The WHIPS Prototype for Data Warehouse Creation and Maintenance*

Wilburt J. Labio, Yue Zhuge, Janet L. Wiener, Himanshu Gupta
Hector Garcia-Molina, Jennifer Widom
Department of Computer Science
Stanford University
Stanford, CA 94305-2140, USA
http://www-db.stanford.edu/warehousing/warehouse.html

1 Overview

A data warehouse is a repository of integrated information from distributed, autonomous, and possibly heterogeneous, sources. In effect, the warehouse stores one or more materialized views of the source data. The data is then readily available to user applications for querying and analysis. Figure 1 shows the basic architecture of a warehouse: data is collected from each source, integrated with data from other sources, and stored at the warehouse. Users then access the data directly from the warehouse.

As suggested by Figure 1, there are two major components in a warehouse system: the *integration component*, responsible for collecting and maintaining the materialized views, and the *query and analysis component*, responsible for fulfilling the information needs of specific end users. Note that the two components are not independent. For example, which views the integration component materializes depends on the expected needs of end users.

Most current commercial warehousing systems (e.g., Redbrick, Sybase, Arbor) focus on the query and analysis component, providing specialized index structures at the warehouse and extensive querying facilities for the end user. In the WHIPS (WareHousing Information Project at Stanford) project, on the other hand, we focus on the integration component. In particular, we have developed an architecture and implemented a prototype for identifying data changes at heterogeneous sources, transforming them and summarizing them in accordance to warehouse specifications, and *incrementally* integrating them into the warehouse. We propose to demonstrate our prototype at SIGMOD, illustrating the main features of our architecture.

Our architecture is modular and we designed it specifically to fulfill several important and interrelated goals: data sources and warehouse views can be added and removed dynamically; it is scalable by adding more internal modules; changes at the sources are detected automatically; the warehouse may be updated continuously as the sources change, without requiring "down time;" and the warehouse is always kept consistent with the source data by the integration algorithms. More details on these goals and how we achieve them are provided in [WGL+96].

*This work was partially supported by Rome Laboratories under Air Force Contract F30602-94-C-0237; by an equipment grant from Digital Equipment Corporation; and by the Advanced Research and Development Committee of the Community Management Staff, as a project in the Massive Digital Data Systems Program.

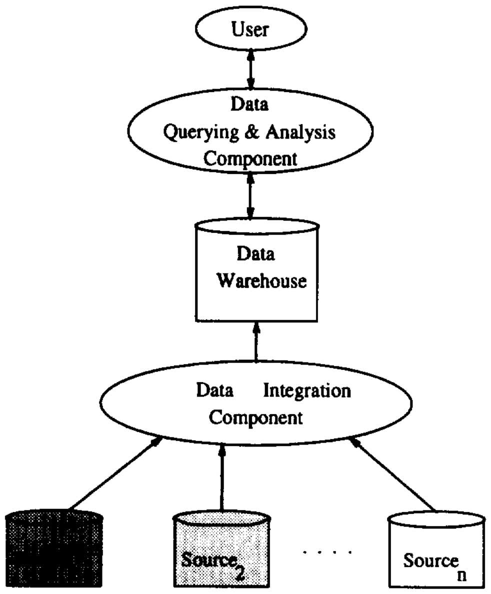
User
Data Querying & Analysis Component
Data Warehouse
Data Integration Component
Source 2
Source n

Figure 1: The basic architecture of a data warehouse system.

2 Whips architecture

In Figure 2 we expand the integration component of Figure 1 to depict the Whips system architecture. As shown in the figure, the system is composed of many distinct modules that communicate with each other although they potentially reside on different machines. We implemented each module as a CORBA object, using the ILU implementation of CORBA [CJS+94].

In the current prototype, we use the relational model to represent the warehouse data: views are defined in the relational model and the warehouse stores relations. The underlying source data is converted to the relational model by the source's monitor and wrapper before it is sent to any other module. We now overview the modules of the architecture.

Each source is encapsulated by a source-specific moni-

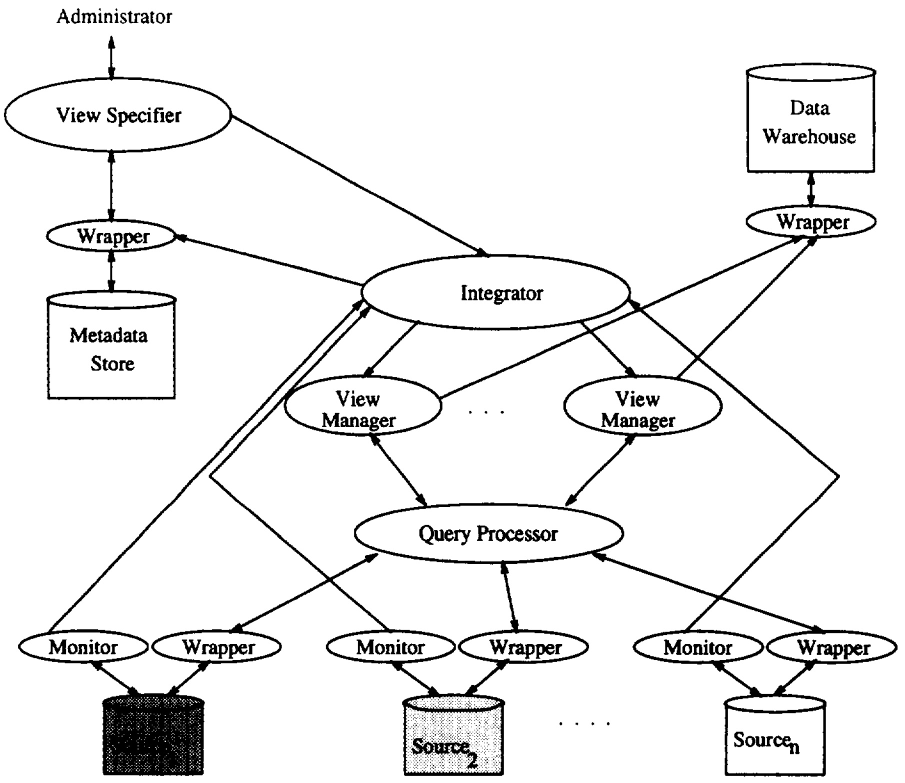

Figure 2: The Whips system architecture for warehouse maintenance.

tor and wrapper. The monitor is responsible for detecting modifications on the source data and notifying the integrator of them. Currently, we have implemented trigger-based monitors for cooperative (relational) sources, and snapshot monitors for flat file sources that only provide periodic snapshots of the source data. We describe algorithms for efficient change detection on snapshots elsewhere [LGM96]. We are working on adding IBM's DataCapture to the system; DataCapture is a log-based monitor which reads the log for DB2 and generates a table of source changes.

The wrappers translate single source queries from the internal relational representation used (which resembles relational algebra) to queries in the native language of its source. For example, a relational database wrapper would merely translate the relational algebra expression into SQL. A wrapper for a flat file Unix source might translate the algebra expression into a Unix *grep* for one selection condition, use postprocessing to apply further selection conditions and projections, and then transform the result into a relation. As stated above, using one wrapper per source hides the source-specific querying details from the query processor and all other modules: all wrappers support the same method interface although their internal code depends on the source.

The warehouse wrapper receives view definitions and modifications to the view data in a canonical (internal) format, and translates them to the specific syntax of the warehouse database. The wrapper thus shields all other modules in the Whips system from the particulars of the warehouse, allowing any database to be used as the warehouse. All modifications received by the warehouse wrapper in a single message are applied to the warehouse in one transaction, as needed by the Strobe view consistency algorithms [ZGMW96].

Views are defined in a subset of SQL that includes select-project-join views and aggregate views over all of the source data, without nesting. Optionally, the view definition may also specify which algorithm to use for view consistency. When a view is defined, the view specifier parses it, adds relevant information from the meta-data store (e.g., key attributes, attribute types), and sends it to the integrator.

The integrator coordinates system startup, including new source additions, and view initialization. However, the main role of the integrator is to facilitate view maintenance, by figuring out which source modifications need to be propagated to which views. To do so, the integrator uses a set of rules that specify which view managers are interested in which modifications. These rules are generated automatically from the view tree when each view is defined. In the simplest case, the rules dictate that all modifications to a relation over which a view is defined are forwarded to the corresponding view manager.

There is one view manager module responsible for maintaining each view, using one of the Strobe algorithms (as specified in the view definition) to maintain view consistency [ZGMW96]. The different Strobe algorithms yield different levels of consistency depending on the modification frequency; all of the algorithms require keeping track of the sequence of modifications and compensating query results for modifications that may have been missed.

The query processor receives global queries from the view managers and poses the appropriate single-source queries to the source wrappers to answer them. It then passes the composite global query answers back to the view managers. In general, the query processor performs distributed query processing, using standard techniques such as sideways information passing and filtering of selection conditions [OV91] to prune the queries it poses to the wrappers. It tracks the state of each global query while waiting for local query results from the wrappers.

3 Demo

The demo provides a graphical user interface (GUI) to the user-visible modules of the Whips system: the view specifier, the sources, and the warehouse; as well as access to a system log that records all internal module communication. By examining the system log, we can see the sequence of steps in propagating changes. The demo consists of three phases.

Phase 1: The system is started up and the modules identify themselves to each other. Also, an initial set of sources is defined. The contents of these sources may then be examined.

Phase 2: A view is defined at the warehouse. The view definition triggers the creation of a new view manager, which then initializes the view at the warehouse.

Phase 3: Source changes are artificially introduced through the GUI. We can then see the changes propagated through the modules until the view(s) are updated at the warehouse.

The current demo uses a financial scenario with three sources: Daily, Price-Earnings, and Portfolio.

1. **Daily** is a flat file dump of the daily stock feed. Each tuple (after conversion to the relational model) contains the date, stock id, high price, low price, closing price, and trade volume for a given stock. (In the current demo, we only include data for 1 day.)
2. **Price-Earnings (PE)** is the current monthly price-to-earnings ratio of various stocks, which is a rough indicator of how good or bad it is to own each stock. Each tuple contains the stock id and PE for that stock. We obtain the PE's from a Dialog source [Kni95].
3. **Portfolio** is the personal stock portfolio holdings of a customer. Each tuple contains the stock id, price paid, number of shares held, and date on which the stock was bought. The Sybase relational database is used for the Portfolio source.

The warehouse is also stored in a relational database, currently Sybase. During the demo, two views are defined. *Bad_stocks* is defined as the join of all three source relations where the customer paid more than today's closing price for the stock and the stock's PE is greater than 60. *Daily_copy* is defined as a copy of the Daily relation. As changes are introduced into the PE and Daily relations during the demo, these changes are detected and propagated into both views.

4 View maintenance example

The following example illustrates the type of activity that is visible through our GUI during a WHIPS demonstration. In particular, this example shows in detail how one source update is detected and propagated to the warehouse view. Before this example, the PE for company MSS was 50 and the *Bad_stocks* relation does not contain any tuple for MSS.

1. The **PE** relation is updated: the PE of MSS is changed from 50 to 66. The monitor for the PE source detects this change and reports it to the integrator.
2. The integrator determines that the view manager for *Bad_stocks* (VM1) is interested in changes to PE and forwards the change to VM1. It also determines that VM2, the view manager for *Daily_copy*, will be unaffected by the change and does not notify VM2.
3. View manager VM1 receives the update and breaks it into a deletion of [MSS, 50] from and an insertion of [MSS, 66] into **PE**. The deletion can be applied to the warehouse view directly, since MSS is the key for **PE**. However, to compute the view change for the insertion, the join of [MSS, 66] with **Daily** and **Portfolio** must be computed. VM1 sends the query $Q = Daily \bowtie [MSS, 66] \bowtie Portfolio$ to the query processor.
4. The query processor receives Q, extracts from it the single source query $Q1 = Daily \bowtie [MSS, 66]$, and sends $Q1$ to the Daily wrapper.
5. The Daily wrapper computes the answer to $Q1$ and sends it back to the query processor. The query processor then integrates the answer into another single-source query and sends to the Portfolio wrapper: $Q2 = [MSS, 10/26/96, 29.0, 66] \bowtie Portfolio$. The number 29.0 is the daily closing of MSS on that day.
6. The Portfolio wrapper computes the answer to $Q2$ and sends it back to the query processor. The query processor takes the now complete answer to Q, $[MSS, 10/26/96, 29.0, 66, 73]$ and sends it back to VM1.
7. VM1 receives the answer to Q, notices that it should be in the view. Therefore, VM1 sends to the warehouse wrapper the deletion of [MSS, 76] and the insertion of $[MSS, 10/26/96, 29.0, 66, 73]$.
8. The warehouse wrapper receives the updates to view *Bad_stocks* and updates the view correctly.

References

[CJS+94] A. Courtney, W. Janssen, D. Severson, M. Spreitzer, and F. Wymore. Inter-language unification, release 1.5. Technical Report ISTL-CSA-94-01-01 (Xerox accession number P94-00058, Xerox PARC, May 1994.

[Kni95] Knight-Ridder Information, Inc., *Dialog Homepage*, 1995. http://www.dialog.com/dialog/dialog1.html.

[LGM96] W. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms in data warehousing. In *VLDB Conference*, pages 63–74, September 1996.

[OV91] M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[WGL+96] J.L. Wiener, H. Gupta, W.J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. A system prototype for warehouse view maintenance. In *The Workshop on Materialized Views*, pages 26–33, Montreal, Canada, June 1996.

[ZGMW96] Y. Zhuge, H. Garcia-Molina, and J.L. Wiener. The Strobe algorithms for multi-source warehouse consistency. In *PDIS Conference*, pages 146–157, Miami Beach, Florida, December 1996.