published in: Proceedings of 3rd IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems - DAIS`01, Vol. 198, ISBN 0-7923-7481-9, pp 311-316

Designing and Implementing an Object–Relational Data Warehousing System

Bodgan Czejdo[1], Johann Eder[2], Tadeusz Morzy[3], Robert Wrembel[3]

[1] Department of Mathematics and Computer Science, Loyola University
St. Charles Ave., New Orleans, LA 70118
czejdo@loyno.edu

[2] Institut für Informatik–Systeme, Universität Klagenfurt
Universitätsstr. 65, A-9020 Klagenfurt, Austria
eder@ifi.uni-klu.ac.at

[3] Institute of Computing Science, Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland
morzy@put.poznan.pl, Robert.Wrembel@cs.put.poznan.pl

Abstract

In this paper we present some of the results achieved while realizing an international research project aiming at the design and development of an Object–Relational Data Warehousing System (*ORDAWA*). Important goals of the project are to develop techniques for the integration and consolidation of different external data sources in an object–relational data warehouse, the construction and maintenance of materialized relational as well as object–oriented views, index structures, query transformations and optimizations, and techniques of data mining. The achievements discussed in this paper concern the application of materialized object–oriented views in the process of building an object–relational data warehouse.

Keywords: object–relational data warehouse, operational data store, object–oriented view, view schema, view materialization, maintenance of a materialized view.

1. Introduction

Organizational decision support systems require a comprehensive view of all aspects of an enterprise. Information collected in an enterprise are often of different data format and complexity (e.g. relational, object–relational, and object–oriented databases, on-line multimedia data stores, Web pages, spreadsheets, flat files etc.). Therefore, one of the important issues is to provide an integrated access to all these heterogeneous data sources.

There are two basic approaches to data integration: the *mediated* approach and the *data warehousing* approach [1]. In the first one, a global query is decomposed and transformed into queries for each of the data sources. The results of each of the queries are then translated, filtered, and merged to form a global result.

Whereas in the *data warehousing* approach, data of interests coming from heterogeneous sources are extracted, filtered, merged, and stored in an integrating database, called *data warehouse*. The data are also enriched by historical and summary information. Then, queries are issued for this data warehouse.

The advantages of the data warehousing approach to data integration are as follows: (1) queries operate on a local centralized data repository, that reduces access time to data, (2) queries need not be decomposed into different formats and their results need not be integrated, (3) a data warehouse is independent of other data sources, that may be temporary unavailable. However, a data warehouse has to be kept up to date with respect to source data, by periodically refreshing it. A data warehouse is often implemented as the set of materialised views.

This paper describes some of the results achieved while realizing an international research project aiming at the design and development of an Object–Relational Data Warehousing System – *ORDAWA* [2]. The project is conducted in co–operation of the Institute for Informatics–Systems at Klagenfurt University, the Institute of Computing Science at Poznań University of Technology, and the Department of Mathematics and Computer Science at Loyola University. Important goals of the project are to develop techniques for the

integration and consolidation of different external data sources in an object/relational data warehouse, the construction and maintenance of materialized relational as well as object–oriented views, index structures, query transformations and optimizations, and techniques of data mining. The achievements discussed in this paper concern the application of materialized object–oriented views in the process of building an object–relational data warehouse.

2. Object–Relational Data Warehouse

The more and more frequent need to create, store, and manage data of a complex structure and behaviour leads to the make use of object–relational (e.g. *Oracle8*, *Extended Parallel Server*, *IBM DB2 UDB*) or object–oriented databases (e.g. *Objectivity/DB*, *ObjectStore*, *Versant*). These complex data, similarly as relational, will be the subject of the integration and analysis in a central repository – a data warehouse. To this end, the data model of such a data warehouse should support complex types in order to reflect the complexity of source information. Moreover, object data use the methods and these methods should also be available in the integrated system, in order to retain the functionality and information as well as to be able to process the information. Therefore, many research papers suggest to use an object–relational or object–oriented data model (cf. [3, 4]) as a common model for integrating heterogeneous data sources.

Our object–relational data warehousing architecture is presented in Figure 1. An intermediate layer, called an *operational data store* (ODS) is built between data sources and a data warehouse [5, 6]. An operational data store is a subject–oriented set of data coming from one or more data sources. While loading an ODS, data are extracted from data sources, transformed to a common data model of an ODS, and preintegrated. Then, the content of various operational data stores is again integrated and loaded into a central data warehouse.

The main differences between a data warehouse and an operational data store are as follows. Firstly, the content of an ODS changes more frequently than the content of a data warehouse. Secondly, data in an ODS are near current with respect to the corresponding data in data sources. Next, data in an ODS can be updated but the updates do not propagate down to data sources. And finally, the aggregation level of data in an ODS is very low.

The purpose to build an ODS is to separate long lasting, time consuming processing, e.g. On–Line Analytical Processing queries, from data sources. Since OLAP queries operate on data in an ODS, they do not interfere with the processing in data sources. Furthermore, an ODS may be designed and tuned especially for a particular pattern of processing, whereas the underlying data sources may be designed and tuned for other kind of processing, e.g. OLTP. For the reason of tuning one may need to change the structure of source data in an ODS, e.g. merge some source information, project some attributes, introduce data redundancies.

From the implementation point of view, an ODS is seen as the set of materialised object–oriented views.

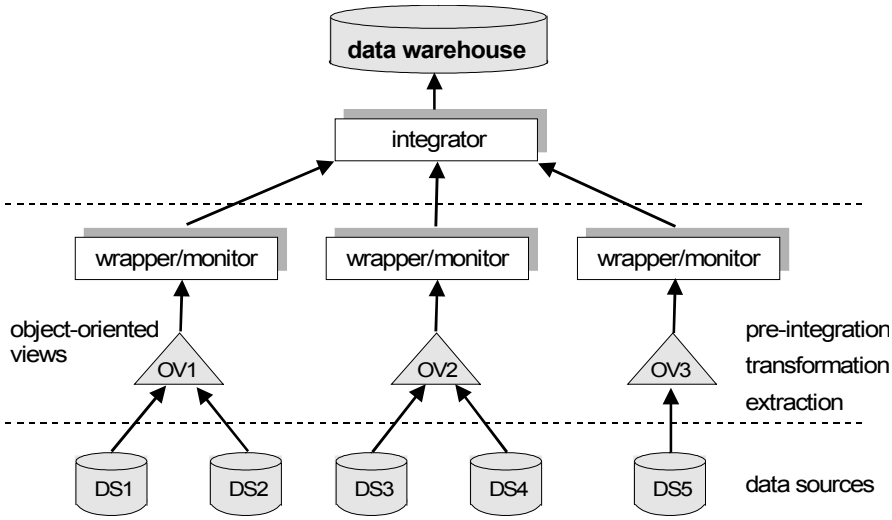

Figure 1. The application of object–oriented views in a data warehouse architecture

While materialising an object–oriented view one has to consider the materialisation of structure as well as the materialisation of methods. The materialisation of methods is important firstly because this technique allows to increase performance of methods whose computation take long time. Secondly, by materialising methods one is able to transform behaviour of objects to relational model. In this case materialised methods are just seen as attributes having persistent values. However, not all methods are good candidates for the materialisation. For example, method comparing or looking for similarities between two pieces of information can not be easily materialised. For these reasons, a data warehousing system has to provide means for the materialisation of methods as well as for invoking and computing methods when needed, i.e. should be able to maintain the behaviour that is not easy to materialise.

To sum up, by using object–oriented views for modelling and implementing operational data stores we achieved the following goals: (1) we are able to extend data warehousing process to be applicable also to object–oriented data sources; (2) due to the expressiveness of an object–oriented data model we are able to transform one data model to another and this transformation may be supported by methods defined in a view; moreover, the restructuring of source data in an operational data store can be easier.

3. The Concept of an Object–Oriented View

In our approach, called *View Schema Approach* (*VSA*), we draw on the model of an object–oriented database, presented in [7], which we extended by the following concepts supporting object–oriented views, their materialization and maintenance: (1) view class, (2) view object, (3) view schema, (4) mappings between database schema and views, and (5) mappings between objects in a database and those in views.

An *object–oriented view* is defined as a *view schema* of an arbitrary complex structure and behavior [8]. A view schema is composed of view classes. Each *view class* is derived from one or more *base classes*, i.e. classes in a database schema. A view class is derived by an OQL–like command. View classes in a view schema are connected by inheritance and aggregation relationships. Several view schemas can be created from the same base schema.

A given view schema can be explicitly materialised. The materialisation of a view schema results in the materialisation of its view classes. When materialising the instances of a given view class it is possible to materialised the structure of objects and results of some methods applicable to view objects.

After objects have been materialised in a view, the maintenance of consistency between base objects and those materialised in a view becomes an important issue. In our approach we have developed three following techniques for propagating modifications from source objects to their materialised counterparts in a view schema: (1) deferred on commit incremental refreshing, (2) deferred on demand incremental refreshing, and (3) deferred on demand complete refreshing. Only certain view classes are allowed to be refreshed incrementally.

In order to incrementally propagate the modifications from base to view objects we have developed additional data structures, called *Class Mapping Structure*, *Object Mapping Structure*, and *Log*. The first two structures represent mappings between base and view classes as well as between their instances, cf. [8]. Whereas *Log* stores records describing operations performed on base objects, cf. [9].

4. The *View Schema Approach Prototype*

The theoretical foundation concerning a materialised object–oriented view have been incorporated into a prototype software, called the *View Schema Approach Prototype* (*VSAP*).

The functionality that has been implemented and is supported by the *View Schema Approach Prototype* is the following: (1) the creation and management of several view schemas, (2) the creation and management of view classes in a given view schema, (3) the materialisation of view schemas and the maintenance of their consistency, (4) checking the consistency of a view schema with regard to structural and behavioural closure criteria, cf. [10].

The *View Schema Approach Prototype* is composed of two parts: the client part and the database part. The client part is composed of the four following software packages, namely: user interface, VSA manager, command analyser, and Pro*C interface. Whereas the database part is composed of one software package. Data and some procedures and functions are stored in the *Oracle8i* database.

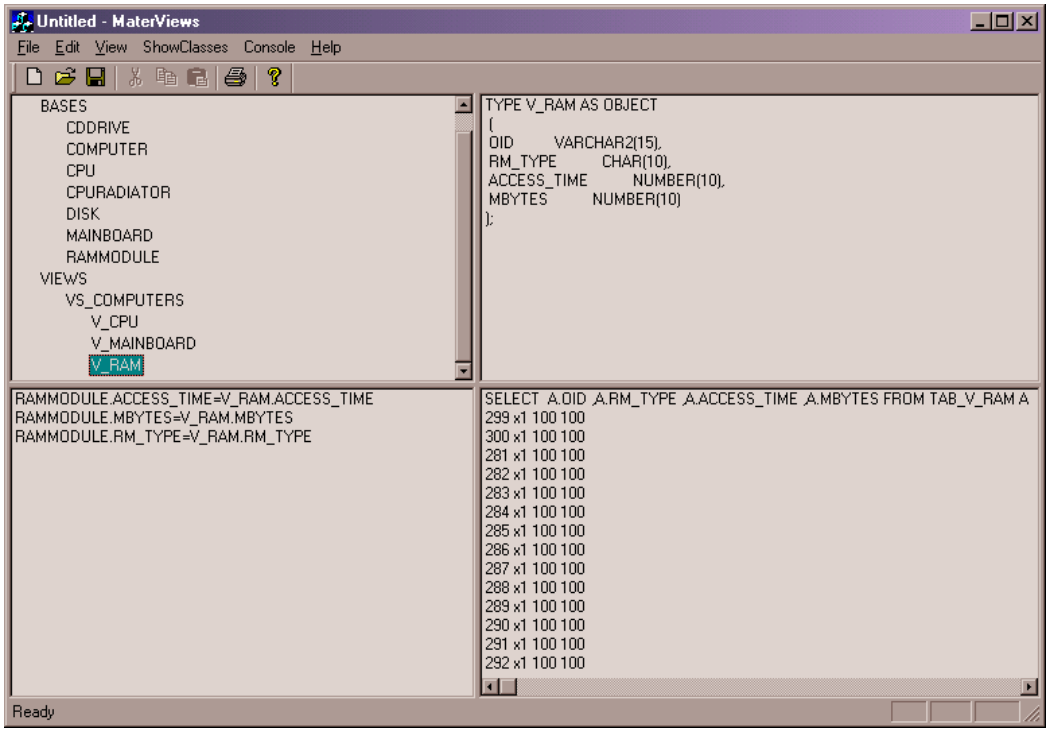

Figure 2. The user interface window of *VSAP*

The *user interface* package is implemented in C/C++. It contains the set of classes implementing the window interface used for managing view schemas. Users execute their commands and inspect the content of a view database in this interface. *VSA manager* is responsible for executing user commands and gathering their results as well as for refreshing view schemas when a transaction commits. In order to perform these tasks, the VSA manager uses command analyser, Pro*C interface, and database interface. C*ommand analyser* is responsible for analysing commands issued via *console*, checking their syntax, translating into the SQL language, and sending to a database for execution. *Pro*C interface* is used for the communication with a database, passing SQL commands to a database for execution, invoking procedures and functions stored in a database, and receiving data from a database.

The *database interface* consists of various procedures, functions and packages stored in a database. They are implemented in the *Oracle* PL/SQL language, often using *dynamic SQL*. Additionally, this software package contains the *VSA data dictionary* used for storing information about view schemas and their components.

The user interface window is shown in Figure 2. The main window is composed of four panels. The upper left hand panel, called *object navigator*, displays base classes (node *BASES*) and view schemas (node *VIEWS*). Both nodes can be expanded or collapsed. Each node under *VIEWS* represents a view schema, e.g. *VS_Computers*. Expanding a view schema node makes its view classes visible. The upper right hand panel displays the definition of a view class which is selected in object navigator. In an example window shown in Figure 2 the definition of the *V_RAM* view class is displayed in this panel. The lower left hand panel shows the derivation information about a view class and its base classes. The lower right hand panel exists in the first version of *VSAP* for testing reasons. It shows the instances of the view class that is selected in object navigator.

4. Summary

The application of object–oriented views in the process of building an object–relational data warehouse system is very promising. Object–oriented views are important mechanisms providing an integrated access to data of various structures. It is due to the following facts:

- object–oriented features, especially methods, may facilitate data transformation, integration, and analysis in an object–relational data warehouse;

- the high expressiveness of an object–oriented data model makes object–oriented views suitable for the transformation and integration of many different data sources that use various data models.

Several approaches to object–oriented views were proposed in scientific publications (see [11, 12] for an overview). But they provide a limited functionality for the application in the process of data warehousing. Our *View Schema Approach* is more general and it encompasses other approaches. Moreover *VSA* allows to materialize a view schema and maintain its consistency by using one of the three maintenance techniques. *VSA* has been implemented as a prototype.

Further research within the *ORDAWA* project will focus on: query transformation and optimization techniques in data warehousing environment, techniques and algorithms for data mining in data warehousing environment, the maintenance of summary data for periodically changing dimension data as well as the maintenance of a data warehouse schema and data in the presence of changes made to the schemas of source databases [13].

References

[1] Widom J.: Research Problems in Data Warehousing, Proc. of the 4th Int. Conference on Information and Knowledge Management (CIKM), 1995, pp. 25-30

[2] J.Eder, H.Frank, T.Morzy, R.Wrembel, M.Zakrzewicz, Designing an Object-Relational Database System: Project ORDAWA. Proc. of challenges of the Enlarged 4th East-European Conference on Advances in Databases and Information Systems ADBIS-DASFAA 2000, Prague, Czech Republic, 2000, proc. of chalenges, pp.223-227

[3] Bukhres O. A., Elmagarmid A. (eds.): Object–Oriented Multidatabase Systems: A Solution for Advanced Applications, Prentice Hall, 1996

[4] Fankhauser P., Gardarin G., Lopez M., Munoz J., Tomasic A.: Experiences in Federated Databases: From IRO–DB to MIRO–Web. Porc. of the VLDB Conference, USA, 1998, pp. 655-658

[5] Inmon W. H.: Building the Data Warehouse, 2nd edition. John Wiley & Sons, 1996

[6] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P.: Fundamentals of Data Warehouses. Springer–Verlag, 2000, ISBN 3-540-65365-1

[7] Abiteboul S., Hull R., Vianu V.: Foundation of Databases. Addison–Wesley Publishing Company, 1995

[8] Wrembel R.: On Materialising Object–Oriented Views. In Barzdins J., Caplinskas A. (eds.): Databases and Information Systems. Selected papers. 4th International Baltic Workshop, Baltic DB&IS 2000, Lithuania, 2000. Kluwer Academic Publishers, March 2001, ISBN 0-7923-6823-1, pp. 15-28

[9] Wrembel R., Morzy T.: Incremental Maintenance of Materialised Object-Oriented Views by Using Deferred Technique. Proc. of the 15th International Symposium on Computer and Information Sciences – ISCIS'2000, Turkey, 2000, pp. 383-390

[10] Wrembel R.: Deriving consistent view schemas in an object-oriented database. Proc. of the 14th International Symposium on Computer and Information Sciences – ISCIS'99, Turkey, 1999, pp. 803-810

[11] Motschnig–Pitrik R.: Requirements And Comparison of View Mechanisms for Object-Oriented Databases. Information Systems, Vol. 21, No. 3, 1996, pp. 229-252

[12] Wrembel R.: Object–Oriented Views: Virtues and Limitations. Proc. of the 13th International Symposium on Computer and Information Sciences – ISCIS'98, Turkey, 1998, pp. 228-235

[13] Czejdo B., Messa K., Morzy T., Putonti C.: Design of Data Warehouses with Dynamically Changing Data Sources. In Proc. of the Southern Conference on Computing. The University of Southern Mississippi, October, 2000