Resolving Inconsistencies in the Multiplex Multidatabase System *

Philipp Anokhin Amihai Motro

May 21, 1999

Abstract

With the development of the Internet and the on-line availability of large numbers of information sources, the problem of integrating multiple heterogeneous information sources requires reexamination, its basic underlying assumptions having changed drastically. Integration methodologies must now contend with situations in which the number of potential data sources is very large, and the set of sources changes continuously. In addition, the ability to create quick, ad-hoc virtual databases for short-lived applications is now considered attractive. Under these new assumptions, a single, complete answer can no longer be guaranteed. It is now possible that a query could not be answered in its entirety, or it might result in several different answers. Multiplex is a multidatabase system designed to operate under these new assumptions. In this paper we describe how Multiplex handles queries that do not have a single, complete answer. The general approach is to define flexible and comprehensive *strategies* that direct the behavior of the query processing subsystem. These strategies may be defined either as part of the multidatabase design or as part of ad-hoc queries.

*This work was supported in part by ARPA grant, administered by the Office of Naval Research under Grants No. N0014-92-J-4038 and N0060-96-D-3202.

1 Introduction

One of the most enduring problems in database research has been the integration of information from multiple, heterogeneous data sources. A common approach to this problem is to provide an integrated view of the data by means of a *global scheme*, map this scheme into the schemes of the various sources, develop query translation algorithms that interpret queries against the global scheme into queries against the sources, and then combine the individual answers received from the sources into an answer to the original query. This approach has often been referred to as the *virtual database* approach.

Originally, the underlying assumptions were that the number of source databases is relatively small, and that the scheme of each database is completely understood. In addition, both the set of source databases and the scheme of each source database were assumed to be static (and when either of these changed, substantial redesign of the virtual database was to be expected). The considerable amount of work required to comprehensively integrate the source databases was justified by the expectations of extensive and prolonged use of the resulting virtual database. Such comprehensive integration ensured that every query on the global scheme had exactly one translation, so that every global query was guaranteed a single, complete answer.

The development of the Internet and the on-line availability of large numbers of information sources have changed these assumptions drastically. Integration methodologies must now contend with situations in which the number of potential sources is very large, and the set of sources changes continuously, with new sources being discovered and added, and current sources changing their structure or disappearing altogether, sometimes without any prior notice. In addition, the ability to create quick, ad-hoc virtual databases for short-lived applications is now considered attractive. In this new breed of virtual databases, integration is no longer comprehensive: the number and complexity of the sources on one hand, and the scope of the applications on the other hand, suggest integration of a more limited extent, in which only small portions of the source databases are brought together. Under these new assumptions, a single, complete answer can no longer be guaranteed. It is now possible that a query could not be answered in its entirety, or it might result in several different answers.

Multiplex is a multidatabase model designed to operate under these new assumptions [14]. The advantages of the Multiplex model are numerous: (1) The global scheme, and hence its user interfaces, are based on the popular *relational* model. (2) It supports *heterogeneity*; that is, the source databases may be structured according to any database model, as long as they can communicate their results in tabular format. (3) It facilitates the creation of ad-hoc global schemes of limited scope, that cull from the source databases only the information relevant to a given application. (4) It facilitates the tracking of evolving data environments, because the addition or removal of new sources of information is governed by a single resource that describes the contributions of the various sources. (5) It reflects the dynamics of multidatabase environments where source databases may become temporarily unavailable, and global queries might therefore be unanswerable in their entirety. (6) It accepts that requested information may be found in more than one database, and admits

the possibility of inconsistency among these alternatives.

Viewed together, items 5 and 6 address a frequent situation in which it is not possible to extract a single authoritative answer from the aggregate of source databases. This may be either because some of the information requested in the query is unavailable (item 5), or because some of the information requested in the query is available from multiple (possibly inconsistent) sources (item 6). When "too little" information is available, a multidatabase system should issue a partial answer, and when "too much" information is available, it should resolve the inconsistencies in a single, "best" answer. Note that these two problems could occur together, requiring inconsistency resolution in partial answers.

In a previous paper, we described a solution that addressed the unavailability of information by computing upper and lower bound answers; that is, an unavailable answer is approximated by a pair of answers consisting of the largest contained set of tuples and the smallest containing set of tuples. The solution to the issue of inconsistency was largely based on voting; that is, when multiple versions of the information are available, the version advocated by most sources is preferred.

An advantage of that method for handling inconsistencies is that it can be applied "automatically", without any human intervention. In practice, however, we have observed that many applications require more powerful resolution strategies. Resolution strategies may be classified into two types. When an item of information is available from several sources, it may be desirable (1) to *prefer* one of the sources on the basis of available meta-information, such as source *quality* or *currentness*, or (2) to *consolidate* the alternative data values, for example, adopt the *average* or the *minimum* of the alternative values.

In this paper we describe a comprehensive approach that allows users to specify appropriate conflict resolution strategies. Some strategies are simple; for example, they construct all answers with the most current data. Other strategies may be quite elaborate; for example, in a single query, it may be desirable to resolve age values by their average, salary values by the most current, and email addresses by the mode (the most frequently mentioned value, similar to voting).

Conflict resolution strategies may be considered either as part of the multidatabase *design* specification, to be provided by the multidatabase designer as part of the global scheme, or as part of the *retrieval* specification, to be provided by the multidatabase user as part of each query. Indeed, a three-tier approach may be conceived, in which user strategies override designer strategies, which, in turn, override system default strategies. With this latter tier, the aforementioned advantage of automatic conflict resolution is recouped.

Section 2 reviews the basic concepts and definitions of the Multiplex multidatabase model. The version described admits the possibility of inconsistencies, but does not attempt to resolve them. Sections 3 and 4 are the center of this paper. They describe our formal framework for resolving inconsistencies and the syntax and semantics of the **resolve** statement. In Section 5 compares our work to other research, and Section 6 reports on the implementation of the Multiplex multidatabase system and sketches our future research plans.

2 The Multiplex Multidatabase Model

In this section we define the database concepts that will be used throughout this work. Our formalization of relational databases is mostly conventional.

2.1 Schemes and Instances, Views and Queries

Assume a finite set of attributes T, and for each attribute $A \in T$ assume a finite domain $dom(A)$, and assume a special value called *null* and denoted $-$, which is not in any of the domains. A *relation scheme* R is a sequence of attributes from T. A *tuple* t of a relation scheme $R = (A_1, \ldots, A_m)$ is an element of $dom(A_1) \cup \{-\} \times \cdots \times dom(A_m) \cup \{-\}$. A *relation instance* (or, simply, a *relation*) r of a relation scheme R is a finite set of tuples of R. A *database scheme* D is a set of relation schemes $\{R_1, \ldots, R_n\}$. A *database instance* d of the database scheme D is a set of relations $\{r_1, \ldots, r_n\}$, where r_i is a relation on the relation scheme R_i $(i = 1, \ldots, n)$. Finally, a *database* (D, d) is a combination of a database scheme D and a database instance d of the scheme D.

Let D be a database scheme. A *view* of D is an expression that defines (1) a new relation scheme V, and (2) for each instance d of D an instance v of V. v is called the *extension* of the view scheme V in the database instance d.

We shall assume that all views are of the family known as *conjunctive* views [16]. Although conjunctive views are a strict subset of the relational tuple calculus, they are a powerful subset, corresponding to the set of relational algebra expressions with the operations *Cartesian product*, *selection* and *projection* (where the selection predicates are conjunctive).

A *query* Q on a database scheme D is a view of D. The extension of Q in a database instance d of scheme D is called the *answer* to Q in the database instance d.

As an example, given the relation schemes $Emp = (Name, Salary, Dname)$ and $Dept = (Dname, Supervisor)$, a view Emp_sup may be defined by

$$project_{Name,Supervisor}\, select_{Emp.Dname=Dept.Dname}\, Emp \times Dept$$

The scheme of Emp_sup is (*Name,Supervisor*).

Our basic definitions allowed nulls in relation instances. This requires appropriate extensions to the model to determine the results of comparisons that involve nulls. Here, we adopt Codd's three-valued logic, in which comparisons that involve nulls evaluate to the value *maybe*. In general, we shall provide control over the interpretation of *maybe* values. A *permissive* interpretation will map *maybe* values to *true*, and a *restrictive* interpretation will map *maybe* values to *false*.

2.2 Scheme Mappings

Consider a database (D, d). Let D' be a database scheme whose relation schemes are defined as views of the relation schemes of D. The database scheme D' is said to be *derived* from the database scheme D. Let d' be the database instance of D' which is the extension of the views D' in the database instance d. The database instance d' is said to be *derived* from the database instance d. Altogether, a database (D', d') is a *derivative* of a database (D, d), if its scheme D' is derived from the scheme D, and its instance d' is derived from the instance d.[1]

Let (D_1, d_1) and (D_2, d_2) be two derivatives of a database (D, d). The derivative databases are mutually "consistent" in the sense that "equivalent" views are extended identically in the databases in which they apply. These notions of view equivalence is defined formally as follows.

A view V_1 of D_1 and a view V_2 of D_2 are *equivalent*, if for every instance d of D the extension of V_1 in d_1 and the extension of V_2 in d_2 are identical. Intuitively, view equivalence allows us to substitute the answer to one query for an answer to another query, although these are different queries on different schemes.

Given two different database schemes, which are both derivatives of the same data scheme, we express their commonality by means of scheme mappings.

Assume two database schemes D_1 and D_2, which are both derivatives of a database scheme D. A *scheme mapping* (D_1, D_2) is a collection of view pairs $(V_{i,1}, V_{i,2})$ $(i = 1, \ldots, m)$, where each $V_{i,1}$ is a view of D_1, each $V_{i,2}$ is a view of D_2, and $V_{i,1}$ is equivalent to $V_{i,2}$.

As an example, the equivalence of attribute *Salary* of relation scheme *Emp* in database scheme D_1 and attribute *Sal* of relation scheme *Employee* in database scheme D_2 is indicated by the view pair

$$(\pi_{Salary} Emp,\ \pi_{Sal} Employee)$$

As another example, given the relation schemes *Emp = (Name, Title, Salary, Supervisor)* in database scheme D_1, and *Manager = (Ename, Level, Sal, Sup)* in database scheme D_2, the retrieval of the salaries of managers is performed differently in each database, as indicated by the view pair

$$(\pi_{Name,Salary} \sigma_{Title=manager} Emp,\ \pi_{Ename,Sal} Manager)$$

2.3 Multidatabase

To define a multidatabase, we shall assume that there exists a single (hypothetical) database that represents the real world. This ideal database includes the usual components of scheme

[1] In this paper we are not concerned with an effective procedure for determining whether one database is a derivative of another, a question that depends on the language for expressing views. For our purpose here, it is sufficient to note that a database may or may not be a derivative of another database.

and instance, which are assumed to be perfectly correct. We now formulate two assumptions. These assumptions are similar to the Universal Scheme Assumption and the Universal Instance Assumption [13], although their purpose here is quite different. These two assumptions are statements of the *integrability* of the given databases.

The Scheme Consistency Assumption (SCA). All database schemes are *derivatives* of the real world scheme. That is, in each database scheme, every relation scheme is a view of the real world scheme. The meaning of this assumption is that the different ways in which reality is modeled are all correct; i.e., there are no *modeling errors*, only *modeling differences*. To put it in yet a different way, all intensional inconsistencies among the independent database schemes are reconcilable.

The Instance Consistency Assumption (ICA). All database instances are *derivatives* of the real world instance. That is, in each database instance, every relation instance is derived from the real world instance. The meaning of this assumption is that the information stored in databases is always correct; i.e., there are no factual *errors*, only different *representations* of the facts. In other words, all extensional inconsistencies among the independent database instances are reconcilable.

The Multiplex model assumes that the Scheme Consistency Assumption *holds*, meaning that all differences among database schemes are reconcilable, and that the Instance Consistency Assumption *does not hold*, allowing the possibility of irreconcilable differences among database instances.

In other words, the member databases are all assumed to have schemes that are derivatives of a hypothetical real world database scheme; these schemes are related through the multidatabase scheme, which is yet another derivative of this perfect database scheme. But the member database instances are not assumed to be derivatives of the real world instance.

Clearly, without subscribing to the SCA, it is not possible to integrate a given set of databases. On the other hand, subscribing to the ICA would not reflect the reality of independently maintained databases.

Formally, a *multidatabase* is

1. A scheme D.
2. A collection $(D_1, d_1), \ldots, (D_n, d_n)$ of databases.
3. A collection $(D, D_1), \ldots, (D, D_n)$ of scheme mappings.

The first item defines the scheme of a multidatabase, and the second item defines the member databases in the multidatabase environment. The third item defines a mapping from the global scheme to the schemes of the member databases. The schemes D and $D_1, \ldots, D_n$ are assumed to be derivatives of the real-world scheme, but the instances $d_1, \ldots, d_n$ are not necessarily derivatives of the real-world instance (see Figure 1).

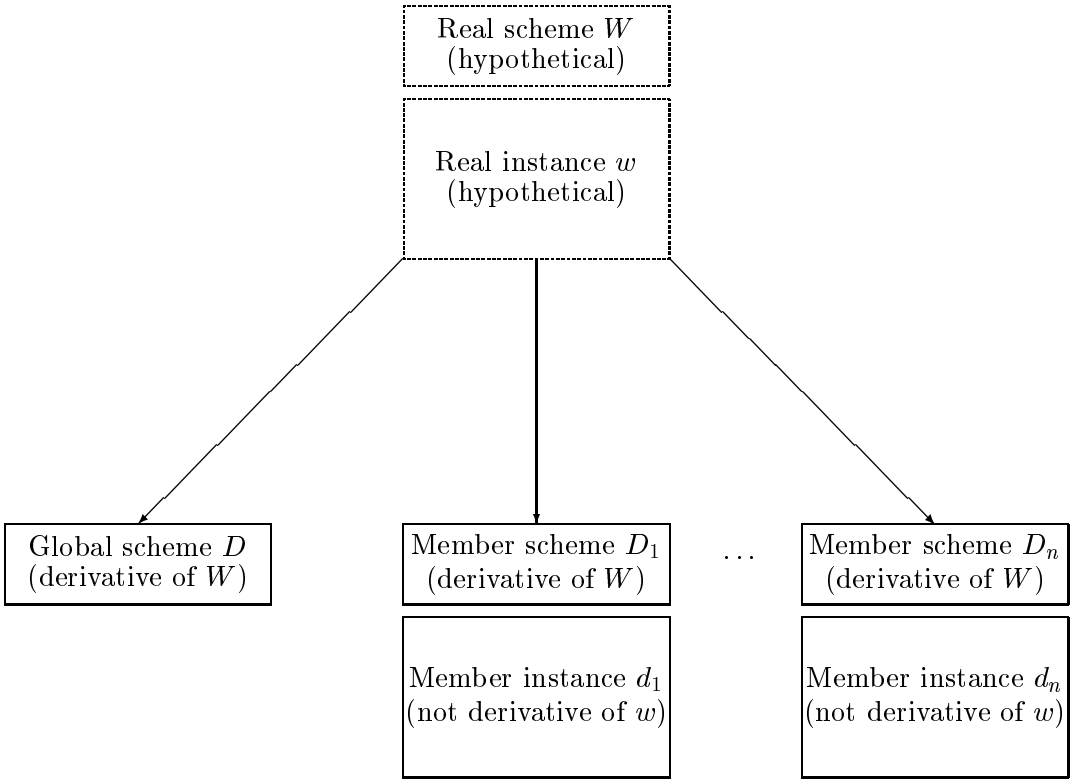

Figure 1: Consistency assumptions in Multiplex.

The "instance" of a multidatabase consists of a collection of global view extensions that are available from the member databases. Specifically, the views in the first position of the scheme mappings specify the "contributed information" at the global level, and the views in the second position describe how these contributions are materialized.

As defined earlier, scheme mappings allow to substitute certain views in one database with equivalent views in another database. In a multidatabase, the former database is the global database, and the latter is a member database.

Our definition of multidatabases provides four important "degrees of freedom", which reflect the realities of multidatabase environments.

First, the mapping from D to the member schemes is not necessarily *total*; i.e., not all views of D are expressible in one of the member databases (and even if they are expressible, there is no guarantee that they are mapped). This models the dynamic situation of a multidatabase system, where some member databases might become temporarily unavailable. In such cases the corresponding mappings are "suspended", and some global queries might not be answerable in their entirety. Similarly, if an authorization mechanism is enforced, a user may not have permission to some views.

Second, the mapping is not necessarily *surjective*; i.e., the member databases may include views that are not expressible in D (and even if they are expressible, there is no guarantee

that they are mapped). This models the pragmatism of multidatabases, which usually cull from existing databases only the information which is relevant to a specific set of applications. For example, a large database may share only one or two views with the multidatabase.

Third, the mapping is not necessarily *single-valued*; i.e., a view of D may be found in several member databases. This models the realistic situation, in which information is found in several overlapping databases, and provides a formal framework for dealing with multidatabase inconsistency. Recall that if we do not assume that the Instance Consistency Assumption holds, then we do not assume that the member instances are all derived from a single instance. Thus, the inclusion of view pairs (V, V_1) and (V, V_2) in two scheme mappings of a multidatabase does not imply that the extensions of V in the member databases are identical. Rather, it implies that they *should be* identical.

Fourth, while the definition assumes that the member databases adhere to the relational model defined here, they need not be relational, or even of the same data model. Recall that the only purpose of the views in the second position of the scheme mappings is to describe how the views in the first position are materialized. Therefore, the member databases need not be relational, and the views in the second position need not be relational expressions. The only requirement is that they compute tabular answers.

2.4 Multidatabase Queries and Answers

Syntactically, a multidatabase query is simply a query Q of the scheme D. Intuitively, the answer to a multidatabase query Q should be obtained by transforming it to an equivalent query of the views in the first position of the scheme mappings (the available information). These views would then be materialized (using the view definitions in the second position of the scheme mappings), and the translated query would be processed on these materialized views. Formally, the required transformation of Q is stated as follows.

Let $D = \{R_1, \ldots, R_n\}$ denote a database scheme, and let $M = \{V_1, \ldots, V_m\}$ denote a set of views of D. Translate a given query $Q(D)$ of the database scheme to an equivalent query $Q'(M)$ of the view schemes.

However, a solution to this translation problem may not exist, or there could be multiple solutions. To observe that multiple solutions may exist, consider a database with a relation $R = (A, B, C)$ and views $V_1 = \pi_{A,B}R$ and $V_2 = \pi_{A,C}R$, and consider the query $Q = \pi_A R$. Q can be answered from both V_1 or V_2. To observe that a solution may not exist, consider a database with two relations $R = (A, B)$ and $S = (B, C)$, and one view $V = R \bowtie S$, and consider the query $Q = \sigma_{A=a}R$. Clearly, Q cannot be answered from the view V, because the join would not necessarily include all of R's tuples.

This translation problem (for conjunctive queries and views) has been addressed by Larson and Yang [10, 11], by Levy et al. [12] and by Brodsky and Motro [5]. We shall assume that a translation algorithm exists which is sound and complete; i.e., it computes all the correct translations that exist.

Assume a multidatabase with scheme D and mapped views M. The *answer* to a query Q on this multidatabase is the set of answers produced by a sound and complete translation algorithm.

There are two possible cases:

1. When the translation algorithm produces more than one solution, these solutions may evaluate to different answers. Each such answer is a *candidate answer*. The answer to Q is the set of all candidate answers.[2]

2. When a solution to the translation problem does not exist, the answer to Q is the empty set of answers. This empty set of answers should be interpreted as *answer unavailable*.[3]

3 A Framework for Handling Inconsistencies

3.1 The Issues

From a user perspective, each legitimate database query should evaluate to a single answer. The multidatabase answers defined in the previous section deviate from this ideal in two cases: when no answer is available, and when several different answers are available. In either case, it is clear that a single *perfect* answer (i.e., an answer identical to the real world answer) cannot be determined from the multidatabase environment. At best, the system can provide an *approximation* of this elusive perfect answer. In this section, we discuss important extensions to the basic Multiplex model to handle these two situations.

No answer is available. Intuitively, a global query cannot be translated to an equivalent query of the available views, because the mapping of the global scheme to the member schemes is *not total*; i.e., some information "promised" in the global scheme cannot be "delivered", either because it has never been mapped, or because some member databases are not responding. In situations where a query Q cannot be rewritten as an equivalent query of the available views, an issue of great importance is how well can Q be *approximated* using the available views; i.e., what is the best approximation of Q that can be evaluated from the views?

Multiple answers are available. Intuitively, a global query is translatable to different equivalent queries over the available views, because the mapping of the global scheme to the member schemes is *not single-valued*; i.e., there exists a view of the global scheme that can be materialized in more than one way. This happens when two view pairs of the mapping have the *same* view in their first position. Obviously, for every translation that uses one view, there is an equivalent translation that uses the other view. More generally, it happens when

[2]Note that possibly none of the candidate answers is consistent with the real world answer.

[3]Note the difference between an empty set of answers and an empty answer.

two pairs have *overlapping* views in their first position, as this implies that the intersection view can be mapped in two different ways. The most common reason for such multivalued mappings is that the information resources have overlapping information. Unless the ICA holds, these different translations could evaluate to different answers. In situations where there are different ways to rewrite Q as an equivalent query of the available views, and they evaluate to different answers, an issue of great importance is whether any one specific answer should be *preferred* over the others, or how the answers could be *combined* into a single answer.

3.2 The Approach

Given a global query, our approach is to populate the virtual relations targeted by the query with data gathered from those contributed views that are relevant to the query. During this process, both of the above issues are addressed.

No answer is available. Each available view is analyzed for all its possible contributions to those parts of the virtual relations that are targeted by the query. Essentially, every selection-projection subview of the virtual relations that contains data relevant to the query is deployed. These subviews may even be incomplete in the attributes that they provide, in which case they will be expanded with null values. This implies that later, when the query is processed in the global relations, answers that are unavailable are approximated by *partial answers* (i.e., tables with scattered nulls).

Multiple answers are available. When the requisite parts of the global relations are assembled, inconsistencies among the available views are detected and resolved. This implies that later, when the query is processed in the global relations, a single answer would be issued.

It should be emphasized that we do not construct global relations in their entirety. Rather, we request from the component databases only those parts that are relevant to the query. Our construction is a form of translation. In general, a translation paradigm (1) decomposes a query Q on the scheme D into a set of queries $\{Q_i\}$ on the views M, (2) retrieves the corresponding answers $\{A_i\}$, and (3) combines them into an answer A. Our method uses this specific third step: the retrieved data is assembled into relations to which the original query is then applied.

To combine different answers, some mechanism is needed to determine when different tuples (from different sources) are indeed different versions of the same tuple. One such mechanism are *keys*. If we assume that all contributions to a global relation include the relation's key attributes, then we may conclude that different tuples are versions of the same tuple if they have the same key values. Another possible mechanism is *tuple similarity*. If we define a semantic similarity among tuples of the same relation, then we may conclude that different tuples are versions of the same tuple, if their similarity exceeds a predefined threshold. We shall assume that a *tuple identification mechanism* of some kind is being

enforced. Whichever mechanism is chosen, we shall assume that each member database provides a set of tuples that does not contain tuples that are versions of each other (e.g., no two tuples have the same key, or no two tuples are "too similar").

3.3 Contributions and Properties

The n scheme mappings in the definition of multidatabases may be represented in a single two-column *mapping table*: the first column contains views of the global scheme, and the second column contains instructions for materializing these views from the member databases. Each row in this table describes an individual "contribution". The global view definition in the first position of the row *defines* this contribution, and the expression in the second position of the row shows how to *materialize* it.

In practice, we must consider the possibility that the instance materialized by the expression in the second column is *inconsistent* with the global view definition in the first column. In general, a view definition involves a product, followed by a selection, and followed by a projection. This leads to three types of inconsistency:

Projection: The instance has an incorrect number of columns.

Selection: The instance includes tuples that are inconsistent with the selection condition. For example, the view definition may include the condition $A > 3$ or $B = C$, yet a tuple would include the value 2 for A, or different values for B and C.

Product: The instance is not a product of instances. For example, the view may be defined as the product of relations $R = (A, B)$ and $S = (C, D)$, yet the instance would include only two tuples (a_1, b_1, c_1, d_1) and (a_2, b_2, c_2, d_2), which, clearly, could not be the product of any two instances.

Contributions that are inconsistent with their definitions may be explained in two ways: (1) The definition of the contribution (i.e., the mapping of the global scheme to the member scheme) is correct; it is the instance of the source database which is inconsistent with its scheme. In the last example, the source relation is indeed the product of R and S, but it is missing two additional tuples (a_1, b_1, c_2, d_2) and (a_2, b_2, c_1, d_1). (2) The source instance is consistent with its scheme; it is the definition of the contribution (i.e., the mapping) which is incorrect. In the same example, the two-tuple instance is correct, but it is simply not the product of R and S.

It is relatively simple to detect contribution instances that are inconsistent with their definitions, and there are three possible approaches for handling such situations: (1) With the exception of projection violations, the instances could be accepted, ignoring their inconsistencies, (2) contributions that exhibit inconsistencies could be *discarded* altogether, and (3) attempts could be made to *correct* the inconsistencies. Our approach here is to discard all

inconsistent contributions. Whenever we cannot detect any of these 3 types of inconsistency, we assume that the definition of the contribution is correct.

Instances of contributions may be associated with *properties*. Each property provides meta-information about the instance, such as the quality of the data, its currentness (timestamp), or its security classification. Each property is described by a pair (*(keyword=value)*; for example, *quality=0.8*, *timestamp=990305*, or *security="high"*, and a contribution instance may be associated with any number of such property pairs.

Some properties that are associated with instances can also be associated with each individual tuple of the instance. For example, the property *timestamp=x*, which denotes the currentness of an instance, can be associated with individual tuples of the instance. On the other hand, the property *count=x*, which denotes the cardinality of an instance, cannot be associated with individual tuples. Properties that can be associated with individual tuples will be referred to as *tuple-applicable* properties.

3.4 Decomposing Contributions

Given a global query, our aim is to populate the virtual relations targeted by the query with data gathered from those contributed views that are relevant to the query. This task raises two difficulties. First, a contribution may not provide all the attributes of a global relation. This is solved by using null values for the attributes not provided. Second, a contribution may provide tuples to more than one global relation. This is approached by decomposing such contributions to contributions to single relations. In some cases the decomposed tuples that originate from a single tuple, must remain associated. To accomplish this, we assume that every relation R_i has an additional pseudo-attribute, denoted Δ.

Consider a contribution C to the global database. In general, C is defined as the product of several global relation schemes, followed by a selection, and followed by a projection: with the global relations.

$$C = \pi_\alpha \sigma_\theta R_1 \times \cdots \times R_n$$

Possibly, the projection attributes α are contained in a *single* global relation scheme, meaning that a contribution instance c contributes tuples to a single global relation. Let R_i be the relation scheme that contains all the attributes α. We enlarge the contribution instance c by adding null values for the values of the attributes of $R_i - \alpha$ (the attributes of R_i that are not in the contribution).

Alternatively, the projection attributes α may contain attributes from *several* relation schemes, meaning that a contribution instance c contributes tuples to several global relations. We *decompose* a contribution instance c to contributions to the individual relations that share attributes with α. This decomposition is made in several steps. Let R_i be a relation that shares some attributes with α.

1. First, we project c on the attributes $R_i \cap \alpha$ (the attributes of the R_i that appear in the contribution).

2. Next, we enlarge the resulting instance by adding null values for the values of the attributes of $R_i - \alpha$ (the attributes of R_i that are not in the contribution). Each tuple t of the original instance c is now decomposed into several null-extended tuples in different instances.

3. Finally, we add a pseudo-attribute Δ to R_i. If the condition θ refers to attributes that are not in α *and* it contains comparisons that involve more than one relation, then all the tuples that originate from the same tuple t receive a unique Δ value; otherwise, Δ values are simple nulls.

At the end of this process, each global relation scheme R is associated with a set of such *fragments* $r_1, \ldots, r_n$, each originating from a different contribution. Let ϕ_i denote the selection condition of the contribution that produced the fragment r_i. ϕ_i defines the part of R to which r_i contributes tuples. Note that ϕ_i may involve attributes that are outside the relation scheme R.

As an example, assume the global schemes $R = (A, B, C)$ and $S = (C, D, E, F)$, a contribution defined by $C_1 = \pi_{A,B,D,E}\sigma_{(R.C=S.C)\wedge(B>4)\wedge(D>3)}R \times S$, and a contribution instance

c_1

A	B	D	E
a_1	5	7	e_1
a_2	6	4	e_2
a_1	5	4	e_3

This selection condition compares attributes in R and S, and it refers to an attribute C which is not among the projection attributes. Hence, the decomposition requires pseudo-values to link the tuples of the fragments. The instance c_1 generates two fragments

$r(R)$

Δ	A	B	C
δ_1	a_1	5	–
δ_2	a_2	6	–
δ_3	a_1	5	–

$s(S)$

Δ	C	D	E	F
δ_1	–	7	e_1	–
δ_2	–	4	e_2	–
δ_3	–	4	e_3	–

Both r and s are associated with the selection condition $(R.C = S.C) \wedge (B > 4) \wedge (D > 3)$.

As another example, assume the same global schemes $R = (A, B, C)$ and $S = (C, D, E, F)$, a contribution defined by $C_2 = \pi_{A,B,C,D,E}\sigma_{(R.C=S.C)\wedge(B>4)\wedge(D>3)}R \times S$, and a contribution instance

c_2

A	B	C	D	E
a_1	5	6	7	e_1
a_2	6	3	4	e_2
a_1	5	6	4	e_3

Because all the attributes in the selection condition are among the projection attributes, this decomposition does not require pseudo-values. The instance c_2 generates two fragments

$r(R)$

Δ	A	B	C
–	a_1	5	6
–	a_2	6	3

$s(S)$

Δ	C	D	E	F
–	6	7	e_1	–
–	3	4	e_2	–
–	3	4	e_3	–

Again, both r and s are associated with the selection condition $(R.C = S.C) \wedge (B > 4) \wedge (D > 3)$.

An important concern here is that the process of decomposing contributions into fragments of individual relations would be correct; that is, it would neither store more information than provided, nor less. The criteria of correctness is that the *reversal* of the decomposition (i.e., the retrieval of the view C from the decomposed fragments) would generate the original contribution instance. Assume the contribution $C = \pi_\alpha \sigma_\theta R_1 \times \cdots \times R_n$ and the contribution instance c, and let $r_1, \ldots, r_n$ denote the generated fragments. It can be shown that

$$c = \pi_\alpha \sigma_\theta r_1 \times \cdots \times r_n$$

Where the product is interpreted as an equijoin on Δ. Note that the final projection discards any pseudo-attributes. For lack of space we omit the proof of this claim.

3.5 Slices, Polyinstances, and Mosaic Instances

Consider two contributions to a relation *Employee*: the set of all *female* employees and the set of all employees who are *engineers*. It is important to observe that the two contributions overlap, and hence might induce inconsistencies, only in the set of *female engineers*. When constructing a relation from different fragments, we must intersect their selection predicates to obtain those "areas of contention". Conflict resolution is conducted only among the sets of tuples that belong to these areas.

Let f_1 and f_2 be two fragments of a global scheme R, and let ϕ_1 and ϕ_2 denote their selection conditions, correspondingly. These two conditions divide R into three mutually

exclusive "slices", which are defined by the predicates:

$$\begin{array}{rccc} \psi_1: & \phi_1 & \wedge & \phi_2 \\ \psi_2: & \phi_1 & \wedge & \neg\phi_2 \\ \psi_3: & \neg\phi_1 & \wedge & \phi_2 \end{array}$$

Obviously, only f_1 could contribute tuples to the slice defined by ψ_2, and only f_2 could contribute tuples to the slice defined by ψ_3. However, if the selection conditions are not contradictory, i.e., $\phi_1 \wedge \phi_2 \neq false$, then both fragments could contribute tuples to the slice defined by ψ_1, leading to the possibility of inconsistency. Hence, each slice is associated with either one or two sets of tuples.

In general, let $f_1, \ldots, f_n$ be fragments of R, and denote $\phi_1, \ldots, \phi_n$ their corresponding selection conditions. These fragments define $2^n - 1$ mutually exclusive slices that are defined by $2^n - 1$ mutually contradictory predicates. These predicates are formed by conjoining the n selection conditions, where the ith condition is either ϕ_i or $\neg\phi_i$. The new predicates are

$$\begin{array}{rccccccccc} \psi_1: & \phi_1 & \wedge & \phi_2 & \wedge & \ldots & \wedge & \phi_{n-1} & \wedge & \phi_n \\ \psi_2: & \phi_1 & \wedge & \phi_2 & \wedge & \ldots & \wedge & \neg\phi_{n-1} & \wedge & \phi_n \\ & & & & \vdots & & & & & \\ \psi_{2^n-1}: & \neg\phi_1 & \wedge & \neg\phi_2 & \wedge & \ldots & \wedge & \neg\phi_{n-1} & \wedge & \phi_n \end{array}$$

Formally, a *slice* S of a global relation scheme R is a pair (ψ, s):

1. ψ is the predicate of the slice, as defined above.
2. s is a *polyinstance*; i.e., a collection of sets of tuples. Each of the instances in s is derived from a different fragment (and hence from a different contribution) in the following way. Each predicate ψ is applied to each fragment f_i. Consequently, for each ψ, a collection of sets of tuples is derived, where each such set has been obtained from a different fragment.[4]

Note that some of the ψ predicates may evaluate to *false*, in which case the corresponding polyinstance will be empty.

The application of a predicate ψ to a fragment is complicated by the possibility of null values in fragments. Consider the selection predicates $\phi_1 = (C < 10)$ and $\phi_2 = (C \geq 10)$, and the following fragment f

A	B	C
1	2	12
2	3	–
7	1	5

[4]Note that some of these sets would be empty. A contribution generates a non-empty set of tuples, if the selection predicate of the contribution appears positively in ψ.

In this case, $\psi_1 = \phi_1 \wedge \phi_2 = false$, $\psi_2 = \phi_1 \wedge \neg\phi_2 = (C < 10)$, and $\psi_3 = \neg\phi_1 \wedge \phi_2 = (C \geq 10)$. Obviously, ψ_1 will generate an empty set of tuples. The first tuple will be selected by ψ_3 and discarded by ψ_2, whereas the third tuple will be selected by ψ_2 and discarded by ψ_3. However, the decision on the second tuple is complicated, as the true value of C is unknown. According to Codd's three-valued logic, the outcome of both ψ_2 and ψ_3 for the second tuple is *maybe*. A permissive interpretation maps these *maybe* values to *true*, thereby inserting the tuple into both sets. A restrictive interpretation maps these *maybe* values to *false*, thereby inserting the tuple into neither set.[5][6]

The collection of polyinstances will be referred to as a *mosaic instance* of the global relation scheme R. Our goal is to convert this mosaic instance to an ordinary instance. This goal would be achieved by separately converting every polyinstance into an ordinary instance, and then taking their union.

Converting a polyinstance into an ordinary instance is straightforward, unless the polyinstance contains at least two different instances, in which case the conflicts must be *resolved*.

4 Conflict Resolution

4.1 The Approach

Conflicts are resolved by means of conflict resolution statements. Each global relation requires one statement. However, a statement may allow different conflict resolution strategies for different "slices" of the relation. Formally, the statement involves a set of selection predicates which are mutually exclusive and their disjunction is a tautology. These predicates partition the relation in a way similar to the slices defined earlier.

Altogether, the global relation is "sliced" in two different ways: First, by the contributions, and then by the conflict resolution statement. These two partitions are combined, by using the partition defined in the statement to *refine* the partition defined by the contributions. Consequently, the mosaic instance of the global relation consists of a larger number of polyinstances, each for a "narrower" slice. Each strategy now governs several slices, but each slice is governed by a single strategy.

To describe our approach to conflict resolution, we consider a single slice and its polyinstance. The conflict resolution process has three stages.

1. In the first stage, we filter the set of *contributions* to the polyinstance. This filtration is based on the properties of the contributions. For example, a possible strategy is to

[5]Note that because the ψ predicates *partition* the fragment, this tuple belongs in exactly one of the partition sets.

[6]The interpretation to be applied is stated as part of the conflict resolution strategy, to be described in Section 4.2.

consider only contributions whose *timestamp* is later that a certain value. Note that the properties used in this stage need not be tuple-applicable.

2. The remaining tuples are now grouped into *multituples*. A multituple is a collection of tuples that are versions of each other. When keys are used to identify tuples, then a multituple can be visualized as a table

	a_{11}	a_{12}		a_{1k}
key	a_{21}	a_{22}	$\ldots$	a_{2k}
	$\vdots$	$\vdots$		$\vdots$
	a_{p1}	a_{p2}		a_{pk}

   At this point we disregard the original tuple associations and consider a multituple as a single "tuple" with multiple values in each of its attributes (i.e., a non First-Normal-Form tuple). The second stage allows us to eliminate some of the values in each of the columns of multituples. This elimination may be based on both properties and data values. Note that at this stage only tuple-applicable properties may be used. For example, a possible strategy is to retain only A_1 values that are from contributions with *quality* above 0.3, to retain only A_2 values that are between 5 and 100, and to retain the most frequent A_3 value (the mode).

3. In the final stage the remaining alternative values in each of the attributes are *fused* into a single value. By forcing the resolution of every attribute to a single value we are ensuring that the mosaic instance of R becomes a simple relation instance. For example, a possible strategy is to combine the values of attribute A_1 in their average value.

4.2 The Resolve Statement

A *conflict resolution scheme* is a set of **resolve** statements, one for each global relation. In this section we describe the syntax and semantics of the **resolve** statement. In our discussion R is the global relation, and $A_1, ..., A_n$ are its attributes.

The **resolve** statement has the following overall structure

resolve {**permissive, restrictive**} R
if ϕ_1 **then** $strategy_1$
else if ϕ_2 **then** $strategy_2$
$\vdots$
else if ϕ_m **then** $strategy_m$
else $strategy_{(m+1)}$

The header clause **resolve** states the global relation name. The keyword **permissive** or **restrictive** indicates how comparisons with null values should be handled: **permissive**

treats these comparisons as *true*, thus retaining all questionable tuples, whereas **restrictive** treats them as *false*, thus discarding all questionable tuples.

The body of the statement is a structured "if then else" statement with predicates $\phi_1, ..., \phi_m$ and $m+1$ individual resolution strategies. This allows specifying different strategies for different "horizontal" slices of R, while guaranteeing that the strategies do not overlap (i.e., each slice is governed by a single strategy).

Each $strategy_i$ has the following structure:

select *value-fusion*
from *contribution-qualification*
where *value-elimination*

The three clauses of a strategy correspond to the three conflict resolution stages discussed in Section 4.1. The **from** clause is applied first. It prunes the set of available contributions to those that will be used to construct the present slice of R. The **from** clause comprises a *list* of individual qualifying conditions. Only contributions that satisfy all the conditions in the list are retained. Conditions have these two forms

$$\begin{array}{lll} property & \theta & threshold \\ !property & \theta & threshold \end{array}$$

where *property* is any property, θ is one of $=, \geq, \leq, >, <, \neq$, and *threshold* is either a constant or one of the keywords **max**, **min** or **avg**. A contribution satisfies a condition *property* θ *constant*, if the value of the contribution for that property stands in relation θ with the specified constant. When **max**, **min** or **avg** are substituted for the constant, the value of the property is compared, correspondingly, with the highest, lowest or average value of that property for all the contributions that possess it. The symbol ! indicates how to handle contributions that do not have the specified property: if the symbol is included, the property is required, and contributions that do not have it are disqualified; otherwise, contributions that do not have the specified property qualify. Note that not having the specified property is similar to having a null value for that property, with the outcome of the comparison being *maybe*. Our treatment here is consistent with our previous approach of providing user control over two alternative interpretations: including ! indicates a restrictive interpretation, whereas omitting it indicates a permissive interpretation. As an example, the **from** clause $!quality \geq .85,\ cost \leq 75$ will retain only contributions for which the data quality is known and is above .85, and the retrieval cost does not exceed 75. As another example, $timestamp =$ **max** will assure that the present slice of R is constructed from the most recent contribution only. [7]

The **where** clause is applied next. At this point, the present slice of R is a set of multituples, and the **where** clause is used to reduce the number of alternative values in each of the attributes, by specifying a condition for every attribute. Hence, the **where** clause is

[7] Note that it is possible to specify conditions that eliminate all contributions.

list of n attribute-specific conditions. Each such condition comprises any number of *basic* conditions, which are connected with **and** and **or** operators. A basic condition has one of these forms

$$\begin{array}{rcl} \mathit{property}(A_i) & \theta & \mathit{threshold} \\ A_i & \theta & \mathit{threshold} \\ \textbf{occur}(A_i) & \theta & \mathit{threshold} \\ \mathit{true} & & \end{array}$$

where θ and *threshold* are as before. The first form prunes multituples on the basis of a property. Note that the property must be tuple-applicable; for example, *timestamp(Address)* $\geq$ *990305* will retain only the values of *Address* whose timestamps are later than 990305. The second form prunes multituples based on the data values themselves; for example, *Age* = **min** will resolve the multiple *Age* values to the lowest value, and *Salary* $\geq$ *40,000* will discard all the salaries that are below 40,000. The third form prunes multituples on the basis of ratio of occurrence. For each different value of the attribute **occur** calculates its ratio of occurrence within a given multituple. For example, assume a multituple has 20 versions with a total of 5 different values in its *Telephone* attribute, as follows: t_1 and t_2 occur 6 times each, t_3 occurs 5 times, t_4 occurs twice, and t_5 occurs once. The condition **occur**(*Telephone*) = **max** will retain t_1 and t_2, whereas **occur**(*Telephone*) $\geq .3$ will also retain t_3. The fourth form is used when no pruning is desired for a particular attribute.

The **select** clause is applied last. At this point, the present slice of R may still have multiple values in some of its "tuples", and the **select** clause specifies how these values are to be fused into a single value. This clause is a list of n fusion instructions, one for each attribute. A fusion instruction has the form

$$\mathit{aggregate}(A_i)$$

where *aggregate* is either **min**, **max**, **avg**, **mode** or **random**. The first three aggregations are useful for fusing numeric values. **mode** and **random** can be used with non-numeric values as well; the former selects the most popular value, whereas the latter selects one of the alternatives at random. In the previous example, a possible **select** clause is **min**(*Age*), **avg**(*Salary*), **random**(*Telephone*). Conceivably, the options for fusing alternative values may be expanded with the addition of user-defined aggregate functions. One example would be a function that, given a set of different text strings, would identify some of the strings as misspelling of another string, and would consolidate the set with the correct string.

Finally, when the entire relation is to be governed by a single conflict resolution strategy, the **resolve** statement is simply

resolve {**permissive, restrictive**} R
strategy

With all its different features, the statement is versatile and powerful. It can be used to

express basic strategies quickly, or to design elaborate strategies. Two examples follow.

Consider a global relation *Employee* = {*Ssn, FirstName, LastName, Salary, Age*} with *Ssn* as its key [8] and these two contributions:[9]

$C_1 = (\pi_{Ssn,LastName,Salary,Age}\sigma_{Age>21}(Employee)$, "http://www.x.com/getDB.cgi?abc=1")
$C_2 = (\pi_{Ssn,FirstName,LastName,Age}\sigma_{Age<=65}(Employee)$, "http://www.y.com/getDB.cgi?abc=3")

Assume that C_1 has the properties $\{quality = .9, cost = 2\}$ and C_2 has the properties $\{quality = .8, timestamp = 981231\}$.

C_1 and C_2 slice *Employee* into three parts: $(Age \leq 21)$, $(Age > 21 \wedge Age \leq 65)$, and $(Age > 65)$. Because both contributions insert tuples into the second slice conflicts may arise within that slice.

Example. The following statement provides a single strategy for all of *Employee*.

resolve permissive *Employee*
select mode(*Ssn*), **random**(*FirstName*), **mode**(*LastName*), **min**(*Salary*), **max**(*Age*)
from *quality* $\geq .7$
where *true*, *true*, *true*, *Salary* ≥ 40000, *Age* $>$ **avg** $- 5$ **and** *Age* $<$ **avg** $+ 5$

It dictates that in the case of conflicts:

1. Data of low quality should be discarded (in this example both contributions qualify).
2. Within each multituple, low *Salary* values and extreme *Age* values should be discarded.
3. The remaining alternative values within each multituple should be fused to a single value as follows: the most frequently mentioned last name, a randomly chosen first name, the average salary, and the highest age. Since *Ssn* is the key there would only be one value in each multituple, and the choice of fusion function is irrelevant.

Note that tuples in which the value of *Age* is *null* would be inserted into all three slices, and would therefore participate in subsequent conflict resolution phases.

Example. This second statement provides two separate strategies for *Employee*.

[8] We assume that keys are used as the tuple-identification mechanism.

[9] For brevity we use only two contributions, but to demonstrate some of the possibilities, the example strategies shown below include features more appropriate for a large numbers of alternative values.

resolve restrictive *Employee*
if $Age > 40$
select mode(Ssn), **random**($FirstName$), **mode**($LastName$), **min**($Salary$), **max**(Age)
from $quality \geq .7$
where $true$, $true$, $true$, $Salary \geq 40000$, $Age >$ **avg** $- 5$ **and** $Age <$ **avg** $+ 5$
else
select mode(Ssn), **random**($FirstName$), **mode**($LastName$), **min**($Salary$), **max**(Age)
from $!cost < 15$
where $true$, $true$, **occur**($LastName$) $\geq .33$, $true$, $true$

Note that tuples in which the value of Age is *null* would be discarded from all four slices, and would therefore not participate in subsequent conflict resolution phases.

By providing two strategies, this statement refines the three-slice partition into a four-slice partition: ($Age \leq 21$), ($Age > 21 \ \wedge \ Age \leq 40$), ($Age > 40 \ \wedge \ Age \leq 65$), and ($Age > 65$). The first strategy governs the first two slices, and the second one governs the last two slices. However, conflicts are possible in second and third slices only. The first strategy is as before. The second strategy dictates that in case of conflicts

1. Data with high cost (or without cost information) should be discarded.
2. Within each multituple, values of $LastName$ that were "agreed upon" by less than a third of the contributions should be discarded.
3. The remaining alternative values within each multituple should be fused to a single value as follows: the most frequently mentioned last name, a randomly chosen first name, the average salary, and the highest age.

5 Comparison with Other Work

As mentioned in the introduction, there has been considerable work in the area of multidatabases. A comprehensive discussion of every project or product is beyond the scope of this paper. The discussion here is divided into two. First, we discuss four different integration systems, representing fairly different approaches, and compare them to Multiplex. None of these systems considers extensional inconsistencies ("too much data") and their handling of partial answers ("too little data") is fairly limited. Then, we discuss additional works that are related to the inconsistency resolution methods that were presented in this paper.

UniSQL [9] is an example of a multidatabase system based on a comprehensive mapping of its global database scheme to the component database schemes. UniSQL provides an exhaustive framework for handling schematic heterogeneity (i.e., intensional inconsistencies) among the participating databases. Its reliance on predefined, comprehensive mappings dictates that UniSQL may not be as suitable for ad-hoc integration, in which (1) relatively small

portions of the component sources are of interest (their entire schemes possibly being irrelevant, unavailable or incomprehensible), and (2) component sources change frequently, with new sources being added and existing sources undergoing structural changes, or becoming altogether obsolete.

The TSIMMIS project [8] is an example of a system that is based on mediators and wrappers. *Mediators* [17] are software modules designed to deal with representation and abstraction problems that occur when trying to use data and knowledge resources. Mediators are understood to be active and knowledge-driven. *Wrappers* [17] are simpler software interfaces that allow heterogeneous information sources to appear as if they conform to a uniform design or protocol. For example, a wrapper could be built to make a legacy database respond to a subset of SQL queries, as if it were a relational database. Multiplex makes fairly standard use of wrappers. With respect to mediators, a Multiplex query (a global view) may be considered a new "object". Its translation produces an ad-hoc "mediator", describing how the global object is to be constructed from the presently available sources. The advantage of such "dynamic mediation" are two: (1) Whereas with "static" mediators all integrated "objects' must be anticipated and predefined, in Multiplex an unlimited number of global objects may be defined spontaneously. (2) Static mediators need to be redefined whenever the available information sources change, whereas Multiplex only needs to have its mapping updated.

The approach of SIMS [3] to the integration problem is somewhat different. SIMS creates a *domain model* of the application domain, using a knowledge representation language to establish a fixed vocabulary describing objects in the domain, their attributes, and the relationships among them. Given a global query, SIMS identifies the sources of information that are required to answer the query and reformulates the query accordingly. SIMS is similar to Multiplex in that both do not rely on pre-programmed mediators, making the addition of new sources relatively simple. In both systems new sources have only to be *described* to the system. In SIMS, this description is in the knowledge representation language, using terms in the shared domain model; in Multiplex it is via pairs of equivalent views. Arguably, the SIMS descriptions are more demanding, but may allow the system to perform additional tasks. In contradistinction, Multiplex makes no claims of "intelligence"; it is a direct extension of relational model concepts, without the costs, risks, and possibly some benefits of a "knowledge-based" approach.

In many ways, the Information Manifold (IM) [2] is similar to SIMS. IM uses an object-relational model to integrate the various information sources, called *sites*. The individual sites are described and related to the global scheme, called the *world-view*, using the knowledge description language Classic. Like Multiplex, global query processing requires translation from the global set of relations to the set of available views. Like SIMS, and unlike Multiplex, the selection of relevant sites depends heavily on the quality of the site descriptions.

An early consideration of extensional inconsistencies may be found in [6], which describes Multibase, an early multidatabase system. Essentially, the method used in Multibase is to provide solutions to all anticipated inconsistencies in the design of the global schema.

The virtual database is thus guaranteed to be conflict-free. In contrast, the approach of Multiplex is much more dynamic, allowing different resolution strategies for different users or for different executions of a query. While Multiplex offers a powerful **resolve** statement, it should be possible to embed simple resolution strategies directly in the **from** clause of user queries.

The purpose of the Flexible relational model [1] is to represent the inconsistencies that are detected in a multidatabase environment, and preserve this information during query processing. This is achieved by a new relation structure and extensions to the relational algebra. The solution provides end users with complete information on inconsistencies, and leaves them to interpret it any way they see fit. In contrast, our goal here is to provide tools for resolving inconsistencies.

In the area of deductive databases, [15] develops methods for amalgamating inconsistent knowledge bases. Conflict resolution axioms are used to determine which facts should be removed from the amalgamation. These axioms are based on properties such as reliability or timestamps. However, it does not seem possible to remove information based on its contents, or to fuse alternative values into a single value.

One of the features of Multiplex is that when some of the information required for answering a query is unavailable, the system delivers a partial answer. A similar feature is described in [4], where in addition it is shown how the part answer which is presently unavailable can be calculated subsequently submitted as a complementary query.

6 Conclusion and Future Work

Multiplex is a system for integrating heterogeneous information sources. It facilitates the quick creation of ad-hoc virtual databases in networked environments that are highly dynamic and involve large numbers of possibly complex sources. An important feature of Multiplex is its handling of "troublesome" situations in which there is "too little data" (e.g., some sources are unavailable), or "too much data" (i.e., some sources are inconsistent).

Implementation. At the present, the Multiplex system is being upgraded from an earlier version described in [14] to the version that was described here. We mention here some of the architectural features common to both versions. Multiplex extends the query language of conjunctive queries with *aggregate functions*. A language based on conjunctive queries with aggregation provides a fairly powerful querying tool. The user interfaces include a simple window for entering global queries, as well as a query assistant for guiding users through the creation of global queries. All relations are assumed to have keys, which are used as the tuple identification mechanism. Multiplex employs the widely-used client-server architecture. The server is an application written in pure Java, whereas the client applications (e.g., the submission of global queries) are CGI-scripts which are usually invoked through the Multiplex user interface. All communication between the server, the clients and the member databases is carried on the Internet. With respect to heterogeneity, Multiplex retrieves information

from different kinds of sources, including: (1) relational (using Oracle), (2) object-oriented (using Ode), (3) simple files (using Unix shell scripts), and (4) Wide-area information services (WAIS, using SWISH 1.1 and WWWAIS 2.5) and (5) spreadsheets (Microsoft Excell's HTML output).

Null values. The virtual relations constructed by Multiplex may contain null values. These nulls originate either from actual nulls that were present in the contributions, or from the extension of contributed views with null columns. The processing of global queries on databases with nulls is an issue that has not been discussed in this paper. Essentially, our approach is to provide both a restrictive interpretation (in which null comparisons are interpreted as false) and a permissive interpretation (in which null comparisons are interpreted as true). This creates two answers: A *sound* answer which contains the tuples that are certain, and a *complete* answer which contains the tuples that are possible. The "real answer" is thus "sandwiched" between these sound and complete estimates.

Optimization. Multiplex tries to optimize global query processing by striving to minimize the amount of data extracted from the member databases. The challenge to the Multiplex optimizer is that member databases may not be able to process requests other than those that precisely match the views they promised. For example, assume a member database that contributes a large set of employee records, and consider a query about the salary of Jones. The optimizer might recognize that it is sufficient to issue a request for just one tuple, but the source might be unable to respond to any request other than for the entire set. One way to optimize requests to such sources is to embed them in appropriate *wrappers*.

Future directions. Finally, we mention several research problems that are still open. First, we are interested in deriving the properties of the answers assembled by Multiplex in response to user queries from the properties of the input sources that were used in generating these answers. The Multiplex model we described does not incorporate integrity constraints. The subject of global constraints, the mapping of global constraints to member constraints, and the use of global constraints in the construction of global relations requires further study. Another interesting issue is whether a global scheme is "covered" by a given set of contributing views. Fourth, we are interested in investigating the semantic distance among tuples as an alternative tuple identification mechanism [7].

References

[1] S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In *Proceedings of ICDE-95, the IEEE Computer Society Eleventh International Conference on Data Engineering (*Taipei, Taiwan, March 6–10*)*, pages 495–504, 1995.

[2] A. Levy an D. Srivastava and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2):121–143, September

1995.

[3] 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, June 1996.

[4] P. Bonnet and A. Tomasic. Partial answers for unavailable data sources. In *Proceedings of the FQAS-98,the Third International Conference on Flexible Query Answering Systems* Roskilde, Denmark, 13–15 May, Lecture Notes in Artificial Intelligence No. 1495, pages 43–54. Springer-Verlag, Berlin, Germany, 1998.

[5] A. Brodsky and A. Motro. The problem of optimal approximations of queries using views and its applications. Technical Report ISSE-TR-95-104, Department of Information and Software Systems Engineering, George Mason University, May 1995.

[6] U. Dayal. Query processing in a multidatabase system. In *Query Processing in Database Systems*, pages 81–108. Springer-Verlag, Berlin, Germany, 1984.

[7] P. Fankhauser and E.J. Neuhold. Knowledge based integration of heterogeneous databases. In *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)* (Lorne, Victoria, Australia, November 16-20), pages 155–175. North-Holland, 1993.

[8] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. In *Proceedings of NGITS-95, the Second International Workshop on Next Generation Information Technologies and Systems* (Naharia, Israel, June 27–29), pages 185–193, 1995.

[9] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18, 1991.

[10] P.-A. Larson and H. Z. Yang. Computing queries from derived relations. In *Proceedings of the Eleventh International Conference on Very Large Data Bases* (Stockholm, Sweden, August 21–23), pages 259–269, 1985.

[11] P.-A. Larson and H. Z. Yang. Computing queries from derived relations: Theoretical foundations. Technical Report CS-87-35, Department of Computer Science, University of Waterloo, August 1987.

[12] A. L. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries from views. In *Proceedings of PODS-95, the 14th Symposium on Principles of Database Systems*, pages 95–104, 1995.

[13] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[14] A. Motro. Multiplex: A formal model for multidatabases and its implementation. Technical Report ISSE-TR-95-103, Department of Information and Software Systems Engineering, George Mason University, March 1995.

[15] V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, June 1994.

[16] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, Maryland, 1982.

[17] G. Wiederhold. Glossary: Intelligent integration of information. *Journal of Intelligent Information Systems*, 6(2/3):281–291, June 1996.