History-Driven View Synchronization

Andreas Koeller and Elke A. Rundensteiner*

Department of Computer Science
Worcester Polytechnic Institute
100 Institute Road. Worcester, MA 01609–2280
{koeller|rundenst}@cs.wpi.edu

Abstract. Views over distributed information sources rely on the stability of the schemas of these underlying sources. In the event of meta data changes in the sources, such as the deletion of a table or column, such views may become undefined. Using meta data about information redundancy and user preferences, views can be evolved as necessary to be defined on a modified information space after a source meta data change, while assuring the best possible compliance to preferences that users may have.
Previous work in view synchronization focused only on deletions of schema elements. We now offer the first approach that makes use of additions also. The algorithm is based partly on returning view definitions to previous versions by "backtracking" in the history of views and meta data. This technology enables us to adapt views to temporary meta data changes by cancelling out opposite changes and allows undo/redo-operations on meta data without deteriorating the quality of the view. The mechanism described in this paper will therefore improve the quality of evolved views.

1 Introduction

In recent years, the number of digital information storage and retrieval systems has increased immensely. Many of those database systems are available in some kind of network, and the task of integrating data from different source databases is an increasingly important one. Solutions to this problem include data warehouses, a term which in the database community refers to materialized views over distributed information sources. Newer developments, such as the ubiquity of the World Wide Web, emphasize the independence of data producers from data consumers. Independent data producers or providers have control over the schema of their information sources which raises the question of the influence of *meta data updates* (deletions of attributes or relations in underlying databases)

* This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 97–96264, 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.

on a view [RKZ00]. In traditional views (as introduced in the literature handling data updates, e.g., [BLT86, AASY97]), meta data updates can render a view definition undefined.

Some work has been done in the *EVE*-Project [RLN97, NLR98, KRH98] that deals with the rewriting of views under schema updates in the underlying sources in the case of *schema element deletions*, retaining all or a part of the view extent in the new rewritten view. We described algorithms allowing generated view rewritings to be non-equivalent to the original query [LNR97], and presented a model for a numeric assessment of the quality and cost of such rewritings [LKNR99], allowing for the automatic selection of the best view rewriting among many generated ones.

One drawback of the previous work mentioned is the finality of a view synchronization operation. After the deletion of an underlying relation that has been used by a view, the view is rewritten to not refer to that relation any more. Even if the same relation is later added back to the information space (for instance, after a temporary unavailability due to a network problem), it will never be used by the view again since without a global domain model the view synchronization algorithm cannot determine in what relationship a new data element stands to other previously available elements.

Note that with all view synchronization algorithms that have been devised so far, only one-step view synchronization is possible (cf. Section 3). The existing algorithms POC/SPOC [NR98], SVS [RLN97], CVS [NLR98], and GRASP [NR98] cannot synchronize a view under a sequence of schema changes. Rather, they generate a new view definition after each schema change in an information source, without taking any information about earlier IS changes into account. Also, they do not support any meaningful synchronization under *add*−capability-changes. This work describes a way to overcome this shortcoming of previous approaches.

In this paper, we present a complete approach capable of handling all information source schema changes, namely *adds*, *renames*, and *deletes* of attributes and relations. The main contribution of this paper is the utilization of additional available meta data to keep views as close to their original definition as possible, under a sequence of meta data changes that occurs over time.

2 Related Work

Materialized views over distributed information sources have been explored for a number of years. First work focused on questions of materialized view maintenance under data updates in the sources [BLT86, GM95]. Work on maintenance of the more recent mediator-based (constrained) heterogeneous database systems was done by Lu et al. [LMSS95]. More recently, questions of optimizing view queries given varying parameters or capabilities of underlying sources have also been explored [AASY97, GMHI^{+}95]. Generally, work in this area assumes that the rewritten view query computes a view extent *equivalent* to the original one. Prominent approaches that deal with equivalent query rewriting include work by Jarke et al. [JK84] or Agrawal et al. [AASY97]. Important work on integrating heterogeneous sources in one view using a common semistructured data model (OEM) has been done in the TSIMMIS-project [GMHI+95]. Chaudhuri and Shim [CS96] present work on query optimization under user-defined SQL predicates (e.g., stored procedures), which deals with finding optimized query plans for such queries but does not consider changing or non-equivalent queries. Finkelstein [Fin82] describes a model to achieve query optimization in an environment in which a sequence of related (similar) queries is executed on a query engine (using *query-graphs*).

Work on rewriting queries using views (e.g., [LMSS93]) is used in subsequent work by Levy et al. which is closely related to ours in terms of its goal, but not the approach taken. In [LSK95], the notion of the *world-view* is introduced as a global, fixed domain model of a certain part of the world on which both information providers and consumers must define views. This work is in some sense an approach inverse to ours [RLN97] but relies on the existence of fixed, invariant "world model" which is unlikely to be available in practice. Furthermore, changes to the world model are not possible in this approach or would require a manual redefinition of both information providers' and consumers' queries.

Related approaches to Levy's are Arens et al. [AKS96] and the SoftBot project [EW94]. The SoftBot project has a different approach to query processing as they assume that the system has to discover the "links" among data sources that are described by action schemas. While related to our view synchronization algorithm CVS [NLR98], the SoftBot planning process also relies on discovering connections among information sources when very different source description languages are used. Neither SIMS nor SoftBot address the particular problem of evolution under capability changes of participating information sources.

3 Background: The EVE-System

Since our work is based on the *EVE*-Project [RLN97], we give a short overview over those elements of the EVE-System that we need as background in this paper. The algorithm described in this paper builds on top of the *one-step* synchronization algorithms as described in the *EVE*-Project [RLN97, NLR98, KRH98]. As we will see, the new algorithm in some cases falls back on these old algorithms and performs one or a sequence of single-step view synchronizations.

E-SQL or *Evolvable-SQL* is an extension of SQL that allows the view definer to express preferences for view evolution [LNR97]. A user defining a view can specify what information is dispensable and what information is replaceable by similar information from other information sources (ISs), and whether a changing view extent is acceptable. Those user preferences are expressed as parameters that can be attached to basic SQL query elements. For example, the qualifier AD declares an attribute dispensable, RR means a replaceable relation, and $\mathcal{VE}$ (which is one of $\subseteq, \supseteq, \equiv, \approx$) expresses the relationship between the original and rewritten query as required by the view definer. The default for $\mathcal{VE}$ is $\equiv$, for all other parameters it is *false*.

Example 1. A typical E-SQL query would be:

```
CREATE VIEW Asia-Customer (VE = "⊇") AS
SELECT        Name, Address, Phone (AD = true, AR = true)
FROM          Customer C (RR = true), FlightRes F
WHERE         (C.Name = F.PName) AND (F.Dest = 'Asia') (CD = true)
```

View definitions are stored in a *View Knowledge Base (VKB)*. Once a view is defined, *EVE* view synchronization algorithms are invoked after schema changes in all ISs participating in this view and attempt to find replacements for missing view elements. For the purpose of identifying view element replacements from other ISs, we express relationships between ISs through constraints (e.g., agreeing data types, functional dependencies between attributes, and extent overlaps between relations, all of which are stored in the *Meta Knowledge Base (MKB)*). One important constraint is the *containment-constraint* between two relations, stating that a (horizontal and/or vertical) fragment of a relation is semantically contained or equivalent to a (horizontal and/or vertical) fragment of another. Examples of view synchronization can be found in the technical report to this paper [KR00].

Since view synchronization algorithms may generate many possible query rewritings, one needs to be selected as the new view definition. For this purpose we have developed the QC-Model [LKNR99] to estimate the *quality* and *cost* of the rewritings. Each legal query rewriting will in general preserve a different amount (extent) and different types (interface) of information, which we measure through a metric that we have defined on views and refer to as the *quality* of the view. Also, each new view query will cause different *view maintenance costs*, since in general data will have to be collected from a different set of ISs. With these two dimensions, the QC-Model can assign a numerical *QC-Value* to each query rewriting and thus *compare* different rewritings with each other.

4 Modeling Meta Data and Meta Data Changes

4.1 Meta Data Model

In the *EVE*-System as introduced by [RLN97], View and Meta Knowledge Bases (VKB and MKB) exist to maintain data warehouses over changing information sources. MKB data is encoded in the form of schema definitions and constraints. In this sense, we will describe schemas of underlying information sources by Attribute Constraints and Relation Constraints, and for the purpose of this paper, we will concentrate on a total of four different constraints that comprise our MKB: *attribute constraints* ($\mathcal{AC}$) define attribute names, *relation constraints* ($\mathcal{RC}$) define relation names, *containment constraints* ($\mathcal{CC}$) define relationships between relations, and *containment pair constraints*($\mathcal{CPC}$) relate attributes in those relations to each other (i.e., define projections of relations who are in a set relationship to each other). We also introduce two functions that relate attributes with their relations (rel: $\mathcal{AC} \rightarrow \mathcal{RC}$) and $\mathcal{CPC}$s with their $\mathcal{CC}$s (con: $\mathcal{CPC} \rightarrow \mathcal{CC}$). For details, refer to our technical report [KR00].

4.2 Changes in Meta Data

Changes to the schemas of sources which are modeled in the Meta Knowledge Base cause MKB and VKB changes. This paper describes how to synchronize view definitions with such *meta data changes* under consideration of previous meta data changes in the information space, and how to make use of added meta knowledge by "re-installing" a previously deleted schema element. In order to better specify the meaning of "previously deleted element", we introduce the concept of *history* of view and meta knowledge to our system, starting with a formal definition of schema changes.

Definition 1 (meta data change). *Let $\mathcal{C}_i$ be a constraint ($\mathcal{AC}$, $\mathcal{RC}$, $\mathcal{CC}$, or $\mathcal{CPC}$, cf. Section 4.1). A* meta data change $\mathcal{MDC}_i$ *is the addition or deletion of $\mathcal{C}_i$ to or from the Meta Knowledge Base, transforming the MKB from a previous state MKB_i to a subsequent state MKB_{i+1}. Valid meta data changes are:* delete-$\mathcal{CPC}$ $(\mathcal{C}_i)$, delete-$\mathcal{CC}$ $(\mathcal{C}_i)$, delete-$\mathcal{RC}$ $(\mathcal{C}_i)$, delete-$\mathcal{AC}$ $(\mathcal{C}_i)$, add-$\mathcal{CPC}$ $(\mathcal{C}_i)$, add-$\mathcal{CC}$ $(\mathcal{C}_i)$, add-$\mathcal{RC}$ $(\mathcal{C}_i)$, add-$\mathcal{AC}$ $(\mathcal{C}_i)$. *Each $\mathcal{MDC}_i$ has as a parameter exactly one constraint in the MKB which is denoted by* constraint$(\mathcal{MDC}_i)$.

We also introduce the notion of an *inverse* meta data change operation. Two meta data changes are *inverse* to each other if they are add- and delete-operations, respectively, and refer to the same constraint (e.g. *delete-*$\mathcal{RC}(\mathcal{RC}_1) = ($*add-*$\mathcal{RC}(\mathcal{RC}_1))^{-1}$).

4.3 Evolution of the MKB under Meta Data Changes

For the eight operations in Definition 1, the respective constraint is simply added to or removed from the MKB. In addition to that, some changes imply others. For example, the meta data change operations *delete-*$\mathcal{RC}$ and *delete-*$\mathcal{CC}$ trigger deletions of Attribute Constraints and Containment Pair Constraints, respectively. Those implicit operations are executed on the MKB immediately after the meta data change is detected, but before the view synchronization algorithm is run on the view definitions in the VKB. Similarly, we assume (or generate) subsequent additions of sub-constraints in the same fashion for *add-*$\mathcal{RC}$ and *add-*$\mathcal{CC}$-operations. In the case that the set of attributes in a relation has changed while this relation was unavailable, we assume that appropriate *add-*$\mathcal{AC}$ or *delete-*$\mathcal{AC}$ operations are generated after the *add-*$\mathcal{RC}$ operation. Note that renames of relations or attributes cause all $\mathcal{RC}$ or $\mathcal{AC}$ in all MKB states to be renamed at once and do not constitute a $\mathcal{MDC}$ in the sense of Definition 1.

4.4 Dependency of Views on Meta Data

In previous view synchronization algorithms [RLN97, NLR98, KRH98] the rewriting of a view is accomplished with the help of constraints that state (partial) redundancy in the information space. For instance, a containment constraint between R and S can be used to replace relation R by relation S in a view if the former becomes unavailable. The meaning of that constraint is that the information provider(s) of relations R and S guarantee that the constraint is and remains valid. Even when relation R becomes unavailable, this condition will still be kept valid. If this $\mathcal{CC}$-constraint would also be deleted from the MKB, it could not be utilized by the view synchronization process. We define a rewritten view $V^{(1)}$ to be *constraint-dependent* on a $\mathcal{CC}$-constraint if the latter was used in the above sense in the rewriting of a view V. A formal definition can be found in [KR00]. Note that any further rewriting of view $V^{(1)}$ can, but does not have to, retain the view's constraint-dependency on any constraint.

4.5 Modeling the History of Meta Data and Views

We will now define a way to model the *history* of meta data and views, i.e., the sequence of changes in meta data that have occurred on a MKB and VKB.

Definition 2 (history graph). *Let a view V be rewritten into $V^{(n)}$ by a meta data change sequence $\mathcal{S} = \{\mathcal{MDC}_1, \mathcal{MDC}_2, \ldots, \mathcal{MDC}_n\}$. The* history $\mathcal{H}(V^{(n)})$ *of view rewriting $V^{(n)}$ is a graph $\mathcal{H} = (N, E)$ with N the set of nodes representing the states of both MKB and VKB at a certain point in time, and E the (labeled directed) edges representing transitions from one state to the next (meta data changes). Each node N_i is a pair $<MKB_i, VKB_i>$ and each edge E_i is labeled with the meta data change $\mathcal{MDC}_i$ it represents. Node N_0 contains the original Meta Knowledge and View Knowledge Bases (i.e., the meta knowledge at view definition time and the originally defined views) while N_n represents the current state of the system.*

5 The History-Driven View Synchronization Algorithm

View synchronization has to be performed when a view is affected after a schema change in the information space. Views whose definitions are still valid after the change do not have to be synchronized. A view V is affected by a schema change if (1) a constraint on which V was constraint-dependent is added to the MKB or (2) a constraint on which V is currently constraint-dependent is removed from the MKB.

The History-Driven View Synchronization Algorithm (HD-VS), which is executed after a schema change for each affected view, uses three main concepts: backtracking in the history of a view, re-applying a part of a meta data update sequence from that history, and reconstructing part of the view's history graph in the process of re-application of meta data changes. Figure 1 gives an example of the algorithm. Four meta data changes are executed on a view V. After each meta data change, HD-VS checks whether the history of the view contains information that could be used to rewrite view V in an appropriate way. Two consecutive *delete-constraint* operations both lead to backtracking in the history of the view but do not find an appropriate inverse operation to cancel with. In both cases, HD-VS falls back on a single-step view-synchronization algorithm (e.g., POC). The third operation, an *add-constraint* operation, also leads to unsuccessful backtracking but in the case of an addition to the information space, no POC-view synchronization is *necessary* for the view to remain valid. The fourth operation, an *add-operation* inverse to a previous *delete*-operation, leads to successful backtracking. Here, the two operations *add-X* and *del-X* cancel out. HD-VS applies a meta data -sequence to V that does not contain these two operations, but all others. Then, the remaining two operations (*del-Y* and *add-U* in our example) are applied in the above mentioned fashion, but do not lead to further cancellations, so the final view obtained is $V_2^{(2)}$.

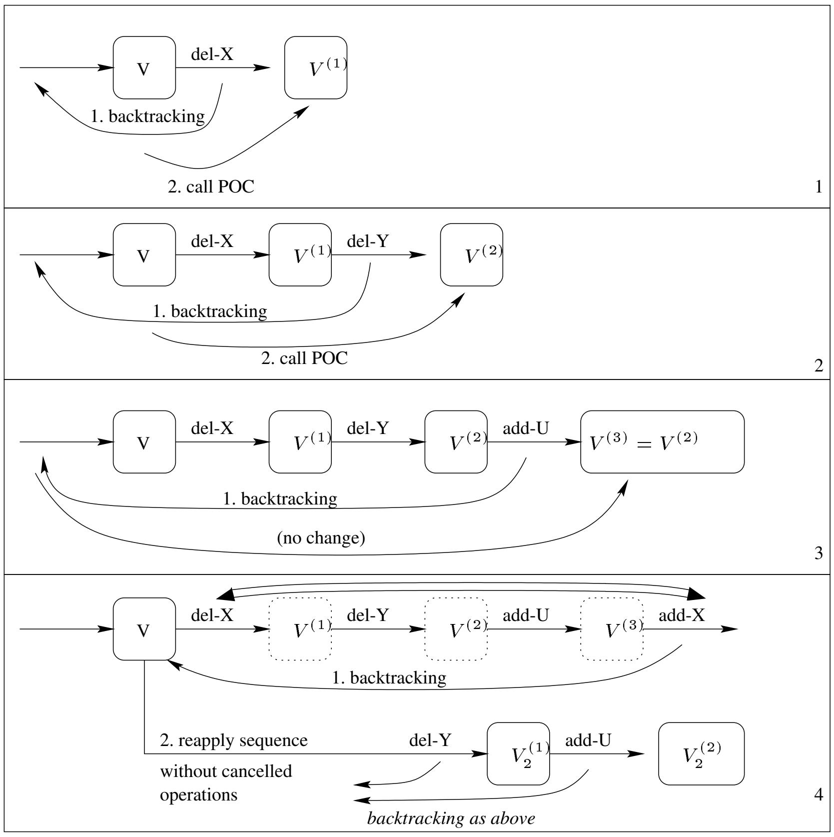

Fig. 1. Example Sequence for the History-Driven View Synchronization Algorithm (HD-VS).

5.1 Backtracking in the History of a View

In order to perform view synchronization, HD-VS applies a technique that we will call *backtracking*. The purpose of the backtracking process is to find a view

V_i in the history $\mathcal{H}(V^{(n)})$ of V from which to re-apply or *redo* a changed meta data update sequence. The idea of backtracking now consists in finding a node in $\mathcal{H}(V)$ for which a certain condition C holds.

Definition 3 (backtracking). *Backtracking in the history $\mathcal{H}(V^{(n)})$ of a view rewriting $V^{(n)}$ after a meta data change $\mathcal{MDC}_{n+1}$ to a constraint $\mathcal{C}_i$ is the traversal of the graph $\mathcal{H}(V^{(n)})$, starting from node N_n through nodes $N_{n-1}, N_{n-2}, \ldots$, until the following condition is satisfied:*

1. *if $\mathcal{MDC}_{n+1}$ is an* add-constr.*-oper.: $V^{(i)}$ is constraint-dependent on $\mathcal{C}_i$ or*
2. *if $\mathcal{MDC}_{n+1}$ is a* delete-constr.*-oper.: $V^{(i)}$ is* ***not*** *constraint-dependent on $\mathcal{C}_i$.*

The backtracking process returns the node N_i with the highest index i (i.e., the most recent view) satisfying the above condition.

The motivation behind this definition of backtracking is as follows: If a constraint $\mathcal{C}_i$ is added to the MKB, we want to determine whether this constraint has previously been used in the history of view V. In this case, $\mathcal{C}_i$ must have been deleted from the MKB earlier and we can *cancel* the add with the delete of $\mathcal{C}_i$. Therefore, we backtrack to a view $V^{(i)}$ that used $\mathcal{C}_i$, since $V^{(i)}$ must precede the deletion of constraint $\mathcal{C}_i$. In the opposite case of a delete-constraint-operation on constraint $\mathcal{C}_i$, we look for the last view $V^{(i)}$ that was not constraint-dependent on $\mathcal{C}_i$ since we can then re-apply the remaining schema change sequence from $V^{(i)}$ on under the assumption that $\mathcal{C}_i$ was never part of the MKB.

5.2 Re-applying a Meta Data Update Sequence

After backtracking to a certain view $V^{(i)}$, a meta data change sequence S' derived from the original sequence S has to be applied to $V^{(i)}$. In the first case of *add-constraint*-operations, we cancel out the corresponding previous delete-operation from S and apply this shorter sequence. In the second case of *delete-constraint*-operations, the constraint C is removed from the MKB. The sequence S is then applied to this changed MKB, without C being available for any view synchronization step. In the case of a *delete-$\mathcal{CC}$* or *delete-$\mathcal{RC}$*, all dependent $\mathcal{CPC}$ or $\mathcal{AC}$ are also deleted from the MKB and subsequent add-$\mathcal{CPC}$- or add-$\mathcal{AC}$-operations are removed from S. A formal definition of this process is given in [KR00].

5.3 Reconstructing the History Graph

As the sequence S' is applied to MKB_i, new nodes $N_{i+1}, \ldots, N_j (j \leq n)$ are created and old nodes are discarded from the graph. Lastly, a new node N_{j+1} is added to $\mathcal{H}(V)$, containing MKB_{j+1} and VKB_{j+1} and connected with N_j through an edge E_{j+1} labeled with meta data change $\mathcal{MDC}_{j+1}$.

In the technical report [KR00], we handle the case when backtracking reaches the beginning of the meta-data change sequence. We also analyze details of the different meta data change operations, such as co-dependencies between different operations, and give examples for a number of cases. This is omitted here due to the limited space.

6 Evaluation

In the technical report accompanying this paper [KR00], we give several theorems on the correctness and advantages of the HD-VS algorithm. We prove that (1) after a valid meta data change that does not lead to backtracking, our algorithm will give a valid view, and (2) backtracking and re-applying a meta-data change sequence as discussed in Section 5 will always lead to valid view rewritings. We also show that under both add- and delete-schema-changes, our algorithm will give equal or better view rewritings than previous algorithms.

7 Conclusion

In this paper, we present a solution for view synchronization under meta data updates. With our algorithm, a view that is defined over a given set of relations and attributes can be synchronized with *additions and deletions* of both. Previous approaches [RLN97, NLR98] to this problem could only adapt views to *deletions* in the information, but not to additions, so that, for example, temporary unavailability of information sources could not be accounted for. This technology can be applied to views defined over distributed information sources, especially as those sources are independent and prone to change their schemas or query capabilities.

Acknowledgments. We would like to thank all members of the DSRG research group at WPI. In particular, we are grateful to Xin Zhang for helping to implement the EVE system. We also thank Amy Lee and Anisoara Nica from the University of Michigan for interactions on the EVE project.

References

[AASY97] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[AKS96] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6 (2/3):99–130, 1996.

[BLT86] J. A. Blakeley, P.-E. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. *Proceedings of SIGMOD*, pages 61–71, 1986.

[CS96] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In T. M. Vijayaraman et al., editors, *International Conference on Very Large Data Bases*, pages 87–98, Los Altos, CA 94022, USA, 1996. Morgan Kaufmann Publishers.

[EW94] O. Etzioni and D. Weld. A Softbot-based interface to the Internet. *Communications of the ACM*, 37(7):72–76, July 1994.

[Fin82] S. Finkelstein. Common subexpression analysis in database applications. In M. Schkolnick, editor, *Proceedings of SIGMOD*, pages 235–245, Orlando, Florida, 2–4 June 1982. ACM, New York.

[GM95] A. Gupta and I. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

[GMHI+95] H. García-Molina, J. Hammer, K. Ireland, et al. Integrating and accessing heterogeneous information sources in TSIMMIS. In *AAAI Spring Symposium on Information Gathering*, 1995.

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, pages 111–152, 1984.

[KR00] A. Koeller and E. A. Rundensteiner. A History-Driven Approach at Evolving Views Under Meta Data Changes. Technical Report WPI-CS-TR-00-01, Worcester Polytechnic Institute, Dept. of Computer Science, 2000.

[KRH98] 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.

[LKNR99] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Non-Equivalent Query Rewritings. In *International Database Conference*, Hong Kong, July 1999.

[LMSS93] A. Levy, I. S. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, query reachability and satisfiability in datalog extensions. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 109–122, Washington, DC, 25–28May 1993.

[LMSS95] J. J. Lu, G. Moerkotte, J. Schue, and V. S. Subrahmanian. Efficient maintenance of materialized mediated views. In M. J. Carey and D. A. Schneider, editors, *Proceedings of SIGMOD*, pages 340–351, San Jose, California, 22–25 May 1995.

[LNR97] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Framework: View Synchronization in Evolving Environments. Technical Report WPI-CS-TR-97-4, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[LSK95] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems - Special Issue on Networked Information Discovery and Retrieval*, 5(2):121–143, 1995.

[NLR98] 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.

[NR98] 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.

[RKZ00] E. A. Rundensteiner, A. Koeller, and X. Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, June 2000.

[RLN97] 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.