Evolvable View Environment (*EVE*): Non-Equivalent View Maintenance under Schema Changes *

E.A. Rundensteiner, A. Koeller, X. Zhang
A. VanWyk, Y. Li
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609
{rundenst,koeller,xinz,avanwyk,yongli}@cs.wpi.edu

A.J. Lee
Dept. of Information and Systems Mgmt.
HK University of Science and Technology
Clear Water Bay, Hong Kong
alee@ust.hk

A. Nica§
Department of EECS
University of Michigan
Ann Arbor, MI 48109
anica@eecs.umich.edu

1 Overview

Supporting independent ISs and integrating them in distributed data warehouses (materialized views) is becoming more important with the growth of the WWW. However, views defined over autonomous ISs are susceptible to schema changes. In the *EVE* project we are developing techniques to support the maintenance of data warehouses defined over distributed *dynamic* ISs [5, 6, 7]. The *EVE* system is the first to allow views to survive *schema changes* of their underlying ISs while also adapting to changing data in those sources. *EVE* achieves this in two steps: applying view query rewriting algorithms that exploit information about alternative ISs and the information they contain, and incrementally adapting the view extent to the view definition changes. Those processes are referred to as *view synchronization* and *view adaption*, respectively. They increase the survivability of materialized views in changing environments and reduce the necessity of human interaction in system maintenance.

1.1 Background: Relaxed Query Semantics and Meta Information

E-SQL or *Evolvable-SQL* is an extension of SQL that allows the view definer to express preferences for view evolution [5, 8, 6]. A user defining a view can specify what information is indispensable, what information is replaceable by similar information from other ISs, and whether a changing view extent is acceptable. Relaxed query semantics provided by E-SQL is the key to obtaining *non-equivalent* but useful query rewritings as it provides the *EVE* system with the flexibility to evolve a view under schema changes in a controlled way while preserving the user's intended semantics.

In order to enable view adaptation and rewrite view definitions affected by IS evolution, our system needs to be able to identify view component replacements from other ISs. Through a model for meta knowledge, we express relationships between ISs using constraints (e.g., agreeing data types, functional dependencies between attributes, extent overlaps between relations). These descriptions form an information pool that is critical in finding appropriate replacements for view components (i.e., attributes, relations, WHERE-conditions) when view definitions become undefined. The meta knowledge about the information space is stored in our system in the *Meta Knowledge Base (MKB)* whereas E-SQL view definitions are stored and maintained in the *View Knowledge Base (VKB)* as depicted in Figure 1.

2 Concepts Presented

2.1 Non-Equivalent View Synchronization

An ideal (*equivalent*) rewriting should preserve the original view interface and generate an identical result set (view extent) without introducing any surplus tuples nor dropping any of the original ones, and it should be efficiently maintainable in the long run. Locating such a perfect rewriting is difficult and sometimes impossible in practical environments. Thus, we relax the requirement of rewriting view queries to now also generate non-equivalent view definitions. We specify through E-SQL which deviations in view extents and view interfaces are acceptable to a user.

Depending on the type of meta information maintained in the MKB and the relaxed query semantics specified by E-SQL, we apply different *view synchronization algorithms* [6] in order to obtain new legal query rewritings (i.e., queries that can be exe-

*This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 94–57609, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 97–32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award and for the IBM corporate fellowship for one of her graduate students.
§Current address: Sybase Inc., Waterloo, Ontario, Canada N2L 3X2.

cuted against the changed information space and fulfill the given user requirements). Algorithms developed for *EVE* include *Project-Containment View Synchronization (POC)* [7], *Complex View Synchronization (CVS)* [6], and *Optimized CVS* [2]. The *EVE*-demo applies these algorithms to generate view rewritings.

2.2 Tradeoffs of Cost and Quality

After a schema change, the *EVE*-demo adapts all affected materialized views to work on the new information space. The view extents and schemas contained in the data warehouse after the schema change may differ from the original. Schemas may differ in the set of attributes retained in the view, and view extents may contain surplus tuples and may lack tuples that have been in the view before. We have introduced a measure of *quality* (non-equivalent maintenance) [3] which allows to compare view rewritings with the original view and determine a ranking among alternative views. A value estimating this "usefulness" of a rewriting to a view user plus an assessment of future view maintenance costs is given for a view rewriting (QC-Value) [3, 4]. In the demo, the best view rewritings according to the QC-Value are ordered by preference and presented to the user after a schema change.

2.3 Scalable View Maintenance and Adaption

Our long-term goal is the development of scalable middle-ware technology in support of efficient distributed data warehousing applications. We have developed incremental view maintenance techniques that parallelize the handling of concurrent data updates (as an extension of the sequential SWEEP algorithm [1]). Our algorithm, called Parallel SWEEP or PSWEEP, significantly increases the data maintenance performance and thus guarantees up-to-date view extents even in dynamic distributed environments. This algorithm is presented in the demo.

In a related effort, we have also incorporated incremental strategies for the adaptation of the view extent of a data warehouse whose information sources have undergone dynamic schema changes. The adaptation algorithm efficiently adapts the extent of the view after the view synchronization, hence avoiding expensive re-computations. One issue addressed is the correct execution of such maintenance even over ISs that are themselves affected by changes during the maintenance task. We are also working on supporting concurrency between schema changes and data updates [9].

3 *EVE*-Demo Implementation

We will briefly describe the main functionality of the demo. Essentially, the interface to the *EVE* system is divided into two parts: a *meta and view knowledge browser* and a *schema change request interface*. This software implements a fully functional data warehouse tool suite over distributed information sources. The *EVE* demo homepage is available under `http://davis.wpi.edu/dsrg/EVE`.

- **Meta and View Knowledge Browser**

  The *MKB/VKB browser* displays the current states of the Meta Knowledge Base and the View Knowledge Base (Figure 2). Information displayed includes the set of information sources and relations in the information space and the constraints on each relation or pair of relations, as well as the views defined in the data warehouse and their extents. The extent of each base relation can also be browsed.

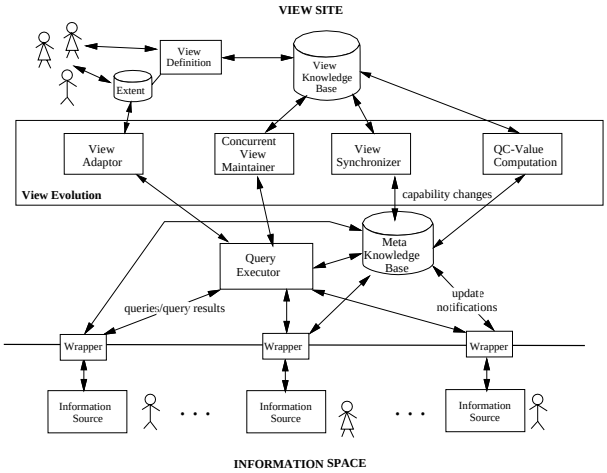

Figure 1: The *EVE* Framework: View Synchronization in an Evolving Environment.

- **Schema Change Request Interface**

  With this interface, relational schema changes (such

Figure 2: The Meta Knowledge Browser.

as `add-attribute`, `delete-relation`) can be initiated. Each schema change can be applied to any relation or attribute defined in the MKB. The schema change is executed against the IS specified and the *EVE* system is then notified of the schema change which causes the view synchronization to take effect. The effect of the schema change (view synchronization as well as view maintenance) can immediately be seen in the following windows (Figure 3), but can also be browsed later in the MKB/VKB browser.

Figure 3: An Example for a Changed View Extent After a Schema Change.

3.1 EVE System Implementation

The demo is implemented in Java 1.1 using Java Foundation Classes (*Swing*). It has a three-tier architecture consisting of a GUI, a middleware layer and a database layer. The middleware connects to several SQL-Databases through JDBC and contains modules to compute data warehouse extents, maintain view knowledge, meta knowledge, parse E-SQL, apply view synchronization and compute QC-Values. The GUI consists of two applets for schema change requests and meta/view knowledge browsing. The whole system currently consists of over 160 Java classes and runs under JDK 1.1.6 on Windows NT 4.0 as well as Linux 2.0.

References

[1] D. Agrawal, A. El Abbadi, and A. Singh. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[2] A. Koeller, E. A. Rundensteiner, and N. Hachem. Integrating the Rewriting and Ranking Phases of View Synchronization. In *Proceedings of the ACM First International Workshop on Data Warehousing and OLAP (DOLAP'98)*, November 1998.

[3] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. In *Proceedings of IEEE International Conference on Data Engineering*, 1999.

[4] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Non-Equivalent Query Rewritings. In *International Database Conference*, Hong Kong, July 1999.

[5] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping Virtual Information Resources Up and Running. In *Proceedings of IBM Centre for Advanced Studies Conference (CASCON'97), Best Paper Award*, pages 1–14, November 1997.

[6] A. Nica, A. J. Lee, and E. A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. In *Proceedings of International Conference on Extending Database Technology (EDBT'98)*, pages 359–373, Valencia, Spain, March 1998.

[7] A. Nica and E. A. Rundensteiner. Using Containment Information for View Evolution in Dynamic Distributed Environments. In *Proceedings of International Workshop on Data Warehouse Design and OLAP Technology (DWDOT'98)*, Vienna, Austria, August 1998.

[8] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

[9] X. Zhang and E. A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. In *Proceedings of IEEE International Conference on Data Engineering*, 1999.