraction of updates at operational sources can be performed in various ways. Updates detected by a monitor can be delivered, for example, 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 in the SIRIUS level. Results delivered by each wrapper provide all necessary information about modified data since the last refreshment.

7.1 Structural and Operation Mappings

For each operational source, the appropriate wrapper provides two kinds of mapping functionality:

- for each basic attribute of the global schema, the wrapper performs a *structural mapping*, i.e., attributes of the integrated sources are mapped to global attributes, and
- for each kind of update operation signalled by a monitor, the wrapper performs an *operation mapping*, i.e., local update operations are assigned to the corresponding global update operations.

Structural and operation mappings are performed using specifications which are defined during the DWH configuration phase. 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. A structural mapping can be defined for one attribute or a group of attributes extracted from (exactly) one operational source. The syntax for *structural mapping specifications* 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 a mapping method that performs a simple arithmetic operation. Figure 6 shows an example of structural mappings for the class PRODUCT of our example global schema.

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 4, 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 6 for an operation mapping example).

7.2 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 persistent objects and their states [3]. 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.

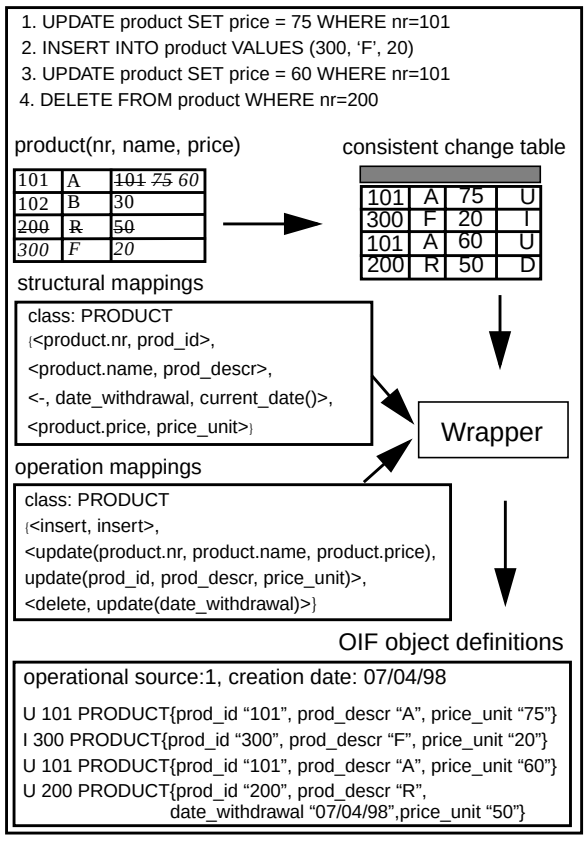

Figure 6 An example for transforming operational updates

Figure 6 demonstrates the transformation steps for a set of updates on table product. According to our example of Figure 5, updates are first propagated to the consistent change table. Then, the wrapper generates for each tuple of this table

an OIF object definition using the according structural and operation mappings.

7.3 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 various auxiliary monitoring structures periodically or upon detection of a new update and translate the entries into OIF object definitions using the structural and operation mapping specifications as described in Section 7.1. 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 [25].

8 Representing Operational Updates as SIRIUS Objects

Data that changed since the last refreshment are collected from the various wrappers by the DWRM (in form of OIF object definitions). The object manager proceeds with the creation of new SIRIUS objects and the assignment values to basic global attributes. First, 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 (in case of an insert) or uses the appropriate local key to match an existing OID (in case of updates or deletes). Recall from Section 5 that unique object identifiers allow the correct assignment of modified operational data to the corresponding warehouse data.

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 warehouse. 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.

In a next step, since data extracted from operational systems contains errors, it must 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. Note that, since only updates of operational data are propagated into the warehouse, cleaning must be performed on modified data. 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 [14], Centric [5], and the Trillium Software System (R) [21].

In the last step of the warehouse refresh process, cleaned data must be complemented with further information and then loaded into the DWH. In most cases, additional preprocessing is required before loading data into the DWH, including calculation of derived attributes, assignment of time-related information (usually by timestamping data with the update date) and providing a common level of detail for data from different sources. Finally, the storage schema mapper and the warehouse

mapper perform the semantical and syntactical mapping from the global schema to the storage schema, respectively [22]. The refreshment process ends with the loading of the target warehouse.

9 Conclusion and Status

In this paper, we presented the main features of the SIRIUS approach for refreshing a DWH incrementally. Our object-oriented approach allows the integration of data from various heterogeneous operational sources and provides mechanisms for refreshing a DWH independently of how warehouse data are persistently stored. We described how various monitoring techniques for detecting relevant updates of operational data can be integrated in our approach. Furthermore, we showed how operational updates are transformed into a uniform global model using structural and operation mapping specifications. Our object-oriented model and specially the notion of object identity allow the assignment of operational updates to the corresponding warehouse data in a powerful way. In contrast to existing approaches which reduce the incremental refresh problem to techniques for maintaining materialized views, SIRIUS provides mechanisms regarding the special properties of warehouse data like maintenance of histories or building aggregations.

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 6.

The main focus of our future work is the investigation of the impact of various decision support applications for the global model design and the specification of the warehouse wrapper. The development of techniques for generating warehouse wrappers based on declarative specifications will reduce the effort for wrapper implementation. Furthermore, we plan to extend the storage schema mapper by mappings for various multidimensional logical schemas.

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

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

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

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

7 J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge. The Stanford Data Warehousing Project. In [23].

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

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

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

11 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.

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

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

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

15 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.

16 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.

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

18 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.

19 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.

20 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.

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

22 A. Vavouras, S. Gatziu, K.R. Dittrich. Modeling and Executing the Data Warehouse Refreshment Process. Technical Report, Department of Information Technology, November 1999.

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

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

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

26 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.

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

28 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.

29 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.

30 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.

31 Y. Zhuge, H. Garcia-Molina, and 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.