WPI-CS-TR-99-23 Aug. 1999

Enhancing Existing Incremental View Maintenance Algorithms Using the Multi-Relation Encapsulation Wrapper

by

Lingli Ding
Xin Zhang
Elke A. Rundensteiner

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Enhancing Existing Incremental View Maintenance Algorithms Using the Multi-Relation Encapsulation Wrapper

Lingli Ding, Xin Zhang, and Elke A. Rundensteiner

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
(lingli | xinz | rundenst)@cs.wpi.edu

Abstract

Data warehouses (DW) are built by gathering information from several information sources (ISs) and integrating and materializing it into one repository customized to user's needs. Some of the most recently proposed algorithms for the incremental maintenance of such materialized DWs, such as SWEEP and PSWEEP, offer several significant advantages over previous solutions, such as high-performance, no potential for infinite waits and reduced remote queries and thus reduced network and IS loads. However, similar to many other algorithms, they still have the restricting assumption that each IS can be composed of just one single relation. This is unrealistic in practice. In this paper, we hence propose a solution to overcome this restriction. The Multi-Relation Encapsulation (MRE) Wrapper supports multiple relations in information sources in a manner transparent to the rest of the environment. The Multi-Relation Encapsulation Wrapper treats one IS composed of multiple relations as if it were a single relation from the DW point of view; thus it can easily be plugged into existing incremental view maintenance algorithms without any change. Hence, our method maintains all the advantages offered by existing algorithms in the literature in particular SWEEP and PSWEEP, while also achieving the additional desired features of being non-intrusive, efficient, flexible and well-behaved.

1 Introduction

Data warehousing (DW) [WB97, Wid95, GMLWZ98, GM96] is a popular technology to integrate data from heterogeneous information sources (ISs) in order to provide data to for example decision support or data mining applications [CD97]. Once a DW is established, the problem of maintaining it consistent and up-to-date of the underlying ISs remains a critical issue. It is popular to maintain the DW incrementally [AAS97, MKK97, KLMR97] instead of recomputing the whole extent of the DW after IS updates due to the large size of DWs and the enormous overhead associated with the DW loading process.

In recent years, there have been a number of algorithms proposed for incrementally view maintenance, e.g, [ZGMHW95, ZGMW96, AAS97, ZDR99]. ECA [ZGMHW95] handles view maintenance under concurrent data updates of one centralized IS, while Strobe [ZGMW96], SWEEP [AAS97] and PSWEEP [ZDR99] handle distributed ISs. However, SWEEP and PSWEEP represent a significant improvement over Strobe in terms of performance (the number of messages and data sizes transferred between the DW and the ISs). PSWEEP is essentially a parallelized extension of SWEEP and thus offers several orders of magnitude in a performance increase over SWEEP given a system setup supporting multiple threads.

Furthermore, the Strobe [ZGMW96] family of algorithms is subject to the potential threat of infinite waiting, i.e., the DW extent may never get updated. The SWEEP [AAS97] and PSWEEP [ZDR99] family of algorithms eliminate this limitation by applying local compensation techniques. This avoids the need for a quiescent state of the environment before being able to update the DW. But like many other DW maintenance algorithms in the literature [ZGMHW95] both SWEEP and PSWEEP assume that each information source only contains one single relation. This is unrealistic in practice as most real data sources often contain 10s, 100s or more relations. It is thus important to be able to support multiple relations for each information source.

1.1 Motivation

Originally, SWEEP and PSWEEP have assumed that there is only one relation per IS. Below we now demonstrate that the local correction techniques they used will not work in the case of multiple relations per IS.

Example 1 *Given two information sources IS_1 and IS_2. R_1 and R_2 are two relations of IS_1 and R_3 is a relation of IS_2. Assume that we have a data warehouse defined by $V = R_1 \bowtie R_2 \bowtie R_3$. Now, there is a data update ΔR_3 of R_3 in IS_2. To calculate the incremental change of this update on the DW extent, denoted by ΔV, the SWEEP mediator would send down the following maintenance query to IS_1:*

$MQ = \Delta R_3 \bowtie (R_1 \bowtie R_2)$.

If a data update ΔR_2 occurs however while the MQ query is send down, then the wrapper would send back the following (incorrect) query result to the mediator:

$MQR = \Delta R_3 \bowtie (R_1 \bowtie (R2 + \Delta R_2))$.

This is obviously incorrect because of the effect of the concurrent data update ΔR_2. The SWEEP mediator handles this case by local compensation. This is the key feature of SWEEP that guarantees that SWEEP can successfully avoid an infinite wait, because any remote query send by SWEEP would raise the possibility of further concurrent data updates (DUs) requiring compensation. But here $\Delta R_3 \bowtie (R_1 \bowtie \Delta R_2)$ cannot be locally compensated because we need information about the extent of R_1 for this purpose. However, R_1 is not locally available in the DW, and hence to get the correct result, the mediator needs to generate a query and send it down to IS_1. In short, the local compensation, the main feature of SWEEP, is broken.

■

To handle this problem, one alternative intuitive solution to this problem may be to model each relation R_j of an IS_i as a separate information source IS_{ij}.

Example 2 *Using the scenario as in Example 1, we now treat each relation as a separate IS. Assuming there is a data update ΔR_3 of R_3 in IS_2. To calculate the incremental change ΔV, the SWEEP mediator would sequentially need to first send down a MQ to IS_{11} and then one to IS_{12}.*

The MQ sent to IS_{11} is defined by: $MQ_1 = \Delta R_3 \bowtie R_1$.

When the query result MQR_1 comes back from IS_{11}, then the mediator tests if there is any concurrent data update and then sends another query down to IS_{12}, which actually at same IS_1:

$MQ_2 = MQR_1 \bowtie R_2$.

If a data update ΔR_2 occurs before the MQ query is being processed in IS_{12}, the query result MQR_2 returned would be:

$MQR_2 = MQR_1 \bowtie (R_2 + \Delta R_2)$

The correct result however should have been: $MQR = MQR \bowtie R_2$.

SWEEP detects the concurrent data update ΔR_2 and applies local compensation to remove $MQR_1 \bowtie \Delta R_2$ to get the correct ΔV.

■

From Example 2, we notice that SWEEP can indeed support multi-relation ISs by treating each relation as a separate IS. However, this solution suffers from numerous shortcomings. First, the mediator would have to send down separate MQs to the same IS multiple times (one for each relation in it). In Example 2, the mediator sends MQ queries down to IS_1 twice to calculate ΔV, since IS_1 holds 2 relations. If there were 10 relations for each IS, then to handle one update, the DW has to send 10 sub-queries to and receives 10 query result messages from the IS as compared to one message exchange only.

In summary, the main drawbacks of this solution are:

- **Network Overhead:** The IS is connected to the DW by a network. To maintain the materialized view, the DW has to send queries down independently to each of the relations. This is big burden on the network.

- **IS Overhead:** Each individual IS needs to receive, handle and process n times (assume n represents the number of relations utilized in this IS) different queries instead of just one single query. This places a burden on the IS, potentially affecting not only the handling of this one DU but also the response time of other users of this IS.

- **DW Overhead:** The DW has more work in terms of having to send queries and collect answers from the independent relations. Together with the added delay from IS query processing and network delays, this would delay the update of the view extent of the data warehouse as it will take longer to get decisions made. So the DW has longer periods of being out-of-date.

1.2 Our Approach

To overcome the limitation of the strawman solution described in the Section 1.1 we instead propose a more efficient solution, namely a Multi-Relation Encapsulation Wrapper that treats one IS composed of multiple relations as if it were a single relation from the DW point of view. Like any traditional wrapper, our wrapper accepts queries from data warehouse, sends the query down to the IS, gets query results from the IS and returns them back to the data warehouse. It will also send data update messages to the data warehouse. The basic idea here is to treat an IS composed of multiple relations as one local view so that the DW will be aware of this one local view relation instead of the relations for each IS. Hence, existing algorithms for DW maintenance from the literature would function unchanged within this environment once enhanced by our Multi-Relation Encapsulation Wrapper.

This implies that the wrapper will need to receive queries from the DW expressed against one relation, namely, the view relation modeling the content of the IS, and then translate this query down into one processable by the actual IS. Similarly, the wrapper will translate on update message for one relation into an update message with respect to the view relation of the IS. In order to calculate the effect of one data update on the whole IS without the threat of an infinite wait, the wrapper needs to adopt a local compensation strategy.

The Multi-Relation Encapsulation Wrapper we propose has the following capabilities:

- Supports multiple relations in one IS.

- Encapsulates the local IS specific update detection and query processing and makes them transparent to the DW system, and it can easily be plugged into practice all current DW environments.

- Handles the local concurrent data updates and thus portrays correct update messages to the upper layer.

A DW framework that uses our Multi-Relation Encapsulation Wrapper will have the following benefits:

- Keeps all the benefits of the previous algorithms, because the DW layer can be used directly from the literature without any change (e.g., local to global name mapping). For example, using the PSWEEP's DW mediator, we can keep the performance benefits gained by PSWEEP [ZDR99].
- Handles concurrent DUs happening at the IS locally by using a local correction (LC) technique in the wrapper. Hence, there is no potential infinite waiting of processing the queries and propagating the data updates in concurrent data updates environment.

As we will demonstrate in this paper, our Multi-Relation Encapsulation Wrapper meets the following goals:

- **non-intrusive**: It does not require any modification to the existing processes and algorithms in a DW system. In particular, we do not want to impose additional modifications on the processing that the DW mediator, e.g., the SWEEP or PSWEEP algorithm, must perform. That is, the interface of the DW layer with the IS layer should remain unchanged, and the fact that the view relation models many actual IS relations should be transparent.
- **efficient**: It maintains all benefits of the previous view maintenance solutions proposed in the literature, while in addition offering improved performance to the overall process. Unlike the candidate solution described in Section 1.1, it should preserve the property of [AAS97] to do local and not remote compensation, as well as the property of PSWEEP [ZDR99] to allow for parallel view maintenance at the DW layer by instantiating multiple view maintenance threads.
- **flexible**: It has limited requirements upon the underlying environment, in particular, the information sources still should be allowed to be semi-autonomous. This means that ISs do not need to assist us with the DW maintenance process beyond reporting individual data updates or processing queries send down to them by the wrapper.
- **well-behaved**: It passes up view maintenance query results that properly incorporated the effects of all local concurrent data updates that take place while determine the query result. This MQR compensation should not require any infinite wait.

Outline: In Section 2, the DW model with the Multi-Relation Encapsulation Wrapper to support multiple relations per IS is given. Section 3 analysis the Multi-Relation Encapsulation Wrapper. In Section 4, the Multi-Relation Encapsulation Wrapper architecture and algorithm is presented. Section 5 describes the related work of the Multi-Relation Encapsulation Wrapper. Conclusions are discussed in Section 6.

2 The Data Warehouse Model Augmented with the Multi-Relation Encapsulation Wrapper

For this work, we assume a standard three-tier DW architecture as depicted in Figure 1. In general, the environment is divided into three layers, the *data warehouse layer*, the *mediator layer* and the *information* *source layer with wrappers.* The three layers are respectively connected by a FIFO network.

Assumption 1 *The order in which the mediator layer receives messages from an IS is the same as the order in which each IS sends out the messages, i.e., there is a FIFO network connecting them.*

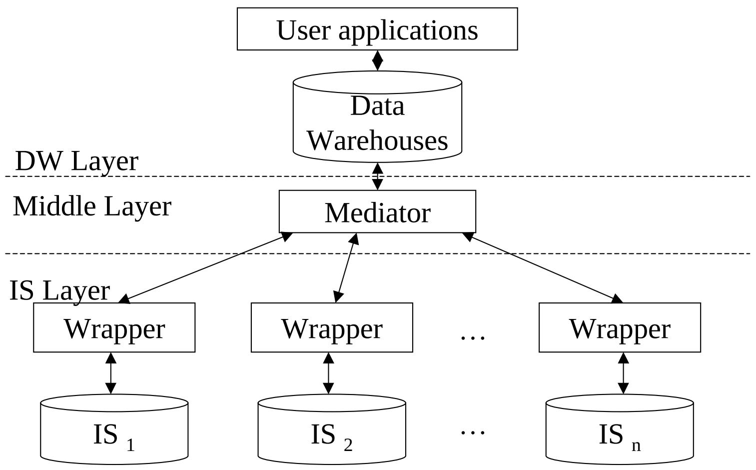

Figure 1: Data Warehousing Architecture

At the DW layer, the DW is materialized and directly responds to query requests by the users. At the middle layer, the mediator integrates the changes into the DW by merging the updates of the ISs with the data already present in the DW and resolving possible update anomalies. At the IS layer, the wrapper detects changes at its designated IS and propagates the changes in the generic form to the upper layer, i.e., the mediator.

Table 1 shows the notation we use in this paper.

There is a lot of work in the literature on the DW layer of concurrency control [KM99] and middle layer of view maintenance [ZGMHW95, ZGMW96, AAS97, ZDR99]. In this paper, we fully focus on the design of the wrappers for such concurrent environment.

2.1 Requirements of the Mediator

The mediator integrates the changes from the ISs into the DW by merging the updates with the data already present in the DW and resolving possible update anomalies. For every view located in the data warehouse, there is one mediator. The mediator is responsible for collecting messages from the corresponding wrappers at the ISs and maintaining the materialized view stored in the data warehouse.

Our claim is now that the utilization of the proposed Multi-Relation Encapsulation Wrapper will allow us to plug in any of the existing incremental view maintenance algorithms without requiring any major change to the mediator. The only change we need is to add (or change) the module for initialization of the

Notation	Meaning
MQ	Maintenance query from mediator to wrapper.
LQ	A local query to generate ΔIS from ΔR
LQR	The query result of LQ
LMQ	The localized Maintenance query.
LMQR	The query result of LMQ. Same as the MQR.
MQR	The query result of MQ.
V_IS_i	An local view definition stored in an IS_i wrapper. Same as LQ.
AQ	Assemble query used by VM in DW.
WMQ	Wrapper Message Queue.
OMQ	Order-Assignment Message Queue.
ΔR_i	A data updates from relation R_i.
ΔIS	Effect of a DU of a relation to the whole IS.
$\Delta IS(R_i)$	ΔIS generated from data update R_i.
ΔV_IS_i	ΔIS from IS_i.

Table 1: Notation and Meaning

system, in particular to help to properly utilize the view query needed by the DW from the Multi-Relation Encapsulation Wrapper.

The general requirements of the mediator that can cooperate with the Multi-Relation Encapsulation Wrapper are described as below:

1. The *mediator* is responsible for only one view definition in the data warehouse.
2. The *mediator* can maintain the data warehouse under the distributed environment where every IS at least has one relation.
3. The *mediator* handles the concurrent DUs for the view maintenance.

The initialization phase of the DW framework should decompose the view definition (local view) for every IS wrapper and generate a assembly query locally to maintain the materialized view. Then it will use the local view to initialize the corresponding *wrapper* for each IS.

Example 3 *As an example of a typical mediator architecture, let's describe the mediator of SWEEP [AAS97]. The mediator is only responsible for maintaining one view. For each data update reported by an IS wrapper, the mediator will send incremental maintenance queries (MQ) down to the individual ISs to calculate the incremental view changes of the data update on the DW extent (ΔV). Once it receives query results MQR from all relevant ISs, it integrates them into the data warehouse as shown in Figure 2.*

The mediator is composed of the Assign Time Stamp *process, the* Update Message Queue *process, the* View Maintenance *process, and the* Query Engine *process. The* View Maintenance *module generates incremental view maintenance queries (MQ) for a specific ΔIS to maintain the data warehouse. The* Query Engine *process sends the maintenance queries (MQ) down to the corresponding ISs and collects answers (MQRs)*

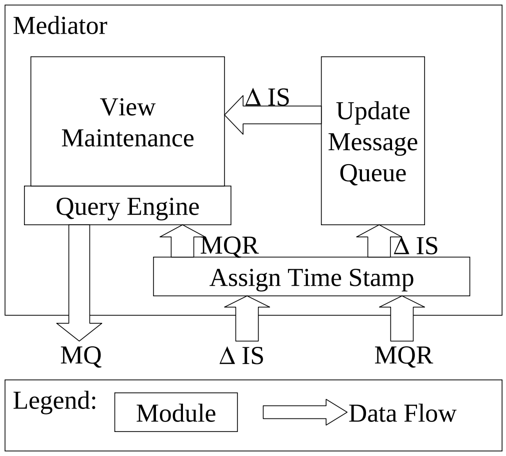

Figure 2: Mediator of PSWEEP

from all ISs and collects them into one ΔV. The Assign Time Stamp *process will assign a local time stamp to all incoming messages, i.e. ΔIS* [1] *and return query result MQR. The time stamps will be used by the* Query Engine *process to detect and compensate the concurrent DUs while processing the maintenance queries. The* Update Message Queue *process is used to buffer all the incoming data updates (ΔIS). The Assign Time Stamp process, Update Message Queue process, Query Engine process and View Maintenance process together can handle the concurrent DUs while maintaining the data warehouse.*

■

2.2 Assumptions for the Multi-Relation Encapsulation Wrapper

In order to make the Multi-Relation Encapsulation Wrapper work smoothly, we place the following assumptions on the DW environment.

Assumption 2 *The communication between the wrapper and the DBMS of that IS where the wrapper is located is FIFO.*

Assumption 2 can easily hold because the wrapper is built upon a centralized local IS. Furthermore, to erase any effect of interdependence among the ISs and focus on the wrapper of the individual IS, we have Assumption 3

Assumption 3 *All ISs are independent from each other, in the sense that a data update at one IS will not propagate into other ISs.*

[1] Here ΔIS is the same as ΔR because of the assumption that each IS contains only one relation.

All the relations, which are involved in the local view of the Multi-Relation Encapsulation Wrapper, generally have some kind of relationships with one another, i.e., relations should be connected by a join condition in the local view, otherwise the ΔIS might be large for a specific ΔR.

3 Analysis of Requirements of the Multi-Relation Encapsulation Wrapper

The primary functionalities of the Multi-Relation Encapsulation Wrapper are the same as those of any other basic wrapper to provide a unified interface to report data update messages and to process queries from the mediator.

Beyond the basic requirements, The Multi-Relation Encapsulation Wrapper offers the additional functionality of supporting multiple relations and handling concurrent local DUs. To support multiple relations in one information source, the Multi-Relation Encapsulation Wrapper stores one local view definition for each view of the DW. This view definition generated at the system initialization time will be used to calculates the ΔIS for each ΔR. The Multi-Relation Encapsulation Wrapper does not actually materialize the local view, instead it directly calculated the ΔIS by doing joins of the ΔR and the underlying relations of that IS specified by the local view.

3.1 Black-box Analysis of the Multi-Relation Encapsulation Wrapper

If we treat the Multi-Relation Encapsulation Wrapper as a black-box, we can identify the following inputs, outputs, and function requirements.

Input:	The wrapper receives maintenance queries (MQs) from the mediator, query results (LMQRs and LQRs) from the its designated IS, and ΔR from the IS.
Output:	The wrapper sends ΔISs and MQRs to the mediator, forwards LMQ queries to the ISs used to process the MQ, sends LQ queries down to the IS relations to calculate the $\Delta IS(R_i)$ from ΔR_i.
Function:	(1) The wrapper generates ΔIS for each ΔR_i with R_i being a relation in IS. (2) The wrapper processes the MQ and returns the MQR. (3) The wrapper ensures the correct order of returning messages. If the ΔIS is sent to DW by the wrapper before the MQR, then the MQR is generated to include the effects of the ΔR. Vice versa, if the MQR is returned before $\Delta IS(R)$, then the MQR will not include the effect of ΔR.

3.2 White-box Analysis of the Multi-Relation Encapsulation Wrapper

Every information source will have a wrapper for every mediator of the data warehouse DW that integrates data from two or more relations from this information source. In order to ensure the functionalities described in Section 3.1, we have following implementation requirements:

1. **Local View Definition**:

   The calculation of ΔIS out of each ΔR is based on the local view definition established for the materialized view defined in the DW.

2. **Local Correction**:

   The calculation of ΔISs for any ΔR will be corrected locally in the wrapper by some local compensation (LC) technique in the concurrent data updates environment. For each ΔR, there is one ΔIS generated and sent to the mediator.

3. **Single Transaction to Calculate the MQR**

   The MQ is executed by the wrapper and the MQR is returned to the mediator. The MQR contains the effect of all concurrent ΔRs that happened at the IS during the execution.

4. **Order Reassignment to Ensure Correctness**

   ΔIS and the MQR will be sent in such an order that the later one will have the effects of the previous one incorporated.

The data update calculation has to be done in the sequential way and apply the local compensation techniques on it, otherwise IS cannot report the ΔIS for a specific ΔR because of other continuous happened concurrent ΔRs. Hence the IS wrapper can not process other queries due to waiting for this calculation of the ΔIS. Then the whole maintenance process in the mediator will be blocked and waiting for that specific IS. Therefor, the wrappers have to use the local correction techniques to calculate the ΔIS.

Theorem 1 *ΔV calculated from ΔIS is identical to ΔV calculated directly from ΔR.*

3.3 System Initialization

To use the Multi-Relation Encapsulation Wrapper in the DW system, the following operations are required during system initialization.

1. The user view definition at the DW will be decomposed into local view queries for each of the involved ISs.

2. There is one assembly view based on the local views, which is stored in the mediator.

3. The DW system initializes the wrappers of informations sources by their respective local views.

Example 4 *This initialization process is described using the same scenario as Example 1 in Section 1. Assume that we have a data warehouse built upon two information sources IS_1 and IS_2. Table 2 shows the schema of the relations in each IS. Assume in the data warehouse, we have a view as defined in Figure 3* [2]*.*

IS name	Relation Name	Attribute Name
IS_1	R_1	(A, C)
	R_2	(D, E)
IS_2	R_3	(B, F)

Table 2: Relation Structure

DW View Q1:

CREATE VIEW	V AS
SELECT	A, B
FROM	$IS_1.R_1$, $IS_1.R_2$, $IS_2.R_3$
WHERE	$IS_1.R_1$.C = $IS_1.R_2$.D AND
	$IS_1.R_2$.E = $IS_2.R_3$.F

Figure 3: Data Warehouse View Definition

During the initialization phase of the mediator, query Q1 will be decomposed into query Q2 for IS_1 and query Q3 for IS_2 to create the local views. As we can see from Figure 4, the local views only contain a subset of information of one information source as required by the DW view.

Local View Q2 of IS_1:

CREATE VIEW	V_IS_1 AS
SELECT	A, E
FROM	R_1, R_2
WHERE	R_1.C = R_2.D

Local View Q3 of IS_2:

CREATE VIEW	V_IS_2 AS
SELECT	B, F
FROM	R_3

Figure 4: Local Views at the Information Sources

The assembly query $Q4$ depicted in Figure 5 uses sub-queries $Q2$ and $Q3$ to initialize the mediator. The mediator is based on the assembly query $Q4$ defined in Figure 5, which is used to maintain the DW instead of the initial user provided query $Q1$.

■

4 Design of the Multi-Relation Encapsulation Wrapper Module

4.1 Architecture of the Multi-Relation Encapsulation Wrapper

Figure 6 shows the Multi-Relation Encapsulation Wrapper architecture. The data structures in the wrapper include a *Wrapper Message Queue* and an *Order Message Queue*. The *Wrapper Message Queue* is used to buffer all the incoming data updates (ΔR) from relations and MQ from the mediator. The *Order Message*

[2] The view V is defined in a extended SQL format that can specify the information sources.

DW Assembled Query Q4:

CREATE VIEW	V AS
SELECT	A, B
FROM	V_IS_1, V_IS_2
WHERE	V_IS_1.E = V_IS_2.F

Figure 5: Assembly Query

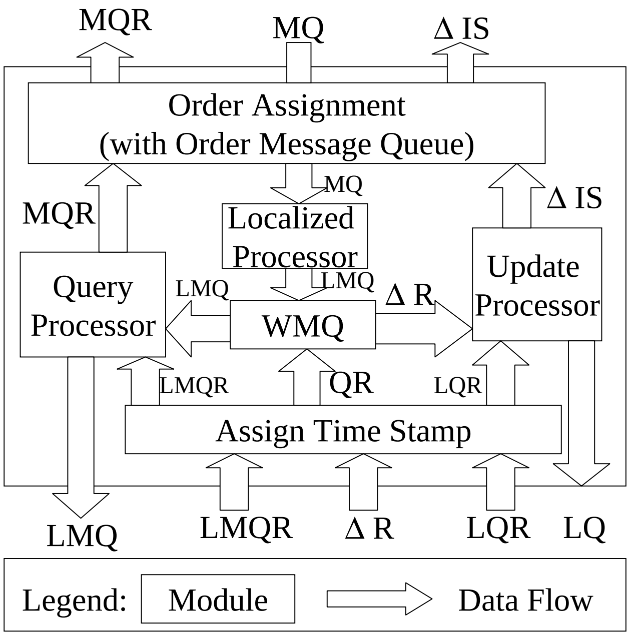

Figure 6: Architecture of the Multi-Relation Encapsulation Wrapper

Queue is used to buffer and reorder the output messages (MQR and ΔIS) in order to ensure the messages will be send to the mediator in the correct order.

The Multi-Relation Encapsulation Wrapper is composed of the *Assign Time Stamp* process, the *Localized Processor* process, the *Query Processor* process, the *Update Processor* process, and the *Order Assignment*.

The **Localized Processor** accepts MQ from the mediator. It localizes MQ in the sense of translate MQ into a query LMQ understood by the IS.

The **Assign Time Stamp** process will assign a local time stamp to all incoming messages, i.e. ΔR and LMQR from the underlying relations. The time stamps will be used by the *Query Processor* to ensure the sequential handling of incoming messages.

The **Update Processor** calculates ΔIS for ΔR. For this purpose, it first generates local queries (LQ) based on the local view definition stored in the wrapper at system initialization for a specific ΔR to produce ΔIS. Then it sequentially processes the LQ to calculate ΔIS and fix any concurrency update problem by using a local compensation technique.

The **Query Processor** handles messages in the *Wrapper Message Queue* sequentially. There are two kinds of messages in WMQ: LMQ and ΔR. When the *Query Processor* gets a LMQ from the WMQ, it will send the LMQ down to the IS relations and get the LMQR back within one transaction. The LMQR is forwarded to the *Order Assignment Processor*. When the *Query Processor* gets ΔR from the WMQ, it will forward it to the *Update Process*.

The **Order Assignment** processor reorders the message (e.g., ΔIS and MQR) sequence and sends them to the mediator. It sends MQR and ΔIS back to the mediator according to their time stamp to ensure the correct sending sequence.

Theorem 2 *As long as the wrapper sends ΔIS and MQR in the order that the later one will have the effect of all the previous data updates, the mediator can correctly maintain the DW.*

4.2 Calculating ΔIS under Concurrent DUs

As we stated before, the Multi-Relation Encapsulation Wrapper reports ΔIS for every ΔR. Because we do not actually materialize local views in the wrapper, the wrapper needs to send down a query LQ to the IS relations to calculate ΔIS for each ΔR and the $\Delta IS(R)$ should only have an effect of ΔR.

Theorem 3 *There will be no potential threat of an infinite waiting when the wrapper calculate ΔIS for a ΔR by using a local compensation technique to handle concurrent DUs.*

The example below shows how the Multi-Relation Encapsulation Wrapper correctly calculates ΔIS from ΔR under concurrent data update environment by using a local compensation technique.

Example 5 *IF an IS has two relations R_1 and R_2. At first, ΔR_1 arrives at the wrapper from R_1 at time t1. The wrapper will calculate $\Delta IS(R_1)$ and report it to the mediator. To calculate $\Delta IS(R_1)$, the wrapper sends down a query LQ to R_2.*

$LQ = \Delta R_1 \bowtie R_2$

If a concurrent data update ΔR_2 happens when the $\Delta IS(R_1)$ is being calculated, the wrapper receives ΔR_2 from R_2 with time stamp t2 and gets the query result LQR back with time stamp t3.

$LQR = \Delta R_1 \bowtie (R_2 + \Delta R_2) = (\Delta R_1 \bowtie R_2) + (\Delta R_1 \bowtie \Delta R_2)$

The query result LQR has an effect of ΔR_2, which is incorrect. When the wrapper gets the LQR back, it will check their time stamp. From $t2 \leq t3$, the wrapper detects that a concurrent data update had occured. The wrapper knows the need to eliminate ΔR_2 from $\Delta IS(R_1)$. Although the wrapper does not store any view, it has the information of ΔR_1 and ΔR_2. It is easy to get the correct result by using a local compensation technique as for example defined in [AAS97].

$LQR = LQR - \Delta R_1 \bowtie \Delta R_2$

So, the correct $\Delta IS(R_1)$ is sent to the mediator by the wrapper.

■

4.3 Algorithm of the Multi-Relation Encapsulation Wrapper

Based on the previous description of the key features of the Multi-Relation Encapsulation Wrapper, we now can give pseudo code of the Multi-Relation Encapsulation Wrapper module. Figure 7 depicts the software module that is employed at the Multi-Relation Encapsulation Wrapper for multi-relation simulation.

The *Query Processor* is used to process the query LMQ. Because the wrapper is only for one IS, the relations in the same IS are local and centralized. The LMQ can be sent to the DBMS of the IS and the DBMS processes LMQ and sends the query result LMQR back to wrapper in one transaction.

The *Update Processor* is invoked for every update received at the wrapper to generate ΔIS. It sequentially calculate the ΔIS and erases any abnormal behavior by local correction techniques [AAS97].

Theorem 4 *Though the calculation time of independent ΔISs may be different for the different IS wrappers and the order in which they are received by DW may be not the same as the order in which the ΔR actually happened in real time, this receive order will not affect the final correctness of the data warehouse after updating.*

```
MODULE Multi-Relation Encapsulation Wrapper;
  CONSTANT
  GLOBAL DATA
    V: RELATION; /* Initialized to the correct view */
    WrapperMessageQueue: QUEUE initially 0;
    OrderMessageQueue: QUEUE initially 0;

  PROCESS UpdateProcessor(ΔR: Relation; UpdateSource:
    INTEGER; TimeStamp: INTEGER): RELATION
  VAR
    ΔIS, TempIS: RELATION;
    j: INTEGER;
  BEGIN
    ΔIS = ΔR;
    /* Compute the left part of the ΔIS from ΔR */
    FOR (j = UpdateSource -1; j ≥ 1; j–) DO
      TempIS = ΔIS;
      SEND ΔIS to Source Relation i;
      RECEIVE ΔIS FROM Sour Relation i;
      IF ∃(ΔR, j, t) ∈ WrapperMessageQueue
        Then ΔIS = ΔIS - ΔR_j ⋈ TempIS;
      ENDIF
    ENDFOR;
    /* Compute the right part to the ΔIS from ΔR */
    FOR (j=UpdateSource+1; j ≤ n; j++) DO
      TempIS = ΔIS;
      SEND ΔIS to Source Relation i;
      RECEIVE ΔIS FROM Sour Relation i;
      IF ∃(ΔR, j, t) ∈ WrapperMessageQueue
        Then ΔIS = ΔIS - ΔR_j ⋈ TempIS;
      ENDIF
    ENDFOR;
    RETURN ΔIS;
    ENDAREA
  END UpdateProcessor;

  PROCESS AssignTimeStamp;
  VAR
    t: TIME; /* current system time at the IS */
  BEGIN
    LOOP
      RECEIVE Message FROM Relation i and LMQ FROM Local-
izeProcess() as received order;
        t= getCurrentTime();
        APPEND (Message,i, t) TO WrapperMessageQueue;
    FOREVER;
  END AssignTimeStamp;

  PROCESS QueryProcessor;
  BEGIN
    WHILE WMQ not empty
      REMOVE a Message FROM WrapperMessageQueue;
      IF the Message is LMQ THEN
        SEND LMQ to Relations /*to calculated ΔV (LMQR)*/
        RECEIVE LMQR FROM Relations
        t= getCurrentTime();
        OrderAssignment(LMQR, t);
      ELSE /*Message is ΔR need calculate ΔIS */
        UpdateProcessor(Message.ΔR, i,Message.t)
        OrderAssignment(ΔIS);
      ENDIF
    ENDWHILE
  END QueryProcessor;

  PROCESS OrderAssignment(QueryResult,t);
  VAR
    r: INTEGER;
  BEGIN
    IF the QueryResult is LMQR THEN
      r = 0;
      WHILE WMQ is not empty
        IF ∃(ΔR, t') AND (t' ≤ t)
          r=1;
          IF (ΔR,t') is not in OMQ /*Order Message Queue*/
            APPEND (ΔR,t') TO OMQ;
          ENDIF
        ELSE
          BREAK;
        ENDIF
      END WHILE
      IF (r = 0)
        SEND LMQR TO Mediator;
      ELSE
        APPEND (LMQR,t) TO OMQ;
    ELSE /*Query Result is ΔIS*/
      SEND ΔIS(R) TO Mediator;
      IF ΔR is in OMQ
        DELETE ΔR FROM OMQ;
        WHILE head of OMQ is LMQR
          DELETE LMQR FROM OMQ;
          SEND LMQR TO Mediator;
        END WHILE
    ENDIF
  END OrderAssignment;

BEGIN /* Start Multi-Relation Encapsulation Wrapper Processes */
  StartProcess(AssignTimeStamp);
  StartProcess(QueryProcessor);
END Multi-Relation Encapsulation Wrapper Process
```

Figure 7: Pseudo Code of the Multi-Relation Encapsulation Wrapper Module

4.4 Multi-Relation Simulation Example in the Multi-Relation Encapsulation Wrapper

Here is an example of how the data updates of the underlying information sources are handled by the Multi-Relation Encapsulation Wrapper. The system has been initialized as described in Section 3.3.

Example 6 *Assume there is a data update ΔR_3 on relation R_3 of IS_2. The* Update Processor *of the wrapper of IS_2 will generate the query LQ based on $Q3$ to calculate $\Delta V\_IS_2(R_3)$. Then the wrapper reports the effect of ΔR_3, that is $\Delta V\_IS_2(R_3)$ to the data warehouse.*

After receiving $\Delta V\_IS_2(R_3)$ from IS_2, the mediator will generate maintenance query $Q5$ defined in Figure 8 with $\Delta V\_IS_2(R_3)$ and send the MQ to IS_1 to calculate ΔV.

MQ Q5:
SELECT A, B
FROM V_IS_1, ΔV_IS_2
WHERE V_IS_1.E = ΔV_IS_2.F

Figure 8: Maintenance Query (MQ) Send to IS_1

After the wrapper of IS_1 receives MQ $Q5$ from the mediator, the Localized Processor merges the MQ query $Q5$ and the local view $Q2$ to generate LMQ (Figure 9). The Assign Time Stamp *of the wrapper of IS_1 assigns a time stamp to this LMQ query and buffers in the WMQ. The* Query Process *gets LMQ from the WMQ. To execute the query LMQ $Q6$, the $QueryProcess$ sends LMQ $Q6$ to base data base relations of IS_1 and gets the query result LMQR back* in one transaction.

Assuming there is concurrent data update ΔR_2 when LMQ $Q6$ is being executed, the wrapper will assign the local time to ΔR_2 and the Update Processor *will generate $\Delta IS_1(R_2)$. The query result LMQR of LMQ $Q6$ will have the effect of ΔR_2. When* Order Assignment *receives the LMQR $Q6$, it knows that there is a concurrent data update ΔR_2 by checking the local time stamp scheme. So it buffered the LMQR in the QM until $\Delta IS_1(R_2)$ is also received by the* Order Assignment*. Then it returns $\Delta IS_1(R_2)$ followed by the query result LMQR that is to ensure the later one has the effect of the previous data updates incorporated.*

LMQ Q6:
SELECT A, B
FROM (SELECT A, E FROM R_1, R_2 WHERE R_1.C = R_2.D) AS V_IS_1,ΔV_IS_2
WHERE V_IS_1.E = ΔV_IS_2.F

Figure 9: Localized Maintenance Query (LMQ) in IS_1

The mediator will receive a $\Delta IS_1(R_2)$ followed by the query result MQR (Q5) from IS_1. ΔV is calculated. Then mediator knows that the MQR has an effect of $\Delta IS_1(R_2)$ by this receive order and $\Delta IS_1(R_2)$ is a *concurrent data updates. Hence the local compensation will erase the effect of the concurrent data update of $\Delta IS_1(R_2)$ and then the data warehouse is correctly updated.*

■

5 Related Work

View maintenance methods concentrate on incrementally maintaining the extent of the DW. Zhuge et al. [ZGMHW95, ZWGM97] introduce the ECA algorithm for incremental view maintenance under concurrent IS data updates restricted to one centralized IS. Hence there is no distinction between the relations and IS. In Strobe [ZGMW96], they extend their approach to handle multiple ISs. Agrawal et al. [AAS97] propose the SWEEP-algorithm that can ensure the consistency of the data warehouse in a larger number of data updates compared to the Strobe family of algorithms. Though their work has reduced the total number of remote join queries, their work is based on the single relation ISs. In a separate work, we have proposed the PSWEEP algorithm [ZDR99] that improves the performance of SWEEP by parallelizing the view maintenance processes of SWEEP. However, PSWEEP is also assumes the limitation of only one relation per IS. In this paper, we are instead considering how to apply those algorithms into a more realistic environment where there are more than one relation at one IS by providing an Multi-Relation Encapsulation Wrapper. Thus our work is complimentary to previous mentioned view maintenance algorithms.

Wrapper's main tasks are to map between data models, e.g., local to global model, or web to relational model. Wrappers can be used to present a simplified interface, to encapsulate diverse sources so that they all present a common interface, to add functionality to the data source, or to expose some of the data source's internal interfaces. There are a lot of recent work on wrapper toolkits. Our work instead focuses on reconstructing the source data and providing the data to the upper level.

6 Conclusions

The SWEEP [AAS97] and PSWEEP [ZDR99] algorithms have the least limitations compared to alternate incremental view maintenance algorithms in the literature [ZGMHW95, ZGMW96]. However they can only support one relation per information source, which is unrealistic in practice as most real data sources have several facts of information. In this work, we propose a Multi-Relation Encapsulation Wrapper to overcome this problem. The main features of the Multi-Relation Encapsulation Wrapper are: 1. Supports multiple relations per IS by treating one IS composed of multiple relations as if it were a single relation from the DW point of view. 2. Is transparent to the DW system and thus can easily work with many mediators while keeping all the benefits of other incremental view maintenance algorithms. 3. Generates ΔIS for each ΔR under concurrent data updates environment by using a local compensation technique. 4. Sends ΔIS and MQR to the mediator in an order that the later one has the effect of the previous data updates. IN summary, our solution of the Multi-Relation Encapsulation Wrapper maintains all the advantages offered by existing algorithms in the literature in particular SWEEP and PSWEEP, while also achieving the additional desired features of being non-intrusive, efficient and flexible. We are currently in the process of enhancing our distributed data warehouse system EVE by this wrapper techniques.

References

[AAS97] D. Agrawal, A. E. Abbadi, and A. Singh. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[CD97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[GM96] A. Gupta and I. S. Mumick. What is the Data Warehousing Problem? (Are Materialized Views the Answer?). In *International Conference on Very Large Data Bases*, page 602, 1996.

[GMLWZ98] H. Garcia-Molina, W. Labio, J. L. Wiener, and Y. Zhuge. Distributed and Parallel Computing Issues in Data Warehousing (Abstract). In *Symposium on Principles of Distributed Computing*, page 7, 1998.

[KLMR97] A. Kawaguchi, D. F. Lieuwen, I. S. Mumick, and K. A. Ross. Implementing Incremental View Maintenance in Nested Data Models. In *Workshop on Database Programming Languages*, pages 202–221, 1997.

[KM99] S. Kulkarni and M. Mohania. Concurrent Maintenance of Views Using Multiple Versions. 1999.

[MKK97] M. K. Mohania, S. Konomi, and Y. Kambayashi. Incremental Maintenance of Materialized Views. In *Database and Expert Systems Applications (DEXA)*, pages 551–560, 1997.

[WB97] M. Wu and A. P. Buchman. Research Issues in Data Warehousing. In *Datenbanksysteme in Bro, Technik und Wissenschaft*, pages 61–82, 1997.

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of International Conference on Information and Knowledge Management*, pages 25–30, 1995.

[ZDR99] X. Zhang, L. Ding, and E. A. Rundensteiner. PSWEEP: Parallel View Maintenance Under Concurrent Data Updates of Distributed Sources. Technical Report WPI-CS-TR-99-14, Worcester Polytechnic Institute, Dept. of Computer Science, May 1999.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.

[ZGMW96] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, December 1996.

[ZWGM97] Y. Zhuge, J. L. Wiener, and H. Garcia-Molina. Multiple View Consistency for Data Warehousing. In *Proceedings of IEEE International Conference on Data Engineering*, pages 289–300, 1997.