1991

Negotiating and Supporting Interdependent Data in Very Large Federated Database Systems

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

James G. Mullen

Report Number:
91-047

NEGOTIATING AND SUPPORTING INTERDEPENDENT DATA IN VERY LARGE FEDERATED DATABASE SYSTEMS

Ahmed K. Elmagarmid
James G. Mullen

CSD-TR-91-047
June 1991

Negotiating and Supporting Interdependent Data in Very Large Federated Database Systems*

Ahmed K. Elmagarmid and James G. Mullen

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

June 12, 1991

Abstract

This paper discusses the negotiation and support of interdependent data in federated database systems. We present extensions to SQL that are used for negotiating distributed interdependent data, and we use quasi-views (a concept similar to quasi-copies and distributed materialized views) to support the interdependent data that is negotiated. The emphasis of our work is on a method that scales well to federations that have many component systems and are very geographically distributed.

1 Introduction

Federated database systems are very loosely coupled distributed database systems that combine local database systems which have a very high degree of autonomy. Usually the systems in the federation are heterogeneous. Heterogeneity may occur in the system hardware, operating system, database system, query language, etc. Federated database systems occur naturally when the database systems of various organizations are integrated (e.g. a travel agent who accesses the databases of multiple car rental agencies, airlines, etc...), and in individual organizations as a result of intra-organizational autonomy, and the dynamic nature of organizations (e.g. corporate mergers) and database technology.

*This research funded by the Indiana Corporation for Science and Technology (CST), a PYI Award from NSF under grant IRI-8857952, grants from the AT&T Foundation, Tektronix, SERC, Mobil Oil and Bell Northern Research.

Interdependent data are data that are related to each other through an integrity constraint. The concept of interdependent data can be considered to be a generalization of replicated data. Interdependent data has been found to occur naturally in organization, and to be costly to maintain. For example, [SK89] discusses interdependent data that occurs in the systems of Bellcore and Bellcore Client Companies.

Although it is desirable to support interdependent data in federated database systems, the high degree of autonomy of the local systems makes this more difficult than in traditional tightly-coupled distributed database systems. For example, it is not possible to atomically commit transactions, in general, in a federated database system without violating the autonomy of the local systems [MEK91]. Also, it is often impractical from a performance perspective to implement immediate consistency in federated database systems, especially those that are very geographically distributed or use slow communication links. Fortunately, the high degree of consistency provided in traditional database systems is often not necessary in the federated database environment. It is often acceptable to have some controlled level of inconsistency.

In this paper we present a method for negotiating and supporting interdependent data in a federated database environment that allows the interdependent data to have weak consistency conditions. Our method uses quasi-views (a concept similar to quasi-copies and distributed materialized views) to support interdependent data. We propose extensions to the query language SQL and an algorithm that uses these extensions to negotiate interdependent data agreements. The emphasis of our work is on a method that is both scalable and practical.

The remainder of this paper is organized as follows. In section 2 we present background information. In section 3 we present our method for negotiating and supporting interdependent data. Finally, section 4 contains our conclusions.

2 Background

2.1 The Federated Database Model

We assume a federated database where not only is each database system that is a member of the federation autonomous, but also there is no central authority to control the interactions between the databases, such as a global transaction manager in many papers. Our model is similar to that used in [HM85] and [AB89]. And, our model would be classified as a *loosely coupled federated database system* in the multidatabase taxonomy presented in [SL90]. In our model, a data item may be replicated, however, one site is considered to "own" the

data. All updates to the data must be applied to the owner's copy of the data. In this paper, we do not consider the case where two or more different sites both own data that has the same semantic meaning and there is a desire to have some level of consistency between the different copies of data. The problem of keeping such replicas consistent is discussed in [SLE91]. See also [RS91] for a transaction-based approach that could be used to manage this kind of consistency.

Each database systems has the following information:

- **Federation Dictionary** - describes the federations that the database system manages.
- **Private Schema** - describes the information about the database system.
- **Export Schema** - describes what information the database system is willing to share with other systems.
- **Import Schema** - describes the information at other database systems for which access agreements have been arranged.

Our system architecture is shown in figure 1.

2.2 Interdependent Data

Interdependent data are data that are related to each other through an integrity constraint. Examples of interdependent data include replicated data, partially replicated data, and summary data (e.g. the sum or average of some data). Interdependent data may have varying degrees of consistency. For example, the interdependent data may be inconsistent by at most ten minutes, three versions, 10%, etc. In [SR90] various classifications, and issues related to interdependent data management are discussed. Four criteria are used to characterize interdependent data:

1. **Type of Interdatabase Dependency.**
2. **System Heterogeneity.**
3. **Degree of Local Autonomy.**
4. **Data Consistency Criteria.**

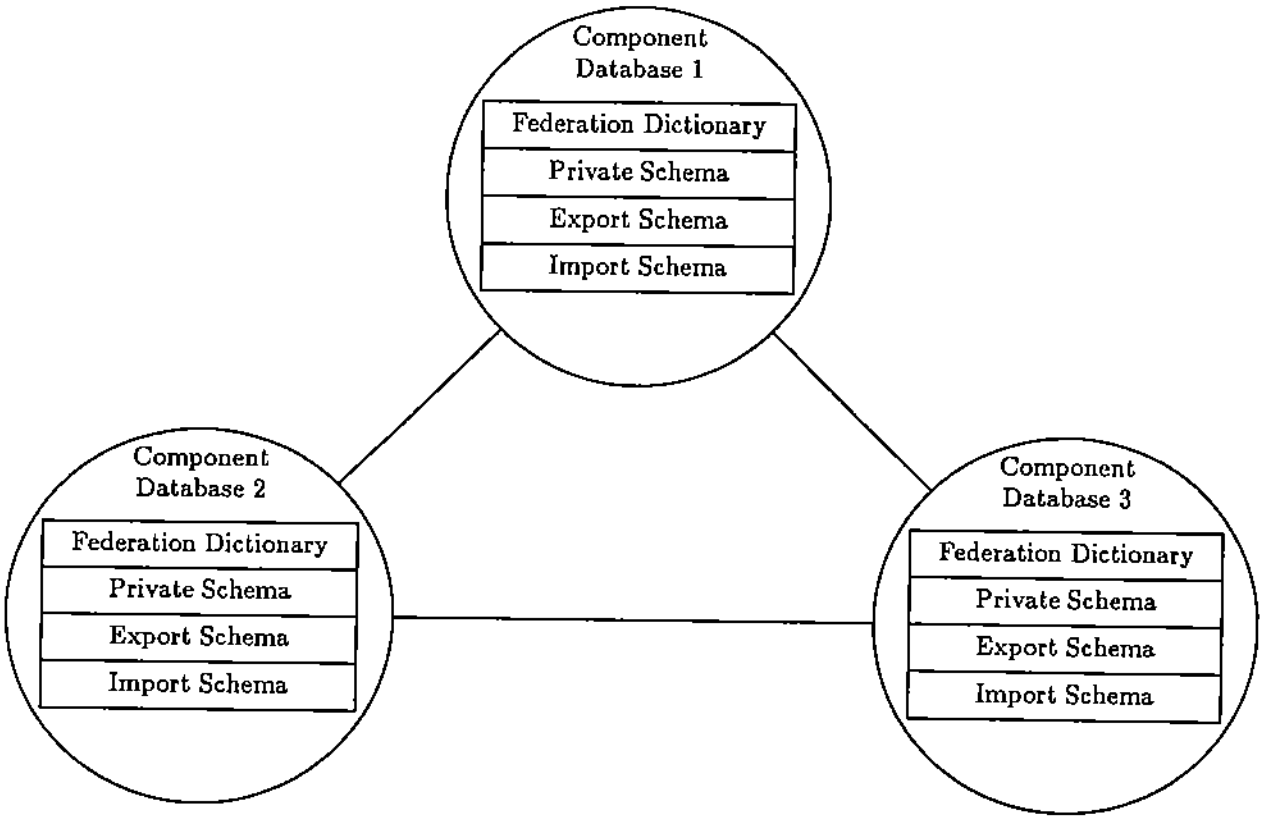

Figure 1: Federated Database Architecture.

3 Negotiating and Supporting Interdependent Data

3.1 Quasi-Views

In this paper we use a concept called *quasi-views* (a generalization of views) to implement interdependent data. We define quasi-views as follows:

> Views of data, that can be derived from other (possibly distributed) data, that may have weak (non-immediate) consistency.

Quasi-views allow transparent distributed data access. Allowing weak consistency levels allows one to sacrifice consistency for improved performance. Quasi-views are similar to both quasi-copies and distributed materialized views.

Quasi-views are similar to quasi-copies [ABGM87, ABGM90]. However, quasi-views may be summary information (for example, the sum or average of several numbers), as well as

full, partial, and overlapping replication of distributed data at multiple remote systems. It does not appear from [ABGM90] that quasi-copies support this level of flexibility.

Quasi-views are also similar to distributed materialized views (see, for example, [SP89]). However, quasi-views emphasize the concept provided, not the implementation of the concept. That is, a quasi-view is simply a view that may not have immediate consistency, it does not have to be materialized. Although in this paper our method implements weak consistency by using materialization, this would not have to be the case. For example, a view of a quasi-view may be considered to be a view that has weak consistency, but does not have a materialization. Another example would be a view that was constructed using a query that approximated the true value of the view. For example, a view that is supposed to be the average of a group of numbers that was constructed by a query that looked at only a sample of those numbers.

In this paper we classify quasi-views along two dimensions:

1. **Distribution Level.** We consider the following classifications listed in order of least general to most general.
   - **Local System.** Quasi-view can only be derived from data at the local system.
   - **Single Remote System.** Quasi-view may be derived from data at the local system, *or* data from a remote system, however the quasi-view must be derived from only a single system.
   - **Multiple Remote Systems.** Quasi-view may be derived from data at the local system and multiple remote systems.
2. **Allowable Consistency Levels.** We consider the following classifications listed in order of least general to most general.
   - **Immediate Consistency.** Interdependent data is always up to date.
   - **Importer-Verifiable Weak Consistency.** In addition to immediate consistency, weak consistency is also allowed, but only those criteria that may be checked solely by the importer of the data, without involving the owner, are allowed.
   - **Weak Consistency.** In addition to the two forms of consistency above that are allowed, general forms of weak consistency, including those that can only be checked by the owner(s) of the data from which the interdependent data was derived.

In this paper, we consider importer-verifiable consistency conditions since we are concerned with finding a method that scales well to large systems. Some importer-verifiable conditions

Distribution Level →

Allowable Consistency Levels ↓	Local System	Single Remote System	Multiple Remote Systems
Immediate Consistency			
Importer-Verifiable Weak Consistency			X
Weak Consistency			

Table 1: Quasi-views.

and their use in load balancing in an environment that supports quasi-copies are discussed in [ABGM90]. Specifically, we look at supporting distributed importer-verifiable consistency interdependent data. An "X" appears for this class in table 1. Note that this class is a generalization of the classes above and to the left of it in the table, since our classifications of distribution level and allowable consistency level are increasingly general.

General weak consistency interdependent data that are derived from data at multiple remote systems seems to be difficult to implement because it requires cooperation between multiple autonomous sites. For example, the sum of a distributed set of data which is to be updated whenever it is 10 minutes or older (an importer-verifiable condition), seems much easier to implement than the same dependency with the condition that the sum should be changed if the sum of the values (which reside at multiple remote autonomous systems) on which it depends changes by $\pm$ 5% (a non-importer-verifiable condition).

In this paper we will consider *importer-verifiable consistency conditions*. That is, consistency conditions which the importer can check on its own, without assistance from the exporter. The consistency conditions we support are:

1. Temporal. The importer can have the view refreshed every t time units, when the view is accessed and is more than t time units old, or at specific times.

2. Access Amounts. View is refreshed every x accesses.

The advantage of these consistency conditions is that they allow a large number of importers of a data item without requiring storage or processing for consistency checking by the owner (exporter) of the data.

It may also be possible to simulate exporter-verifiable consistency conditions. For example, one could simulate the condition of updating the quasi-view every time the data it is derived from changes by 5%, by keeping track of the difference in the value of the new and old copy of the quasi-view, and modifying the time between updates of the copy to support this condition. This method would have some limitations, and we do not discuss such conditions further in this paper.

3.2 SQL Language Extensions

We have created extensions to the SQL query language to support the negotiation of interdependent data in our database environment. A summary of the extensions we use is as follows:

1. Support for new QUASIVIEW construct. Users can create, delete, and query QUASIVIEWS.

2. USE clause added to query statements to allow the specification of access to multiple remote systems. Our USE clause is taken from MSQL described in [LAZ+87].

The syntax of the command used to create a quasi-view is similar to that of the command to create a view in SQL (see, for example, [Dat90]) and is shown below:

```
CREATE QUASIVIEW name [ ( column [, column] ... ) ]
   AS subquery
   [UPDATE consistency_expression [AND consistency_expression] ... ]
   [ON ERROR {ABORT | FLAG | PROCESS} ] ;

   consistency_expression ::=
          EVERY integer ACCESSES
        | EVERY [date_specification] time [, [date_specification] time ] ...
        | EVERY float time_units
        | AT date_specification [time] [, date_specification [time] ] ...
        | WHEN float time_units OLD

   date_specification ::= month/day/year | weekday

   weekday ::= SUNDAY | MONDAY | TUESDAY | WEDNESDAY | THURSDAY |
               FRIDAY | SATURDAY

   time ::= hh:mm

   time_units ::= SECONDS | MINUTES | HOURS | DAYS | WEEKS | MONTHS | YEARS
```

The AS clause is analogous to the standard view AS clause, and specifies the query that is used to construct the quasi-view. We allow the query to use the MSQL USE clause so that it can specify queries covering multiple remote databases. The syntax of the USE clause used in this paper is as follows:

```
USE dbname [ dbname ] ...
```

This clause specifies which databases the query may access. To distinguish relations with the same name in different databases, the database name is prepended, for example:

```
USE empdb1 empdb2
   (SELECT * FROM empdb1.EMPLOYEES)
   UNION
   (SELECT * FROM empdb2.EMPLOYEES)
```

The actual MSQL USE clause is more complex than the one presented here (see [LAZ+87] for a complete description).

The UPDATE clause is used to specify the consistency level of the quasi-view by indicating the condition for updating (or refreshing) the quasi-view. The default action is to provide immediate consistency (i.e. always update). See the next section for specific examples of its use.

The ON ERROR clause specifies what the system should do when a quasi-view is accessed, but the system cannot provide the consistency level specified when the quasi-view was created. The options allowed are as follows:

1. **ABORT.** Abort the transaction, and all subsequent transactions that access the quasi-view until the consistency level desired can be obtained. So, in effect, access to the data is blocked.

2. **FLAG.** Process the query, but also flag the query as not meeting the consistency condition, and also possibly indicate what consistency condition is actually provided (for example, the last time the view was updated).

3. **PROCESS.** Process the query with no indication that the results are not up to the desired level of consistency.

Deleting a quasi-view is analogous to deleting a view, and the following command would be used:

```
DROP QUASIVIEW name
```

Querying a quasi-view is done in the same way as with views, and relations. The SELECT command is used. Quasi-view modification would have the same problems as view modification (see, for example, [Dat90]), in addition to problems caused by dealing with autonomous distributed systems. We do not discuss updates, insertions, and deletions to quasi-views further in this paper.

3.3 Interdependent Data Examples

This section presents examples of interdependent data that can be negotiated and supported using our method.

Example of Traditional Replication. In this example we will show how our model would support traditional replication.

```
CREATE QUASIVIEW EMPLOYEES
   AS USE empdb
      SELECT *
         FROM EMPLOYEES
   ON ERROR ABORT;
```

In other words, any time that a query is performed on the quasi-view EMPLOYEES, the EMPLOYEES relation at remote system empdb will be queried. If the remote query fails, the transaction accessing the quasi-view will be aborted.

Examples of Weak Consistency Replication. In this example we show how weak consistency replication is supported. The following command would set up a quasi-view where the local copy is updated every 15 minutes (regardless of the user-generated queries on the quasi-view).

```
CREATE QUASIVIEW EMPLOYEES
   AS USE empdb
      SELECT *
         FROM EMPLOYEES
   UPDATE EVERY 15 MINUTES
   ON ERROR PROCESS;
```

The following command would set up a quasi-copy where any time that a query is issued on the quasi-view, the date of the quasi-view is checked to see if it is more than 15 minutes old. If not, the local copy (or materialization) is queried, otherwise the remote copy is queried.

```
CREATE QUASIVIEW EMPLOYEES
   AS USE empdb
      SELECT *
         FROM EMPLOYEES
   UPDATE WHEN 15 MINUTES OLD
   ON ERROR FLAG;
```

The consistency condition here has the advantage that no processing time is wasted when the quasi-view goes for long periods of time without being queried, and the possible disadvantage that every query that accesses the quasi-view after 15 minutes of non-use will necessarily have to query the remote data from which the view is constructed.

Example of Summary Interdependent Data. In the following example we will demonstrate how one would negotiate a quasi-view that represents the total number of employees in an organization that has three divisions. The employee relations for the divisions are stored at separate remote systems named empdb1, empdb2, and empdb3. We will assume that the system where the total is stored will only need to have updates on each Friday at 6:00pm. To set up the interdependent data, the following command could be used:

```
CREATE QUASIVIEW EMPLOYEE_COUNT ( COUNT )
   AS USE empdb1 empdb2 emdb3
      SELECT empdb1.EMPLOYEES.count + empdb2.EMPLOYEES.count +
             empdb3.EMPLOYEES.count
         FROM empdb1.EMPLOYEES, empdb2.EMPLOYEES, empdb3.EMPLOYEES
   UPDATE EVERY FRIDAY 18:00
   ON ERROR FLAG;
```

Although one of the nice features of using importer-verifiable consistency constraints is that multiple remote system interdependencies can be provided, one may not wish to use this feature in certain cases due to the increased unreliability of the quasi-view. In the the above approach, if any one of the remote systems that is queried fails, the query that updates the local copy of the quasi-view will not be able to be processed. An alternate way of setting up the interdependent data would be to make three separate interdependent relations, each containing the count of one of the remote system's relation (i.e. use *single remote system* quasi-views). That way, even if one of the systems cannot be accessed, only the quasi-view of its value will be inconsistent. And, if the system cannot be accessed because it has failed, then no updates would be made at the system. The commands that could be used to create separate quasi-views would be as follows:

```
CREATE QUASIVIEW EMP1_COUNT ( COUNT )
   AS USE empdb1
      SELECT COUNT(*)
```

```
        FROM EMPLOYEES
    UPDATE EVERY FRIDAY 18:00
    ON ERROR FLAG;

CREATE QUASIVIEW EMP2_COUNT ( COUNT )
    AS USE empdb2
       SELECT COUNT(*)
          FROM EMPLOYEES
    UPDATE EVERY FRIDAY 18:00
    ON ERROR FLAG;

CREATE QUASIVIEW EMP3_COUNT ( COUNT )
    AS USE empdb3
       SELECT COUNT(*)
          FROM EMPLOYEES
    UPDATE EVERY FRIDAY 18:00
    ON ERROR FLAG;
```

To create the total, a local view would be constructed, in the following way:

```
CREATE VIEW EMPLOYEE_COUNT ( COUNT )
    AS SELECT EMP1_COUNT.COUNT + EMP2_COUNT.COUNT + EMP3_COUNT.COUNT
       FROM EMP1_COUNT, EMP2_COUNT, EMP3_COUNT;
```

3.4 Administrative Information

This section describes the information that the importers and exporters must maintain to provide support for interdependent data.

Exporter information:

1. Maximum rate of update to quasi-view allowed.
2. Maximum percent of time allowed to process all remote queries.
3. Average time to process remote queries.
4. For each relation exported, the name, description (both type and general), and the level of consistency expressed as maximum inconsistency.

Importer information:

1. For each quasi-view
   (a) the consistency criteria used to update it.

(b) the command used to update the quasi-view.

(c) a flag indicating the validity of the quasi-view.

(d) an indicator that says what to do when there is a query on an invalid quasi-view (abort, flag, or process).

(e) the date and time of last update.

(f) the number of queries since the last update.

2. A delta list (see [Com84]) of time events to schedule queries. For "AT" conditions, schedule an event which will execute the query, for "EVERY" conditions, schedule an event that will execute the correct query and reschedule the next event.

3.5 Negotiation Algorithm

In this section we present the algorithms for the importer and exporter that are used for negotiating interdependent data. After the user has determined the data on which the quasi-view will depend, the following algorithm is used:

Importer's Algorithm:

```
for each of the remote systems do begin
    try to request a quasi-copy with a sufficient consistency criterion;
end

if any of the remote systems could not provide a requested quasi-view then
    /* interdependent data can't be created */
    remove all the quasi-views that were established;
else
    if multiple quasi-views were requested then
        create the interdependent data as a local view derived from the quasi-views;
    end
```

Exporter's Algorithm:

```
while TRUE begin
    receive request for quasi-view of data
    if data does not exist or is not exportable OR
      average time to process remote queries ≥ maximum allowed OR
      request exceeds general maximum then
        send message indicating no support;
```

```
    else
        send message indicating support and first copy of data;
end
```

3.6 Interdependent Data Support Algorithm

In this section we present the algorithms used by the importer and exporter to support interdependent data. The relative simplicity of the exporter's algorithm reflects our desire to place the burden of support for interdependent data on the importer of the data. The importer has two algorithms, the first one for processing "UPDATE AT" and "UPDATE EVERY time", and the second one for processing queries that also checks for "UPDATE EVERY ... ACCESSES" and "UPDATE WHEN ... OLD" conditions.

Importer's Algorithm:

(1) Delta List Update Algorithm

```
keep checking delta list every specified time quantum until it is time for next event;
when event occurs, process update procedure;
if the procedure fails then
    mark quasi-view as invalid;
reschedule next event (if any);
```

(2) Query Processing Algorithm

```
if quasi-view is marked as invalid OR
 number of queries since last update exceeds consistency limit OR
 age of quasi-view is older than consistency limit then begin
    query remote system
    if query succeeds then begin
        use result of query and update copy;
        set query count since last update to 0;
        set date and time of last query to current date and time;
    end
    else
        if ON ERROR condition = ABORT then
            abort transaction;
        else if ON ERROR condition = FLAG then begin
            access local copy;
            indicate that copy is out of date,
            give last time of update for data;
```

```
            end
            else if ON ERROR condition = PROCESS then
                access local copy;
    end
    else
        query local copy of quasi-view;
```

Exporter's Algorithm:

```
    while TRUE do begin
        receive remote query;
        if query can be processed within time allowed for remote queries then
            process query
        else
            do not process query
    end
```

3.7 Multi-Level Quasi-Views

One way to implement interdependent data in very large federations is to use multi-level quasi-views. In other words, one creates quasi-views of quasi-views. They can be set up in a hierarchical pattern to reduce message traffic (see figure 2). In the figure, circles represent component systems, X represents data, X_i' represents a quasi-view of X, and X_j'' represents a quasi-view of some quasi-view X_i'.

Of course, the inconsistency level of each quasi-view will increase with each level of quasi-view (or at best stay the same). To calculate the maximum level of inconsistency produced by the consistency specification of a quasi-view the following steps are taken:

1. Ignore access number consistency conditions (e.g. "UPDATE EVERY 3 ACCESSES"), unless the number specified is one. In this case, skip (2) and (3) and consider the maximum inconsistency level to be zero (i.e. the copy is always consistent). Or, if this is the *only* consistency condition specified skip (2) and (3) and consider the maximum inconsistency level unbounded.

2. Find the largest gap between two "UPDATE AT" conditions (if any).

3. Take the minimum of the maximum condition in (2) and the smallest "UPDATE EVERY" condition.

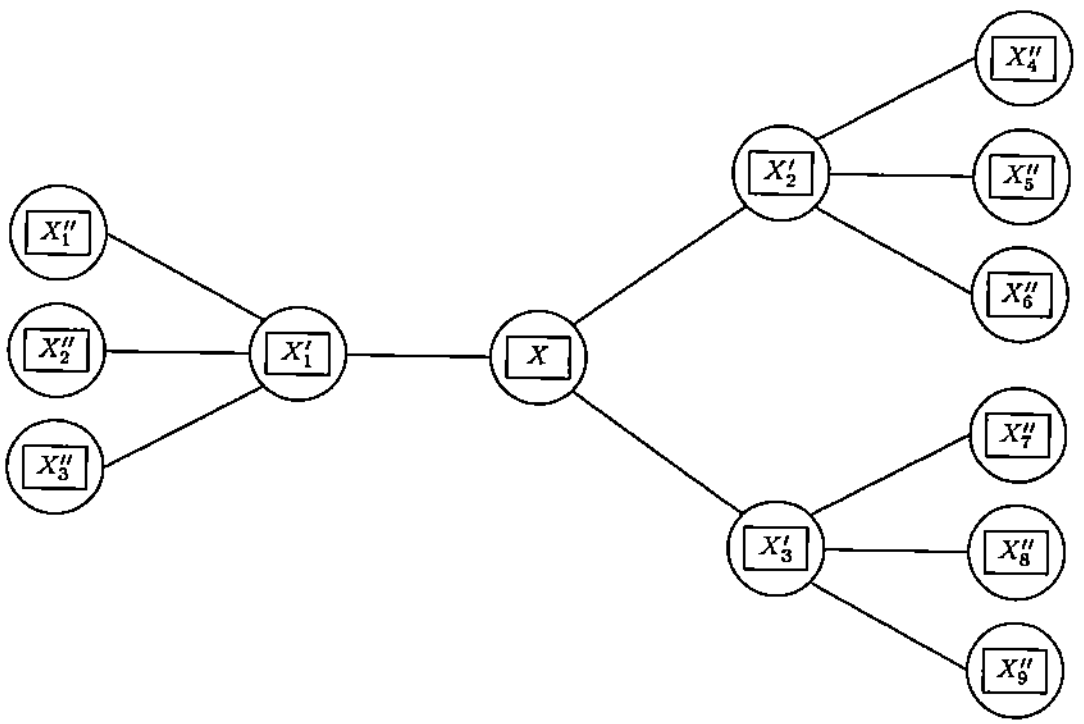

Figure 2: Mulit-Level Quasi-Views.

If a quasi-view is made of a quasi-view the maximum of the new quasi-view will be its maximum added to that of the maximum of the quasi-view from which it was constructed. Also, if a quasi-view is constructed from multiple quasi-views, we consider the maximum inconsistency level to be the maximum of the individual inconsistency levels from which the quasi-view is constructed.

3.8 Discussion of Method

Our negotiation algorithm is relatively simple, since we do not keep detailed information at the exporter sites about negotiations that have taken place. So, our method has the benefits that it is easy to implement, and the exporter is not burdened with a lot of information (and its management) in regard to negotiated agreements. In theory our agreements do not have a strong guarantee as can methods which store such information, however in practice the following effects will make it difficult for methods that *do* store detailed information at the exporter site to give strong guarantees:

- **Failures.** System and communication failures could easily prevent a system from fulfilling its agreements.
- **Dynamic Update Rate.** The method will have to take the update rate of the exported data item into consideration when making agreements. If this rate increases substantially the exporter may no longer be able to provide the level of consistency to which it previously agreed.
- **System Load.** It is also possible that increases in the amount of local queries to the database, and in the general system load might adversely affect the system's ability to meet its agreements.

And, there is no guarantee that an importer will be considerate enough to remove an agreement when it no longer requires the imported data. This could add additional complexity to the negotiation management scheme.

Since the emphasis of our work was to devise a method that would scale well to large federations, one of our goals was to allow many importers of an exporter's data, without burdening the exporter with huge storage and processing costs to provide the exported data. Our method accomplishes this goal by using importer-verifiable consistency conditions. In our method, the storage space the exporter must use is constant, regardless of the number of importers. We also describe how to calculate the maximum inconsistency level of multi-level quasi-views so that they can be used to reduce the message traffic to the original exporter of the data from which the quasi-view is constructed.

With other exporter-verifiable methods it is conceivable that the storage used to maintain the interdependent data agreements and data could be larger than that used to store the exporters database, if many importers are allowed. The problem with exporter-verifiable methods is that, in general, the exporter needs to store the following information, for *each* quasi-view that is exported:

1. The identification of the importer of the quasi view, so the exporter knows where to send the updated view, or notification that the view is inconsistent.
2. The consistency condition, so the exporter knows what consistency level is required.
3. A copy of the importers state of the consistency condition, so the exporter can check the consistency condition.

In [ABGM90] some optimizations for (exporter-verifiable) arithmetic consistency conditions on quasi-copies are discussed that work well in an environment where broadcasting is cheap.

4 Conclusions

We have presented extensions to the SQL language that can be used to negotiate interdependent data in a federated database environment. We have proposed a concept called quasi-views (which is similar to quasi-copies and distributed materialized views) to provide support for the interdependent data that is negotiated. We have presented algorithms for negotiating and supporting interdependent data, and a method for determining the maximum inconsistency of multi-level quasi-views.

Our emphasis was on developing a practical method for supporting interdependent data that scales well to large federations. Our method provides support for large numbers of importers of a data item by:

- Providing weak consistency through quasi-views. Allowing one to trade consistency for improved local system performance and decreased network traffic.
- Using importer-verifiable consistency conditions to relieve exporters from the burden of storing and processing the consistency checks for each importer.
- Supporting multi-level quasi-views to decrease network traffic.

References

[AB89] Rafael Alonso and Daniel Barbará. Negotiating data access in federated database systems. In *Proceedings of the IEEE Data Engineering Conference*, pages 56–65, 1989.

[ABGM87] R. Alonso, D. Barbará, and H. Garcia-Molina. Quasi copies: Efficient data sharing for information retrieval systems. Technical Report CS-TR-101-87, Department of Computer Science, Princeton University, 1987.

[ABGM90] Rafael Alonso, Daniel Barbará, and Hector Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[Com84] Douglas Comer. *Operating System Design, the Xinu Approach.* Prentice Hall, 1984.

[Dat90] C. J. Date. *An Introduction to Database Systems*, volume I. Addison Wesley, fifth edition, 1990.

[HM85] D. Heimbinger and D. Mcloed. A federated architecture for information management. *ACM transaction on office information systems*, 3(3), July 1985.

[LAZ+87] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas, and Ph. Vigier. Msql: A multidatabase language. Technical Report Technical Report 695, INRIA, BP 105, 78153 Le-Chesnay, France, 1987.

[MEK91] J. G. Mullen, A. K. Elmagarmid, and W. Kim. On the impossibility of atomic commitment in multidatabase systems. Technical Report CSD-TR 91-029, Department of Computer Science, Purdue University, April 1991.

[RS91] Marek Rusinkiewicz and Amit Sheth. Polytransactions for managing interdependent data. *IEEE Data Engineering Bulletin*, 14(1), March 1991.

[SK89] A. Sheth and P. Krishnamurthy. Redundant data management in bellcore and bcc databases. Technical Report TM-STS-015011, Bellcore Technical Memorandum, December 1989.

[SL90] Amit P. Sheth and James A. Larson. Federated databases systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, pages 183–236, September 1990.

[SLE91] A. Sheth, Y. Leu, and A.K. Elmagarmid. Maintaining consistency of interdependent data in multidatabase systems. Technical Report CSD-TR-91-016, Department of Computer Science, Purdue University, 1991.

[SP89] Arie Segev and Jooseok Park. Updating distributed materialized views. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):173–184, June 1989.

[SR90] A. Sheth and M. Rusinkiewicz. Management of interdependent data: Specifying dependency and consistency requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 5–11, Houston, TX, November 1990.